Ατομική Διπλωματική Εργασία

Κινητικότητα στα Ασύρματα Δίκτυα Αισθητήρων

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΩΝΣΤΑΝΤΙΝΙΔΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Κινητικότητα στα Ασύρματα Δίκτυα Αισθητήρων

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΩΝΣΤΑΝΤΙΝΙΔΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

ΒΑΣΟΣ ΒΑΣΙΛΕΙΟΥ

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

Ιούνιος 2009

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου Δρ. Βασιλέιου Βάσο, για την υποστήριξη και την καθοδήγηση που μου πρόσφερε καθ' όλη την διάρκεια της εκπόνησης της διπλωματικής μου εργασίας, δίνοντας μου σημαντικές συμβουλές ώστε να καταστεί δυνατή η ολοκλήρωση αυτής της εργασίας, αλλά και για την επίτευξη μιας άψογης συνεργασίας. Ακόμη θα ήθελα να ευχαριστήσω τον διδακτορικό φοιτητή Κούτρουλλο Μάριο για την βοηθεια του στη συγκέντρωση του απαραίτητου υλικού για την υλοποίηση του πιλοτικού δικτύου. Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου που με στήριξε μέχρι το τέλος της διπλωματικής μου εργασίας.

Περίληψη

Τίτλος Διπλωματικής:	Κινητικότητα στα Ασύρματα Δίκτυα Αισθητήρων
Προπτυχιακός Φοιτητής:	Κωνσταντίνος Κωνσταντινίδης, Ατομική Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.
Διατριβή κατευθύνεται από:	Βάσος Βασιλείου, Λέκτορας, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Τα τελευταία χρόνια υπάρχει μια έντονη επιθυμία για ανάπτυξη αισθητήρων, οι οποίοι είναι ενσωματωμένοι πάνω σε κινητές πλατφόρμες όπως για παράδειγμα κινητά robots. Τέτοια είδη δικτύων είναι πολύτιμα σε περιπτώσεις όπου η παραδοσιακή ανάπτυξη σταθερών κόμβων αποτυγχάνει ή δεν είναι επιτρεπτή. Σε αυτή τη διπλωματική εργασία θα παρουσιάσουμε τη γενική θεωρία πίσω από τα ασύρματα δίκτυα αισθητήρων και θα εξηγήσουμε τι εννοούμε με τον όρο “κινητικότητα στα ασύρματα δίκτυα αισθητήρων”. Θα αναφέρουμε ζητήματα κινητικότητας που βρίσκονται υπό έρευνα και θα ασχοληθούμε με τα πιο γνωστά πρωτόκολλα κινητικότητας sink, ασύρματων αισθητήρων καθώς και με τα πρωτόκολλα ελέγχου κάλυψης στα ασύρματα δίκτυα αισθητήρων. Επίσης θα γίνει ανάπτυξη ενός αλγορίθμου ανίχνευσης και ελαχιστοποίησης τρυπών κάλυψης με την προσθήκη επιπρόσθετων κινητών κόμβων αισθητήρων, ο οποίος θα υλοποιηθεί σε πιλοτικό δίκτυο.

Περιεχόμενα

Εισαγωγή

1.1 Γενικά

Η γρήγορη ανάπτυξη στις τεχνολογίες ασύρματης επικοινωνίας και των ενσωματωμένων μικρό-αισθητήρων βοήθησε στη διάδοση των ασύρματων δικτύων αισθητήρων. Τέτοια δίκτυα μπορούν να έχουν πολλούς ασύρματους κόμβους αισθητήρων μικρού χρηματικού κόστους οι οποίοι είναι ικανοί να συλλέγουν, να αποθηκεύουν και να επεξεργάζονται πληροφορίες του περιβάλλοντος και ταυτόχρονα να επικοινωνούν με τους γειτονικούς τους κόμβους.

Τα ασύρματα δίκτυα αισθητήρων είναι πολυδύναμα, γιατί είναι ικανά να υποστηρίξουν πολλές διαφορετικές πραγματικές (real world) εφαρμογές και είναι πρόκληση για έρευνα και για επίλυση προβλημάτων

εφαρμοσμένης μηχανικής λόγω της μεγάλης τους ελαστικότητας. Συνεπώς, δεν υπάρχει κανένα σύνολο απαιτήσεων που ταξινομεί καλά όλα τα ασύρματα δίκτυα αισθητήρων και δεν υπάρχει μια ενιαία τεχνική λύση που καλύπτει το ολόκληρο διάστημα σχεδίου τους. Για παράδειγμα σε πολλά ασύρματα δίκτυα αισθητήρων ο κάθε κόμβος δεν μπορεί εύκολα να ενωθεί με ενσύρματη πηγή ενέργειας έτσι πρέπει να βασιστεί σε μπαταρίες. Σε αυτού του είδους τις εφαρμογές η ενεργειακή αποδοτικότητα οποιασδήποτε προτεινόμενης λύσης είναι ένας πολύ σημαντικός παράγοντας γιατί η μακροχρόνια λειτουργία είναι συνήθως επιθυμητή. Σε άλλες εφαρμογές, μπορεί να μην είναι ζήτημα η παροχή ηλεκτρικού ρεύματος, αλλά διαφορετικές μετρικές, όπως για παράδειγμα η ακρίβεια του υπολογισμού των αποτελεσμάτων ή το αποδεκτό μέγεθος και το κόστος ενός μεμονωμένου κόμβου.

1.2 Εφαρμογές και σχέδια αλληλεπίδρασης των Ασύρματων Δικτύων Αισθητήρων

Υπάρχει η κατάλληλη τεχνολογία αισθητήρων για πάρα πολλές φυσικές και μη φυσικές παραμέτρους, όπως για παράδειγμα αισθητήρας θερμοκρασίας, αισθητήρας υγρασίας, αισθητήρας ορατού ή και υπέρυθρου φωτός, αισθητήρας ήχου, αισθητήρας υπέρηχου, αισθητήρας δονήσεων, αισθητήρας πίεσης, χημικός αισθητήρας, αισθητήρας μηχανικής πίεσης, μαγνητικός αισθητήρας και αισθητήρας radar.

Μερικές από τις πλέον πιο γνωστές εφαρμογές στις οποίες χρησιμοποιούνται αισθητήρες είναι:

- Εφαρμογές βοήθειας σε περίπτωση καταστροφών.
- Εφαρμογές για έλεγχο και για χαρτογράφηση βιοποικιλότητας.
- Εφαρμογές για έξυπνα κτίρια.
- Εφαρμογές για τη διαχείριση διαφόρων εγκαταστάσεων που απαιτούν ασφάλεια.

- Εφαρμογές για επιτήρηση μηχανών και προληπτική συντήρηση.
- Εφαρμογές που αφορούν τη γεωργία.
- Εφαρμογές που αφορούν την ιατρική και την υγεία.
- Εφαρμογές λογιστικών.
- Τηλεπληροφορική (Telematics).

Στις περισσότερες από αυτές τις εφαρμογές υπάρχει μια καθαρή διαφορά μεταξύ των κόμβων που αισθάνονται τα δεδομένα (source) και των κόμβων στους οποίους θα σταλούν τα δεδομένα (sinks). Τα sinks μερικές φορές είναι μέρος του ίδιου του δικτύου αισθητήρων και κάποιες φορές δεν αποτελούν μέρος του δικτύου, αλλά είναι καθαρά συστήματα έξω από αυτό(π.χ PDA κ.α). Τις πλείστες φορές οι κόμβοι αισθητήρων (sources nodes) είναι αρκετά πιο πολλοί από τα sink nodes.

Τα πιο τυπικά σχέδια αλληλεπίδρασης μεταξύ κόμβων αισθητήρων και κόμβων sink είναι:

- **Event detection:** Οι κόμβοι αισθητήρων πρέπει να αναφέρουν στα sinks μόλις ανιχνεύσουν την πραγματοποίηση ενός συγκεκριμένου γεγονότος.
- **Periodic measurements:** Οι κόμβοι αισθητήρων ρυθμίζονται έτσι ώστε να στέλλουν περιοδικά τις μετρήσεις που ανιχνεύουν στα sinks.
- **Function approximation and edge detection:** Ο τρόπος που μια φυσική παράμετρος, όπως η θερμοκρασία, αλλάζει από μια τοποθεσία σε άλλη, μπορεί να θεωρηθεί ανάλογος της τοποθεσίας. Ένα ασύρματο δίκτυο αισθητήρων μπορεί να χρησιμοποιηθεί για να υπολογίσει τη σχέση της τοποθεσίας με τη θερμοκρασία χρησιμοποιώντας περιορισμένο αριθμό δειγμάτων που λαμβάνονται από κάθε μεμονωμένο κόμβο αισθητήρων.
- **Tracking:** Η πηγή ενός γεγονότος μπορεί να είναι κινητή (π.χ. ένας εισβολέας στα σενάρια επιτήρησης). Ένα ασύρματο δίκτυο

αισθητήρων μπορεί να χρησιμοποιηθεί για να αναφέρει τις αναπροσαρμογές στη θέση της πηγής γεγονότος στο sink, ενδεχομένως με τις εκτιμήσεις για την ταχύτητα και την κατεύθυνση. Χαρακτηριστικά οι κόμβοι αισθητήρων πρέπει να συνεργαστούν προτού να μπορέσουν οι αναπροσαρμογές να αναφερθούν στο sink.

Τα ασύρματα δίκτυα αισθητήρων κυμαίνονται από καλά οργανωμένα, με σταθερή ανάπτυξη των κόμβων αισθητήρων, μέχρι και σε τυχαία ανάπτυξη π.χ. ρίχνοντας μεγάλη ποσότητα κόμβων από ένα αεροπλάνο σε ένα δάσος το οποίο καίγεται. Ακόμη οι αισθητήρες μπορεί να είναι κινητοί για να αντισταθμιστεί η ανεπάρκεια στη διαδικασία επέκτασης με την κίνηση προς τις θέσεις όπου δεν υπάρχει κάλυψη, έτσι ώστε οι στόχοι τους να μπορούν να εκπληρωθούν καλύτερα. Οι κόμβοι μπορούν να γίνουν κινητοί εάν είναι συνδεδεμένοι με άλλα κινούμενα αντικείμενα και το δίκτυο πρέπει να προσαρμοστεί στη νέα τοποθεσία τους ανά πάσα στιγμή.

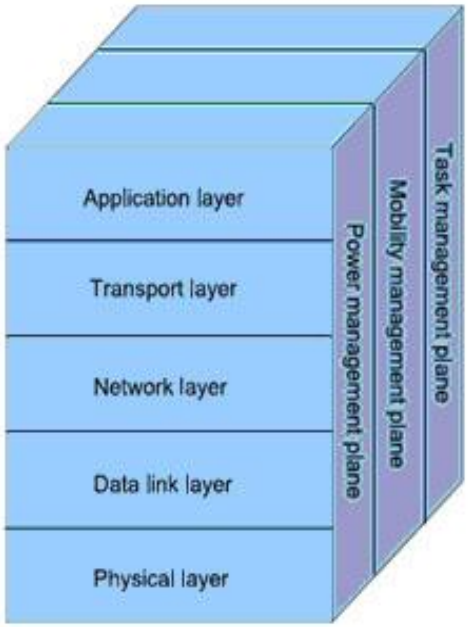
1.3 Σημαντικοί μηχανισμοί Ασύρματων Δικτύων Αισθητήρων

Μερικοί από τους μηχανισμούς που αποτελούν τυπικά κομμάτια του ασύρματου δικτύου αισθητήρων είναι:

- **Multihop wireless communication:** Η επικοινωνία μεταξύ κόμβων σε μεγάλες αποστάσεις είναι εφικτή μόνο αν χρησιμοποιηθεί μεγάλη ένταση μετάδοσης η οποία θα μειώσει σημαντικά το χρόνο ζωής του αισθητήρα (λόγω έλλειψης ενέργειας). Έτσι η χρήση ενδιάμεσων κόμβων για την μεταφορά των δεδομένων μπορεί να μειώσει την απαιτούμενη ένταση μετάδοσης.
- **Energy-efficient operation:** Για να έχει ο κόμβος μεγάλη διάρκεια ζωής, πρέπει να εφαρμοστούν λειτουργίες που δεν καταναλώνουν πολλή ενέργεια.

- **Auto-configuration:** Το ασύρματο δίκτυο αισθητήρων πρέπει να μπορεί να ρυθμίσει αυτόνομα τις περισσότερες λειτουργικές του παραμέτρους ανεξάρτητα από οποιεσδήποτε εξωτερικές ρυθμίσεις.
- **Collaboration and in-network processing:** Σε μερικές εφαρμογές ένας αισθητήρας από μόνος του είναι δύσκολο να αποφασίσει αν ένα γεγονός έχει συμβεί, έτσι πολλοί αισθητήρες πρέπει να συνεργαστούν για την ανίχνευση ενός γεγονότος και μόνο η ένωση των δεδομένων πολλών αισθητήρων παρέχει ικανοποιητικές πληροφορίες. Η πληροφορία επεξεργάζεται στο ίδιο το δίκτυο σε πολλές μορφές για να επιτευχθεί αυτή η συνεργασία, σε αντίθεση με το να μεταδίδει κάθε κόμβος όλα τα δεδομένα σε ένα εξωτερικό δίκτυο και να τα επεξεργάζεται ‘’στις άκρες ‘’ του δικτύου.
- **Data centric:** Σε ένα ασύρματο δίκτυο αισθητήρων, όπου οι κόμβοι έχουν αναπτυχθεί πλεοναστικά για την προστασία κατά τις αποτυχίας ενός κόμβου, ή για την βελτίωση των αποτελεσμάτων, η ταυτότητα ενός συγκεκριμένου κόμβου που μεταφέρει δεδομένα δεν έχει σημασία. Αυτό που έχει σημασία είναι οι ίδιες οι τιμές των μηνυμάτων και όχι ποιος κόμβος τα παρείχε.
- **Locality:** Κόμβοι οι οποίοι έχουν πολύ περιορισμένες προμήθειες, όπως για παράδειγμα μνήμη, πρέπει να προσπαθούν να περιορίσουν τις καταστάσεις που αποθηκεύουν κατά τη διάρκεια της επεξεργασίας του πρωτοκόλλου σε πληροφορίες που αφορούν μόνο τους άμεσους γείτονες. Με αυτό τον τρόπο το δίκτυο θα μπορεί να κλιμακωθεί σε μεγάλους αριθμούς από κόμβους χωρίς να χρειαστεί να βασίζεται σε πολυδύναμη επεξεργασία σε κάθε ένα κόμβο.
- **Exploit trade-offs:** Ένα ασύρματο δίκτυο αισθητήρων πρέπει να βασιστεί σε μεγάλο βαθμό στην εκμετάλλευση διαφόρων trade-offs μεταξύ αμοιβαία αντιφατικών στόχων που αφορούν το σχεδιασμό του συστήματος και των πρωτοκόλλων.

1.4 Τα layers στα Ασύρματα Δίκτυα Αισθητήρων



Σχήμα 1.1: Τα Layers στα Ασύρματα Δίκτυα Αισθητήρων [3]

Τα ασύρματα δίκτυα αισθητήρων είναι δομημένα σε layers για την σωστή οργάνωση και σχεδίαση πρωτοκόλλων [3].

- **Physical layer:** Το physical layer είναι υπεύθυνο για την επιλογή της συχνότητας, την παραγωγή συχνότητας μετάδοσης, την ανίχνευση σήματος, τη διαμόρφωση του και για την κρυπτογράφηση των δεδομένων. Μέχρι σήμερα, είναι από όλους αποδεκτές οι συχνότητες των 816 MHz, 915 MHz και 2.4 GHz ISM για τα ασύρματα δίκτυα αισθητήρων.
- **Data link layer:** Το data link layer είναι υπεύθυνο να κάνει multiplexing της ροής των δεδομένων, να ανιχνεύει τα data frames καθώς και για medium access και error control. Επίσης, διαφυλάσσει αξιόπιστες point to point και point to multipoint επικοινωνίες στο δίκτυο. Το πρωτόκολλο MAC, σε ένα ασύρματο multihop και αυτό-οργανωτικό δίκτυο αισθητήρων, έχει ως στόχους

τη δημιουργία της υποδομής του δικτύου και το δίκαιο και αποδοτικό διαχωρισμό των πόρων επικοινωνίας μεταξύ των κόμβων του δικτύου. Οι δύο γνωστοί μέθοδοι για error control είναι οι forward error correction (FEC) και automatic repeat request (ARQ). Η χρησιμότητα του ARQ σε multihop δίκτυα αισθητήρων είναι περιορισμένη από το επιπλέον κόστος ενέργειας αναμετάδοσης και το overhead. Από την άλλη, η πολυπλοκότητα αποκωδικοποίησης του FEC είναι μεγαλύτερη αφού απαιτείται η ενσωμάτωση ικανοτήτων error control. Λαμβάνοντας αυτό υπόψη, οι απλοί κώδικες error control με μικρή πολυπλοκότητα κωδικοποίησης και αποκωδικοποίησης παρουσιάζονται ως οι καλύτερες λύσεις στα ασύρματα δίκτυα αισθητήρων.

- **Network layer:** Παραδοσιακές ad-hoc τεχνικές δρομολόγησης συνήθως δεν ταιριάζουν στις απαιτήσεις των ασύρματων δικτύων αισθητήρων. Το network layer των δικτύων αισθητήρων είναι σχεδιασμένο σύμφωνα με τις πιο κάτω αρχές:
    - Πρέπει να διαχειρίζονται αποδοτικά τα διαθέσιμα ποσά ενέργειας τους.
    - Πρέπει να είναι data-centric.
    - Η συνάθροιση των δεδομένων είναι χρήσιμη μόνο όταν δεν εμποδίζεται η συνεργασία μεταξύ των κόμβων αισθητήρων.
    - Σε ένα ιδανικό δίκτυο αισθητήρων, οι διευθύνσεις των κόμβων είναι attribute-based και δεν γνωρίζουν την ακριβή τοποθεσία τους οι κόμβοι.

Τα δρομολόγια τα οποία είναι energy – efficient μπορούν να βρεθούν βάση της διαθέσιμης εναπομένουσας ενέργειας σε κάθε κόμβο (available power - PA) ή της ενέργειας που απαιτείται για τη μετάδοση των δεδομένων μέσω των δρομολογίων.

Οι πιο γνωστοί αλγόριθμοι για την επιλογή του πιο αποδοτικού δρομολογίου όσον αφορά την ενέργεια είναι οι πιο κάτω:

- **Maximum PA route:** Επιλέγεται το δρομολόγιο το οποίο έχει τη μεγαλύτερη συνολική εναπομένουσα ενέργεια, η οποία υπολογίζεται με το άθροισμα της εναπομένουσας ενέργειας του κάθε κόμβου ο οποίος ανήκει στο συγκεκριμένο δρομολόγιο.
- **Minimum energy (ME) route:** Επιλέγεται το δρομολόγιο το οποίο απαιτεί την ελάχιστη ενέργεια για τη μεταφορά των πακέτων μεταξύ των sinks και των κόμβων αισθητήρων.
- **Minimum hop route (MH):** Επιλέγεται το δρομολόγιο το οποίο θα έχει τον ελάχιστο αριθμό hops μέχρι το sink.
- **Maximum minimum PA note route:** Επιλέγεται το δρομολόγιο του οποίου η ελάχιστη εναπομένουσα ενέργεια θα είναι μεγαλύτερη από την ελάχιστη εναπομένουσα ενέργεια των υπολοίπων δρομολογίων.

Μια από τις υπευθυνότητες του network layer είναι να παρέχει διασύνδεση με εξωτερικά δίκτυα (πχ. άλλα δίκτυα αισθητήρων ή το διαδίκτυο). Σε ένα σενάριο το sink μπορεί να χρησιμοποιηθεί ως το gateway προς άλλα δίκτυα.

- **Transport Layer:** Αυτό το layer είναι χρήσιμο κυρίως όταν το σύστημα είναι σχεδιασμένο έτσι ώστε να μπορεί να έχει πρόσβαση στο διαδίκτυο ή σε άλλα εξωτερικά δίκτυα. Μια προσέγγιση παρόμοια με αυτήν του TCP μπορεί να φανεί χρήσιμη για την επικοινωνία με άλλου είδους δίκτυα όπως το διαδίκτυο. Η επικοινωνία μεταξύ κόμβων του δικτύου δεν μπορεί να υποστηριχθεί με πρωτόκολλα των οποίων η προσέγγιση είναι παρόμοια με του πρωτοκόλλου TCP, λόγω της περιορισμένης μνήμης που διαθέτουν οι κόμβοι αισθητήρων.
- **Application Layer:** Παρόλο που έχουν οριστεί πολλές περιοχές εφαρμογών για ασύρματα δίκτυα αισθητήρων, δεν έχει ερευνηθεί μεγάλος αριθμός πρωτοκόλλων για το application layer. Μερικά

πρωτόκολλα του application layer τα οποία βρίσκονται υπό έρευνα είναι:

- Sensor Management Protocol (SMP)
- Task Assignment and Data advertisement Protocol (TADAP)
- Sensor Query and Data Dissemination Protocol (SQDDP)

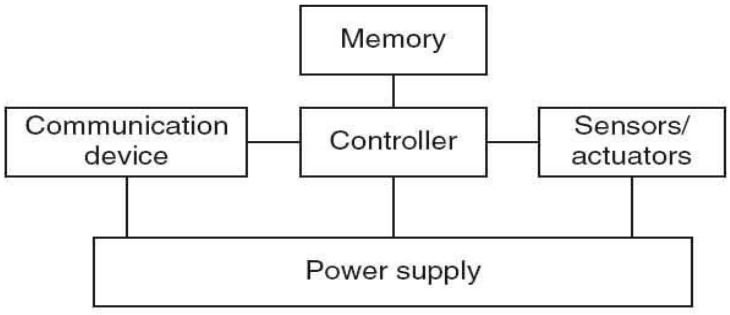
- **Power management plane:** Είναι υπεύθυνο για τη διαχείριση της ενέργειας ενός κόμβου αισθητήρα. Για παράδειγμα, για την αποφυγή παραλαβής διπλών μηνυμάτων, ο κόμβος αισθητήρα μπορεί να κλείσει το δέκτη του μετά από παραλαβή ενός μηνύματος από κάποιο από τους γείτονες του. Ακόμη, όταν ο κόμβος αισθητήρα δεν έχει μεγάλα αποθέματα ενέργειας, ενημερώνει τους γείτονες του για να μην συμμετάσχει σε μελλοντική δρομολόγηση πακέτων και έτσι η εναπομένουσα ενέργεια που διαθέτει θα χρησιμοποιείται μόνο για την ανίχνευση γεγονότων.
- **Mobility management plane:** Εντοπίζει και καταγράφει την κίνηση των κόμβων αισθητήρων έτσι ώστε η δρομολόγηση πίσω προς το χρήστη να είναι πάντα δυνατή και έτσι ώστε οι κόμβοι να γνωρίζουν ποιοι είναι οι γείτονές τους.
- **Task management plane:** Ισοζυγίζει και προγραμματίζει τις διεργασίες για αίσθηση σε μια συγκεκριμένη περιοχή.

Η ευελιξία, η μεγάλη ανοχή σε λάθη, η υψηλή αξιοπιστία αίσθησης, το μικρό κόστος και η γρήγορη ανάπτυξη ενός δικτύου αισθητήρων είναι μερικοί από τους σημαντικότερους παράγοντες που δημιουργούν πολλές καινούριες και συναρπαστικές περιοχές για ανάπτυξη εφαρμογών. Σύντομα αυτό το μεγάλο εύρος εφαρμογών θα κάνει τα ασύρματα δίκτυα αισθητήρων μέρος της καθημερινής μας ζωής.

1.5 Κύρια χαρακτηριστικά ενός Ασύρματου κόμβου Αισθητήρα

Ένας βασικός κόμβος αισθητήρα αποτελείται από 5 κύρια χαρακτηριστικά σύμφωνα με το [27]:

- **Controller:** Ένας controller για να επεξεργαστεί όλα τα σχετικά στοιχεία, ικανός για την εκτέλεση αυθαίρετου κώδικα.
- **Memory:** Μια ορισμένη ποσότητα μνήμης για να αποθηκεύσει τα προγράμματα και τα ενδιάμεσα αποτελέσματα (συνήθως διαφορετικά είδη μνήμης χρησιμοποιούνται για τα προγράμματα και τα δεδομένα).
- **Sensors and actuators:** Η πραγματική διεπαφή με τον φυσικό κόσμο, δηλαδή συσκευές που μπορούν και παρατηρούν ή να ελέγχουν φυσικές παραμέτρους του περιβάλλοντος.
- **Communication:** Για να μετατραπούν οι κόμβοι σε ένα δίκτυο, χρειάζεται μια συσκευή για αποστολή και παραλαβή πάνω από ένα ασύρματο κανάλι.
- **Power supply:** Δεδομένου ότι καμία δεμένη παροχή ηλεκτρικού ρεύματος δεν είναι διαθέσιμη, κάποια μορφή μπαταριών είναι απαραίτητη για να παρέχει την ενέργεια. Μερικές φορές, κάποια μορφή επαναφόρτισης με τη λήψη της ενέργειας από το περιβάλλον είναι επίσης διαθέσιμη.



Σχήμα 1.2: Τα κύρια χαρακτηριστικά ενός Ασύρματου κόμβου Αισθητήρα[27]

1.6 Οικογένεια κόμβων Αισθητήρων Mica

Πρωτοκατασκευάστηκαν στα τέλη του 1990 στο πανεπιστήμιο της Καλιφόρνιας στο Berkeley σε συνεργασία με την Intel. Είναι γνωστοί με το όνομα Mica motes και υπάρχουν διάφορα versions (Mica, Mica2, Mica2Dot, MicaZ). Είναι εμπορικά διαθέσιμα μέσω της εταιρίας Crossbow. Το TinyOs είναι συνήθως το λειτουργικό σύστημα για αυτούς τους κόμβους. Όλα τα boards αποτελούνται από ένα microcontroller το οποίο ανήκει στην οικογένεια Atmel, ένα απλό radio modem (συνήθως ένα TR 1000 από την RFM) και πολλαπλές ενώσεις προς τον έξω κόσμο. Επιπρόσθετα, είναι δυνατόν να ενωθεί με επιπλέον sensor boards επιτρέποντας ευρύτερο πεδίο για εφαρμογές και πειράματα. Οι sensors είναι ενωμένοι με τον controller με I2C bus ή με SPI (ανάλογα με το version).

Σχήμα 1.3: Κόμβοι Αισθητήρων Mica

1.7 Κόμβοι Αισθητήρων TelosB

Οι κόμβοι αισθητήρων TelosB παρέχουν ασύρματη επικοινωνία στα 2.4 GHz, καθώς και επικοινωνία με IEEE/ZigBee 802.15.4. Χρησιμοποιούνται κυρίως σε ασύρματα δίκτυα αισθητήρων χαμηλής ενέργειας. Έχουν την δυνατότητα να προγραμματίζονται μέσω USB το οποίο μπορεί να χρησιμοποιηθεί και για serial communication. Για την ασύρματη επικοινωνία χρησιμοποιούν ένα δέκτη με chip της Chipcon, τον

CC2420 IEEE 802.15.4 (standard-compliant radio transceiver). Ο microcontroller των telosb είναι ο TI-MSP430 από την Texas Instruments. Οι αισθητήρες που χρησιμοποιούν (όπως και η αντένα τους) είναι ενσωματωμένοι πάνω στο board τους σε αντίθεση με τους αισθητήρες micaZ, των οποίων οι αισθητήρες βρίσκονται σε ξεχωριστό sensorboard.

Σχήμα 1.4: Κόμβος Αισθητήρα TelosB

1.8 Quality of Service και αποδοτικότητα της ενέργειας στα Ασύρματα Δίκτυα Αισθητήρων

Τα ασύρματα δίκτυα αισθητήρων διαφέρουν από άλλα δίκτυα επικοινωνίας κυρίως στον τρόπο παροχής υπηρεσιών που προσφέρουν. Αυτού του είδους τα δίκτυα ουσιαστικά μετακινούν μόνο bits από τον ένα τόπο στον άλλο. Επομένως απαιτούνται high level QoS που αντιστοιχούν σε χαρακτηριστικά QoS άλλων συμβατικών δικτύων. Τα πιο βασικά χαρακτηριστικά QoS είναι τα ακόλουθα όπως παρουσιάζονται στο [27]:

- **Event detection/reporting probability:** Η πιθανότητα να συμβεί ένα γεγονός και να μην καταγραφεί ή να μην αναφερθεί σε ένα sink.
- **Event classification error:** Τα γεγονότα δεν πρέπει μόνο να ανιχνευτούν αλλά και να κατηγοριοποιηθούν. Επομένως το σφάλμα στην κατηγοριοποίηση πρέπει να είναι μικρό.
- **Event detection delay:** Η καθυστέρηση μεταξύ του χρόνου ανίχνευσης ενός γεγονότος και αναφοράς του στα sinks.

- **Missing reports:** Σε εφαρμογές που απαιτούν περιοδικές ενημερώσεις η πιθανότητα για μη παραδοτέα ενημέρωση πρέπει να είναι μικρή.
- **Approximation accuracy:** Για εφαρμογές υπολογισμού ποιο είναι το μέσο / μέγιστο απόλυτο ή σχετικό σφάλμα σε σχέση με το πραγματικό αποτέλεσμα.
- **Tracking accuracy:** Οι εφαρμογές παρακολούθησης – καταδίωξης πρέπει να μην χάσουν το αντικείμενο το οποίο παρακολουθούν και η τοποθεσία του που αναφέρουν πρέπει να είναι όσο το δυνατόν πιο κοντά στην πραγματική και το σφάλμα να είναι μικρό.

Η ενέργεια είναι πολύτιμη πηγή σε ένα ασύρματο δίκτυο αισθητήρων και η εξοικονόμηση ενέργειας θεωρείται ένας βασικός στόχος. Διάφορες μετρικές για την κατανάλωση ενέργειας είναι οι πιο κάτω:

- **Energy per correctly received bit:** Η ποσότητα ενέργειας που καταναλώνεται για τη μεταφορά ενός bit δεδομένων από το source στο sink
- **Energy per reported (unique) event:** Η μέση ποσότητα ενέργειας που καταναλώνεται για την αναφορά ενός γεγονότος. Έχοντας υπόψη ότι το ίδιο γεγονός μπορεί να αναφερθεί από πολλά sources ταυτόχρονα, συνηθίζεται να γίνεται κοινωνικοποίηση της μετρικής και λαμβάνονται υπόψη μόνο τα μοναδικά γεγονότα.
- **Delay/energy trade-offs:** Μερικές εφαρμογές πρέπει όταν ανιχνεύσουν ένα γεγονός, να το στείλουν ‘επειγόντως’ στο sink. Αυτό απαιτεί αύξηση της κατανάλωσης της ενέργειας των κόμβων για την ταχεία αποστολή της αναφοράς στο sink.
- **Network lifetime:** Ο χρόνος που το δίκτυο είναι λειτουργικό, δηλαδή μπορεί να ολοκληρώσει την αποστολή του με επιτυχία. Το δίκτυο μπορεί να θεωρηθεί μη λειτουργικό, είτε όταν πάψει να λειτουργεί ο πρώτος κόμβος, είτε αν 50% (ή ένα άλλο ποσοστό) των κόμβων του ξεμείνουν από ενέργεια, είτε αν το δίκτυο διασπαστεί σε

μικρότερα ανεξάρτητα κομμάτια τα οποία δεν επικοινωνούν μεταξύ τους γιατί ο κόμβος που ενώνει τα κομμάτια αυτά είναι νεκρός. Μέσα σε αυτή την κατηγορία μπορεί να θεωρηθεί και ο υπολογισμός του χρόνου μέχρι το δίκτυο να πάψει να καλύπτει ένα κομμάτι της περιοχής για την οποία είναι υπεύθυνο.

Όλες αυτές οι μετρικές μπορούν να υπολογιστούν μόνο υπό κάποιες υποθέσεις που αφορούν τα χαρακτηριστικά κατανάλωσης ενέργειας για τον κάθε κόμβο, το πραγματικό φορτίο το οποίο μπορεί να αντέξει το δίκτυο και την συμπεριφορά του καναλιού των συχνοτήτων που επικοινωνούν οι κόμβοι.

1.9 Προκλήσεις των Ασύρματων Δικτύων Αισθητήρων που πρέπει να αντιμετωπιστούν

- **Χαμηλή κατανάλωση ενέργειας:** Τα ασύρματα δίκτυα αισθητήρων έχουν πολύ περιορισμένη πηγή ενέργειας. Ακόμα και αν είναι επαναφορτιζόμενα χρειάζεται να περιορίζουν την κατανάλωση ενέργειας όταν η ενέργεια είναι λιγοστή (π.χ τη νύχτα για κόμβους οι οποίοι επαναφορτίζονται με ηλιακή ενέργεια). Εφόσον η εξάντληση της ενέργειας και ο “θάνατος” μερικών κόμβων είναι δεδομένος, είναι σημαντικό, οι διάφορες εφαρμογές να δημιουργούνται έτσι ώστε να δουλεύουν αποδοτικά καταναλώνοντας όσο το δυνατόν λιγότερη ενέργεια. Επίσης, στις εφαρμογές, μπορούν να ενσωματωθούν τεχνικές για συλλογή ενέργειας από το περιβάλλον.
- **Περιορισμένες Δυνατότητες:** Εφόσον το μέγεθος των κόμβων αισθητήρων είναι αρκετά μικρό και πρέπει να είναι αρκετά φτηνοί σε κόστος, η υπολογιστική τους δυνατότητα και οι άλλες τους δυνατότητες είναι περιορισμένες.
- **Υψηλά κατανεμημένα:** Τέτοιου είδους δίκτυα αναπτύσσονται σε μεγάλους αριθμούς. Μπορούν να θεωρηθούν ως υψηλά

κατανεμημένα συστήματα. Είναι πολύ σημαντικό όλοι οι σχεδιασμοί των δικτύων να είναι scalable. Εφόσον οι διαθέσιμες δυνατότητες ενός κόμβου αισθητήρα είναι περιορισμένες, η αποδοτική λειτουργία τους εξαρτάται από τον αριθμό των κόμβων. Οι κόμβοι αισθητήρων αναπτύσσονται σε χιλιάδες και απαιτούν τη χρήση πολλαπλών κατανεμημένων αλγόριθμων για να λειτουργήσουν βέλτιστα.

- **Ανάπτυξη και κάλυψη:** Μια από τις αρχικές προκλήσεις στα μεγάλα δίκτυα αισθητήρων είναι η ανάπτυξη τους που έχει ως στόχο την αποτελεσματικότητα και τη λειτουργικότητά τους. Οι κόμβοι πρέπει να αναπτυχθούν στρατηγικά έτσι ώστε να μπορούν να ανιχνεύσουν τα περισσότερα γεγονότα.
- **Ανάγκη για Ρύθμιση των μετρήσεων:** Λόγω του χαμηλού τους κόστους οι κόμβοι αισθητήρων πιθανόν να έχουν σφάλματα. Έτσι είναι πολύ σημαντικό να εφαρμοστούν μέθοδοι για κατανεμημένη ρύθμιση των μετρήσεων των κόμβων αισθητήρων περιοδικά. Η χρήση τεχνικών πιθανοτήτων για την διόρθωση από τα σφάλματα βοηθά στην δημιουργία πιο ευέλικτων αλγορίθμων.
- **Βασικές υπηρεσίες συστήματος:** Εκτός από την ανάπτυξη του δικτύου, υπάρχει ένα σύνολο από υπηρεσίες του συστήματος, οι οποίες πρέπει να υλοποιηθούν ώστε το δίκτυο αισθητήρων να δουλεύει συνεκτικά. Μερικές από αυτές είναι:

    - Το localization, όπου κάθε κόμβος πρέπει να είναι ικανός να υπολογίζει την περιοχή του, είτε globally, είτε τοπικά.
    - Ο συγχρονισμός, όπου όλοι οι κόμβοι του δικτύου πρέπει να είναι συγχρονισμένοι γιατί συνήθως θα παράγουν σειρές από διαδοχικά δεδομένα.
    - Η δρομολόγηση, οπού οι κόμβοι αισθητήρων, οι οποίοι είναι τυχαία διασκορπισμένοι στο περιβάλλον, πρέπει να χρησιμοποιούν ένα αποδοτικό και αξιόπιστο δρομολόγιο για την μεταφορά των γεγονότων που έχουν αισθανθεί από τις

διάφορες περιοχές του δικτύου σε ένα κεντρικό σημείο (sink).

- **Δυναμικότητα του δικτύου:** Λόγω των παραλλαγών στο εύρος επικοινωνίας και του συχνού 'θανάτου' ενός κόμβου που οφείλεται στην εξάντλησης της ενέργειάς του και άλλες περιβαλλοντολογικές αιτίες, δημιουργούνται μεγάλες παραλλαγές στην τοπολογία του δικτύου. Γι αυτό το λόγο οι εφαρμογές που τρέχουν στα δίκτυα αυτά, πρέπει να είναι ευέλικτες στις διάφορες παραλλαγές της τοπολογίας τους.
- **Χρονική και Τοπική Ανωμαλία:** Λόγω του γεγονότος ότι οι περιοχές ενδιαφέροντος ενός ασύρματου δικτύου αισθητήρων δεν είναι ομοιόμορφα κατανεμημένες και λόγω της μορφολογίας της περιοχής και άλλων περιβαλλοντικών παραγόντων, καθιστούν αδύνατη την ομοιόμορφη τοπολογία των δικτύων αυτών. Επομένως παρατηρείται μια ανωμαλία στη δειγματοληψία των ασύρματων δικτύων αισθητήρων και σε χώρο και σε χρόνο.
- **Data – centric Naming:** Λόγω της αναξιόπιστης φύσης των ασύρματων δικτύων αισθητήρων και της περιορισμένης υπολογιστικής τους δύναμης, το σύνολο των δεδομένων έχει περισσότερο ενδιαφέρον παρά από τις μετρήσεις ενός συγκεκριμένου κόμβου. Γι αυτό το λόγο δύνονται ονόματα στα δεδομένα παρά στους κόμβους αισθητήρων.
- **Ταυτοχρονία:** Σε τέτοιου είδους δίκτυα μπορεί να παρατηρηθεί μεγάλη συμφόρηση, όταν ανιχνεύεται ένα γεγονός. Έτσι, χρειάζονται πολλές παράλληλες διεργασίες για το χειρισμό των δεδομένων σε περιόδους συμφόρησης. Ένας από τους σχεδιαστικούς στόχους του λειτουργικού συστήματος TinyOs ήταν η δυνατότητα εκτέλεσης ταυτοχρονίας με τη χρήση 'ελαφρών' threads.

Κεφαλαιο 2

Κινητικότητα στα Ασύρματα Δίκτυα Αισθητήρων

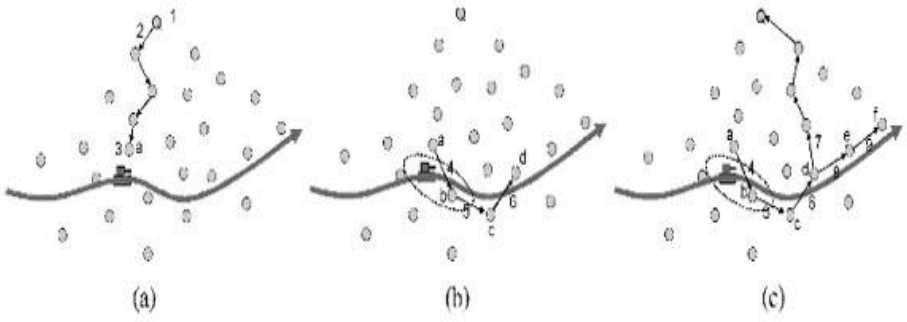
2.1 Γενικά

Πρόσφατα μια νέα κατηγορία για σημαντικές εφαρμογές δικτύων αισθητήρων εμφανίζεται όπου η κινητικότητα είναι θεμελιώδες χαρακτηριστικό για τα συστήματα που εξετάζονται. Σε τέτοιες εφαρμογές οι αισθητήρες συνάπτονται πάνω σε αυτοκίνητα, robots, ζώα ή ανθρώπους που κινούνται γύρω σε μεγάλες γεωγραφικές περιοχές. Η ανταλλαγή δεδομένων μεταξύ μεμονωμένων αισθητήρων και κόμβων υποδομής θα δημιουργήσουν νέες εφαρμογές, όπως εφαρμογές κυκλοφορίας, εφαρμογές παρακολούθησης της άγριας ζωής, έξυπνα σπίτια και νοσοκομεία και εφαρμογές για έλεγχο της μόλυνσης.

2.2 Μορφές Κινητικότητας

Στα ασύρματα δίκτυα αισθητήρων η κινητικότητα μπορεί να εμφανιστεί σε τρεις μορφές συμφωνα με το [27] :

1. **Node mobility:** Οι ασύρματοι κόμβοι αισθητήρων είναι καθεαυτό κινητοί. Το δίκτυο πρέπει να αναδιοργανώνεται αρκετά συχνά για να μπορεί να λειτουργήσει σωστά.
2. **Sink mobility:** Τα sinks είναι κινητά. Σε απλές περιπτώσεις το sink πρέπει να αλληλεπιδρά με το δίκτυο και να τελειώσει την αλληλεπίδραση του προτού προχωρήσει.
3. **Event mobility:** Σε εφαρμογές όπως η ανίχνευση γεγονότος και γενικά σε εφαρμογές καταδίωξης, η αιτία που προκαλεί το γεγονός ή το καταδιωκόμενο αντικείμενο μετακινείται. Σε τέτοιου είδους σενάρια, είναι σημαντικό να καλύπτεται συνέχεια το παρατηρούμενο γεγονός από ένα ικανοποιητικό αριθμό αισθητήρων. Όπου υπάρχει υψηλότερη δραστηριότητα, οι αισθητήρες ενεργοποιούνται γύρω από το αντικείμενο, για να το παρατηρήσουν. Ακολούθως, αφού τελειώσουν, αλλάζουν από την ενεργή κατάσταση σε κατάσταση ύπνου για εξοικονόμηση ενέργειας. Όσο μετακινείται μέσα στο δίκτυο το αντικείμενο που προκαλεί το γεγονός, συνοδεύεται από μια περιοχή με δραστηριότητα των κόμβων που βρίσκονται γύρο από αυτό.



Σχήμα 2.1: Κινητικότητα γεγονότος στα Ασύρματα Δίκτυα Αισθητήρων [5]

2.3 Δυνατότητες ελεγχόμενης κινητικότητας

Η ελεγχόμενη κινητικότητα στα ασύρματα δίκτυα αισθητήρων επιτρέπει τη δημιουργία ενός νέου μεγάλου συνόλου από δυνατότητες. Η ικανότητα

της ενεργούς αλλαγής της τοποθεσίας μπορεί να λύσει πολλές από τις σχεδιαστικές προκλήσεις των στατικών ασύρματων δικτύων αισθητήρων.

Πρέπει να διευκρινιστεί ότι υπάρχει διαφορά σε σχέση με τα Mobile Adhoc Networks(MANETS). Τα MANETS θεωρούν την κινητικότητα ως δεδομένη και μελετούν τις επιδράσεις της κινητικότητας χωρίς να έχουν κανένα έλεγχο πάνω της. Από την άλλη, η ελεγχόμενη ή η ρομποτική (με την χρήση robots) κινητικότητα είναι η ικανότητα της σκόπιμης κινητικότητας.

Πλεονεκτήματα της ελεγχόμενης κινητικότητας είναι:

- **Ανάπτυξη δικτύου:** Η ελεγχόμενη κινητικότητα μπορεί να χρησιμοποιηθεί έτσι ώστε να τοποθετηθούν οι αισθητήρες σε βέλτιστες τοποθεσίες π.χ. μεγιστοποιώντας την κάλυψη της περιοχής του δικτύου.
- **Προσαρμοστική Δειγματοληψία:** Η ελεγχόμενη κινητικότητα μπορεί να χρησιμοποιηθεί για να περιοριστούν ως ένα βαθμό τα προβλήματα που δημιουργούνται λόγω της Χρονικής και Τοπικής Ανωμαλίας που παρουσιάζονται στα στατικά ασύρματα δίκτυα αισθητήρων. Μπορούν να εφαρμοστούν στρατηγικές προσαρμοστικής δειγματοληψίας για να αντιμετωπιστούν οι χωρικές ανωμαλίες. Όμως, για την αντιμετώπιση των χρονικών ανωμαλιών χρειάζονται πιο πολύπλοκοι μηχανισμοί.
- **Τοπολογική Προσαρμοστικότητα:** Γίνεται καλύτερος έλεγχος όσον αφορά την τοπολογία του δικτύου. Οι διάφορες μεταβολές στο περιβάλλον αλλάζουν το ρυθμό παραγωγής των δεδομένων των κόμβων αισθητήρων με αποτέλεσμα τη μεταβολή και της κυκλοφοριακής συμφόρησης στο δίκτυο. Σε περιπτώσεις μεγάλης κυκλοφοριακής συμφόρησης τα στατικά δίκτυα πιθανών να μην μπορούν να καλύψουν τις νέες απαιτήσεις κυκλοφορίας. Με την κινητικότητα των αισθητήρων το δίκτυο μπορεί να προσαρμοστεί στις δυναμικές καταστάσεις κατά το χρόνο εκτέλεσης.

- **Επιδιόρθωση δικτύου:** Πολλές φορές, μπορεί να δημιουργηθούν τρύπες στην κάλυψη του δικτύου λόγω θανάτου συγκεκριμένων κόμβων, είτε από έλλειψη ενέργειας των κόμβων, είτε από καταστροφής τους. Η τρύπα αυτή μπορεί να αποκατασταθεί προγραμματίζοντας κινητούς κόμβους να μετακινηθούν σε αυτές τις περιοχές έτσι ώστε να επιδιορθωθεί το δίκτυο.
- **Αύξηση της χωρητικότητας καναλιού του δικτύου:** Η χωρητικότητα μεταφοράς δεδομένων αυξάνεται με τη χρήση κινητικότητας. Με τη χρήση κινητών κόμβων ως δρομολογητών, ο χρόνος μεταφοράς δεδομένων από έναν κόμβο σε έναν άλλο κόμβο είναι μικρότερος από το χρόνο μεταφοράς μέσω ασύρματου μέσου.
- **Αύξηση της ενεργειακής χωρητικότητας του δικτύου και μείωση λαθών δεδομένων:** Οι κινητοί κόμβοι μπορούν να χρησιμοποιηθούν ως μεταφορικό μέσο για μεγάλους όγκους δεδομένων. Αυτό έχει ως αποτέλεσμα την αύξηση του goodput του δικτύου. Αυτό συμβαίνει, γιατί η μνήμη είναι λιγότερο ευαίσθητη σε λάθη παρά το ασύρματο μέσο. Επίσης μειώνεται το bandwidth και η ενέργεια που απαιτείται για την αποστολή πακέτων που έχουν ξανασταλθεί.
- **Συγκομιδή ενέργειας (Energy Harvesting):** Μπορούν να χρησιμοποιηθούν μέθοδοι κατά τις οποίες οι κινητοί κόμβοι αισθητήρων θα μεταφέρουν ενέργεια από περιοχές πλούσιες σε ενέργεια (π.χ. ηλιακή ενέργεια) σε άλλες περιοχές του δικτύου με ελάχιστα διαθέσιμα ποσά ενέργειας.
- **Localization:** Είναι δυνατό να τροποποιηθούν μερικοί αλγόριθμοι που έχουν αναπτυχθεί για κατανεμημένο localization σε τομείς ρομποτικής για να χρησιμοποιηθούν από κινητούς κόμβους αισθητήρων. Μπορούν επίσης να δημιουργηθούν υβριδικοί αλγόριθμοι οι οποίοι θα χρησιμοποιούνται και από στατικούς κόμβους αισθητήρων, αλλά και από κινητούς οι οποίοι θα συνυπάρχουν μέσα στο δίκτυο.

- **Εντοπισμός γεγονότος:** Υπάρχει περίπτωση η αρχική ανάπτυξη της τοπολογίας του ασύρματου δικτύου αισθητήρων να μην είναι βέλτιστη για να ανιχνεύσει έγκαιρα τα γεγονότα. Το πρόβλημα αυτό μπορεί να διορθωθεί χρησιμοποιώντας κινητούς αισθητήρες οι οποίοι θα επισκέπτονται ανά περιόδους ή δυναμικά τις περιοχές με υψηλή πιθανότητα εμφάνισης γεγονότος.
- **Απομόνωση από ελαττωματικούς ή κακόβουλους κόμβους:** Η απόδοση του ελαττωματικού κόμβου μπορεί να συγκριθεί με την απόδοση κάποιου γνωστού λειτουργικού κόμβου και στη συνέχεια να μεταφερθεί ο λειτουργικός κόμβος στη θέση του ελαττωματικού. Με παρόμοιο τρόπο μπορούν να εφαρμοστούν τεχνικές για την απομόνωση των κακόβουλων κόμβων. Αυτό θα έχει ως αποτέλεσμα την αύξηση της αξιοπιστίας του δικτύου και την απομόνωση κόμβων που ίσως να επηρεάσουν αρνητικά το δίκτυο.

2.4 Ζητήματα κινητικότητας αισθητήρων που βρίσκονται υπό έρευνα

Μερικά από τα ζητήματα που μελετούνται και αφορούν την κινητικότητα στα Ασύρματα Δίκτυα Αισθητήρων σύμφωνα με το[17] είναι:

- **Adaptive Localization**: Οι περισσότερες από τις προτεινόμενες λύσεις είναι συγκεντρωμένες σε μια υπολογιστική μονάδα και υπολογιστικά πολύ ακριβές. Όμως προσθέτοντας κινητούς κόμβους σε ένα σε στατικό δίκτυο αισθητήρων, μπορούν να χρησιμοποιηθούν δεδομένα από την κίνηση των κόμβων για τον υπολογισμό της αλλαγής της τοποθεσίας τους (odometry). Μέσω της έρευνας που γίνεται, στόχος είναι να βρεθούν αλγόριθμοι με λιγότερη υπολογιστική πολυπλοκότητα οι οποίοι να είναι και κατανεμημένοι.
- **Κατανεμημένη χαρτογράφηση:** Οι τεχνικές για την χαρτογράφηση του περιβάλλοντος γύρω από ένα κινητό κόμβο αισθητήρα είναι υπολογιστικά πολύπλοκες. Μέσω της έρευνας μελετούνται τρόποι

για την αξιοποίηση πληροφοριών από στατικούς κόμβους αισθητήρων.

- **Κάλυψη:** Μέσω της έρευνας που γίνεται στόχος είναι να βρεθούν αποτελεσματικοί τρόποι συνεργασίας κινητών κόμβων με στατικών, για τη μέγιστη κάλυψη της περιοχής του δικτύου. Όμως δεν έχει μελετηθεί ακόμα αρκετά η περίπτωση κατά την οποία οι κινητοί αισθητήρες προσπαθούν να καλύψουν δυναμικά τις ακάλυπτες περιοχές του δικτύου.
- **Μαζικός επαναπρογραμματισμός:** Όταν πρέπει να αναπτυχθούν χιλιάδες κόμβοι αισθητήρων σε ένα δίκτυο, δεν είναι εφικτό να προγραμματιστεί ο κάθε κόμβος ξεχωριστά. Είναι όμως δυνατό ένας κινητός κόμβος να κινηθεί έτσι ώστε να καλύψει όλο το δίκτυο και να επαναπρογραμματίσει όλους τους κόμβους που βρίσκονται στην εμβέλειά του.
- **Κατανεμημένη ρύθμιση των μετρήσεων:** Οι κόμβοι αισθητήρων είναι αρκετά φθηνοί είναι πολύ επιρρεπείς σε σφάλματα και γι' αυτό χρειάζεται να γίνονται συχνές ρυθμίσεις των μετρήσεων (calibration). Υπό μελέτη βρίσκεται η περίπτωση κάποιοι κινητοί κόμβοι αισθητήρων να μετακινούνται σε όλο το δίκτυο και να ρυθμίζουν όλους τους γείτονές τους.
- **Επιδιόρθωση του Δικτύου:** Μελέτη γίνεται για την ανάπτυξη αποδοτικών αλγορίθμων κατά τους οποίους οι κινητοί κόμβοι αισθητήρων θα μετακινούνται και θα αντικαθιστούν νεκρούς κόμβους με όσο το δυνατό λιγότερη κατανάλωση ενέργειας.
- **Κατανεμημένη αποθήκευση πληροφοριών, συνάθροιση πληροφοριών και τεχνικές querying:** Οι περισσότερες μελέτες μέχρι τώρα που έχουν γίνει για την data-centric αποθήκευση, την συνάθροιση και για τεχνικές querying, επικεντρώνονται στα στατικά δίκτυα αισθητήρων. Πιστεύεται ότι δεν μπορούν να εφαρμοστούν αποδοτικά τέτοιου είδους τεχνικές στα κινητά δίκτυα ασύρματων αισθητήρων. Όμως αυτό δεν σημαίνει ότι δεν μπορούν να

εφαρμοστούν αποδοτικά σε υβριδικά δίκτυα όπου συνυπάρχουν στατικοί και κινητοί κόμβοι αισθητήρων

Σε αυτή τη διπλωματική εργασία διερευνήθηκαν πρωτόκολλα συλλογής δεδομένων με πολλαπλά κινητά Sink και πρωτόκολλα για τον καθορισμό θέσης με τη χρήση κινητών Sink. Πιο εκτεταμένα μελετήθηκε το θέμα της κάλυψης στα ασύρματα δίκτυα αισθητήρων με την χρήση κινητών κόμβων. Συγκεκριμένα, το κεφάλαιο 3 ασχολείται με τα πρωτόκολλα κινητικότητας sink που αφορούν την συλλογή δεδομένων και τον καθορισμό θέσης, το κεφάλαιο 4 ασχολείται με τα πρωτόκολλα που αφορούν τον έλεγχο κάλυψης στα ασύρματα δίκτυα αισθητήρων. Στο κεφάλαιο 5 γίνεται ανάπτυξη ενός αλγορίθμου για την ανίχνευση και ελαχιστοποίηση των τρυπών κάλυψης με την προσθήκη επιπρόσθετων κινητών κόμβων αισθητήρων και στο κεφάλαιο 6 ακολουθεί η υλοποίηση του αλγορίθμου σε πιλοτικό δίκτυο.

Κεφαλαιο 3

Πρωτόκολλα κινητικότητας Sink

3.1 Γενικά

Στις συσκευές αισθητήρων η ενέργεια παρέχεται μέσο μπαταριών. Έτσι η ενέργεια είναι πολύτιμη αφού η περιοδική αλλαγή των μπαταριών σε μεγάλου μεγέθους αναπτύξεις δικτύων δεν είναι εφικτή.

Τα δεδομένα που μαζεύονται συνήθως στέλλονται σε ένα στατικό σημείο ελέγχου, το οποίο ονομάζεται sink, και διαδίδονται από κόμβο σε κόμβο. Αυτή η διαδικασία καταναλώνει μεγάλα ποσά ενέργειας ειδικά στην περιοχή που βρίσκεται το sink, όπου οι κόμβοι πρέπει να μεταδίδουν τα δεδομένα τα οποία έρχονται από κόμβους που βρίσκονται πιο μακριά. Έτσι δημιουργούνται σημεία στο δίκτυο στα οποία υπάρχει μεγάλη κατανάλωση ενέργειας. Ο τερματισμός της λειτουργίας αυτών των κόμβων λόγω εξάντλησης της ενέργειάς τους οδηγεί σε αποσυνδεδεμένα δίκτυα και σε δίκτυα που δεν λειτουργούν σωστά.

Έτσι μια νέα προσέγγιση είναι να κινείται το sink μέσα στην περιοχή του δικτύου. Με αυτό τον τρόπο το sink θα βρίσκεται σχεδόν πάντα κοντά σε ένα υποσύνολο από αισθητήρες και θα μπορεί να λαμβάνει τα δεδομένα

από αυτούς με πολύ λίγη κατανάλωση ενέργειας. Καθώς θα μετακινείται το sink μέσα σε όλη την περιοχή του δικτύου, θα μπορεί να μαζέψει όλα τα διαθέσιμα δεδομένα.

Παρόλα αυτά, η διάσχιση της περιοχή του δικτύου με αποτελεσματικό τρόπο είναι πολύ κρίσιμη, γιατί η αποτυχία της επίσκεψης μερικών περιοχών του δικτύου μπορεί να οδηγήσει σε απώλεια δεδομένων και η μη συχνή επίσκεψη σε μερικές περιοχές μπορεί να οδηγήσει σε μεγάλες καθυστερήσεις παράδοσης δεδομένων. Ακόμη, στην περίπτωση που έχουμε κινητά sinks, τα προβλήματα δρομολόγησης και καθορισμού θέσης είναι πιο δύσκολο να λυθούν.

Πλεονεκτήματα της κινητικότητας των Sink:

Ανεξάρτητα από τις εμφανείς δυσκολίες, αυτός ο τρόπος συλλογής δεδομένων έχει πολλές ελκυστικές ιδιότητες συμφωνα με το [29].
Ένα κινητό sink το οποίο κινείται κοντά στους κόμβους αισθητήρων μπορεί να βοηθήσει στην μείωση της κατανάλωσης ενέργειας αφού τα δεδομένα μεταδίδονται μέσω λιγότερων hops, με αποτέλεσμα την μείωση του αριθμού των μεταδιδόμενων πακέτων. Η επιπλέον ενέργεια που καταναλώνεται με την λειτουργία και την κινητικότητα ενός sink δεν επηρεάζει το ολικό χρόνο ζωής του δικτύου αισθητήρων εφόσον το sink θεωρείται εξωτερικός παράγοντας για το δίκτυο (π.χ μπορεί να είναι όχημα του οποίου η κίνηση του να ελέγχεται από άνθρωπο ή μπορεί να είναι ένα robot το οποίο να είναι προγραμματισμένο να επιστρέφει αυτόματα σε ένα σταθερό σημείο για να επαναφορτιστεί).
Ένα άλλο πλεονέκτημα είναι το ότι με την χρήση κινητού sink μπορούν να χειριστούν καλύτερα αραιά κατανεμημένα δίκτυα και αποσυνδεδεμένα δίκτυα. Ακόμη, η χρήση κινητού sink επιτρέπει την παρακολούθηση μιας περιοχής με την χρήση λιγότερων αισθητήρων μειώνοντας το συνολικό κόστος λειτουργίας του δικτύου.

Επιπλέον οι κόμβοι αισθητήρων μπορούν να μειώσουν την εμβέλεια μετάδοσης τους σε χαμηλότερη τιμή για την επικοινωνία με ένα sink.
Τα κινητά sink μπορούν να προσπεράσουν προβληματικές περιοχές στις οποίες οι αισθητήρες δεν μπορούν να λειτουργήσουν, όπως μικρές λίμνες, μεγάλες πέτρες που εμποδίζουν την μετάδοση και άλλα εμπόδια. Σε τέτοιου είδους καταστάσεις τα multi-hop πρωτόκολλα είτε αποτυγχάνουν, είτε καταναλώνουν πολλή ενέργεια για να ξεπεράσουν τα εμπόδια.

Το throughput και η ορθότητα των δεδομένων μπορεί να επιτευχθεί με την χρήση ενός mobile sink: Ελαττώνοντας τον αριθμό των hops, μειώνεται η πιθανότητα για την μετάδοση λάθους και οι συγκρούσεις των πακέτων ελαττώνεται.

Σε εφαρμογές στις οποίες η ασφάλεια είναι σημαντική, με την χρήση κινητού sink, η παρουσία του δικτύου δεν ανιχνεύεται εύκολα λόγω της χαμηλής δύναμης μετάδοσης που χρησιμοποιείται.

3.2 Πρωτόκολλα συλλογής δεδομένων με πολλαπλά κινητά Sink

Τρία εξελικτικά πρωτόκολλα συλλογής δεδομένων για Ασύρματα δίκτυα αισθητήρων με πολλαπλά κινητά sinks που έχουν προταθεί στο [29] είναι τα πιο κάτω:

- Στρατηγική της ολοκληρωτικής κινητικότητας (The fully mobile strategy).
- Μικτή στρατηγική από στατικά και κινητά Sinks (Mixed strategy of static and mobile sinks).
- Χρήση του Equidistribution πρωτοκόλλου.

Η διεξαγωγή των πειραμάτων των πιο πάνω πρωτοκόλλων – στρατηγικών έγινε με τις ακόλουθες προϋποθέσεις:

- Οι αισθητήρες δεν κινούνται.
- Η περιοχή του δικτύου είναι ένας τετραγωνικός χώρος D x D.

- Η θέση των αισθητήρων είναι τυχαία και ακολουθούν μια ομοιόμορφη κατανομή.
- Οι αισθητήρες μπορούν να μετρήσουν αρκετές καταστάσεις του περιβάλλοντος.
- Κάθε αισθητήρας έχει σταθερή εμβέλεια μετάδοσης R και παίρνει ενέργεια από μπαταρία
- Κάθε αισθητήρας έχει μνήμη γενικής χρήσης χωρητικότητας C.
- Κάθε αισθητήρας μπορεί να βρίσκεται σε μια από τρεις καταστάσεις (κατάσταση μετάδοσης μηνύματος, κατάσταση παράληψης μηνύματος, κατάσταση αίσθησης γεγονότος).
- Υπάρχουν πολλαπλά sinks στο δίκτυο τα οποία είναι κινητά.
- Τα sinks έχουν μεγάλη υπολογιστική ισχύ, μνήμη και μεγάλα αποθέματα ενέργειας.
- Τα sinks μπορούν να υπολογίσουν με ακρίβεια την θέση τους (π.χ με τη χρήση GPS).
- Τα sinks γνωρίζουν τα όρια του δικτύου.

Στρατηγική της ολοκληρωτικής κινητικότητας: Υπάρχουν K sinks τα οποία κινούνται το καθένα ανεξάρτητα μέσα σε όλη την περιοχή του δικτύου και μαζεύουν δεδομένα από τους αισθητήρες. Κάθε sink κινείται τυχαία προς οποιαδήποτε κατεύθυνση. Αυτή είναι η πιο απλή δυνατή κίνηση. Τα sinks δεν γνωρίζουν τις τοποθεσίες των αισθητήρων. Αυτή η μέθοδος είναι πολύ εύρωστη και αυτό οφείλεται στο ότι η τυχαία κίνηση εγγυείται την επίσκεψη όλων των αισθητήρων του δικτύου και την συλλογή των δεδομένων ακόμα και από αποσυνδεδεμένες περιοχές. Τα δεδομένα συλλέγονται παθητικά, δηλαδή κάθε sink μεταδίδει ένα beacon message περιοδικά (σε τυχαία επιλεγμένες χρονικά περιόδους για την αποφυγή προβλήματος broadcast storm). Κάθε κόμβος αισθητήρα ο οποίος λαμβάνει το beacon στέλλει πίσω στο sink τα δεδομένα που μάζεψε. Αυτή η προσέγγιση ελαχιστοποιεί την κατανάλωση ενέργειας αφού γίνεται μόνο μια μετάδοση ανά ανιχνευμένο γεγονός. Όμως χρειάζεται αρκετός χρόνος

για την επίσκεψη όλων των κόμβων, ειδικά σε περιπτώσεις που τα sinks δεν είναι πολλά.

Μικτή στρατηγική από στατικά και κινητά Sinks: Υβριδική λύση όπου ένα μέρος των sinks είναι κινητοί. Οι κινητοί sink κινούνται τυχαία όπως αναφέρθηκε πιο πάνω, ενώ οι στατικοί παραμένουν στην αρχική τους θέση. Η συλλογή των δεδομένων γίνεται με την εφαρμογή ενός πρωτοκόλλου το οποίο δημιουργεί περιορισμένου μεγέθους propagation trees με ένα sink να αποτελεί τη ρίζα του. Κάθε sink κάνει broadcast ένα beacon message κατά περιόδους, το οποίο μεταφέρει την ταυτότητα του sink, ένα hop counter, και ένα time to live counter (TTL). Κάθε κόμβος αισθητήρα γνωρίζει πόσα hops απέχει από πιο κοντινό sink και ένα timestamp το οποίο δηλώνει την τελευταία φορά που ανανεώθηκε η απόσταση σε hops από το πιο κοντινό sink. Όταν ένας κόμβος αισθητήρα παραλάβει ένα beacon αποφασίζει, αν πρέπει να ανανεώσει τον πατρικό του κόμβο ή όχι. Οι αισθητήρες οι οποίοι ανήκουν σε ένα propagation tree, μπορούν να μεταδώσουν τα δεδομένα τους στο αντίστοιχο sink. Το βάθος των δέντρων καθορίζεται από την τιμή του TTL, η οποία είναι μια λειτουργική παράμετρος του sink. Με αυτό τον τρόπο ο διαχειριστής του δικτύου μπορεί να κάνει trade off μεταξύ της μείωσης της καθυστέρησης και της κατανάλωσης ενέργειας. Όταν τα sink κινούνται, υπάρχει περίπτωση τα propagation trees να αποσυνδεθούν. Όταν ο κόμβος αισθητήρα που απέχει μηδέν hops από το sink δεν μπορεί πλέον να επικοινωνήσει με το sink απλά αποθηκεύει τα δεδομένα που παράγει ή παραλαμβάνει από κόμβους παιδιά του και περιμένει να ακούσει κάποιο beacon message από κάποιο sink το οποίο να απέχει μηδέν hops. Αν μετά από χρονικό περιθώριο δεν βρεθεί κάποιο sink για να του μεταδώσει τα δεδομένα που έχει αποθηκευμένα συμμετέχει σε κάποιο άλλο δέντρο.

Το πρωτόκολλο Equidistribution: Και σε αυτή την προσέγγιση χρησιμοποιούνται κινητά και στατικά sink και χρησιμοποιούν το πρωτόκολλο που αναφέρθηκε πιο πάνω. Η βασική διαφορά είναι ότι τα

sink περιλαμβάνουν την θέση τους και ένα sequence number μέσα στο beacon message. Όταν ένα sink Si ακούσει ένα beacon message ενός άλλου sink Sj, τότε προσπαθεί να αλλάξει την τροχιά του για να απομακρυνθεί από το άλλο sink. Επιπλέον το sink Si αποθηκεύει το sequence number του Sj και την επόμενη φορά που λαμβάνει ένα beacon message από το Sj μεταβάλλει την τροχιά του μόνο αν το νέο beacon message που παρέλαβε έχει μεγαλύτερο sequence number από αυτόν που έχει αποθηκεύσει. Εφόσον το sink δεν παραλαμβάνει beacon message από άλλο sink, κινείται τυχαία. Παρόλο που το πρωτόκολλο αυτό κάνει τα sinks τα οποία βρίσκονται κοντά το ένα σε σχέση με το άλλο να απομακρύνονται, χρησιμοποιεί τυχαία κίνηση των sink τον περισσότερο χρόνο, έτσι διατηρεί τις ιδιότητες που έχουν αναφερθεί πιο πάνω όπως για παράδειγμα την δικαιοσύνη. Παρόλα αυτά, ο υπολογισμός της θέσης των άλλων sink είναι βασισμένος σε τοπικές πληροφορίες, έτσι δεν εγγυείται πλήρως ότι τα sink θα κατανέμονται δίκαια μέσα στο δίκτυο όλη την ώρα.

Η απόδοση που έχει επιτευχθεί σε καθένα από τα πιο πάνω πρωτόκολλα, είναι ανάλογη με τον αριθμό των κινητών sink. Χρησιμοποιώντας περισσότερα κινητά sink η απόδοση μεγαλώνει ανάλογα. Παρόλο που η απόδοση μειώνεται όσο μειώνεται ο αριθμός των κινητών sink, ένα σχετικά μικρό ποσοστό κινητών κόμβων είναι ικανό να κρατήσει την αποδοτικότητα σε ικανοποιητικά επίπεδα.

Για την συλλογή των δεδομένων από τους αισθητήρες έχουν μελετηθεί άλλα τέσσερα πρωτόκολλα τυχαίας και προβλεπόμενης κινητικότητας στο [13], των οποίων η κινητικότητα των sink συνδυάζεται με τρεις στρατηγικές συλλογής δεδομένων οι οποίες είναι η παθητική, η multihop και η limited multihop. Τα πρωτόκολλα αυτά είναι τα ακόλουθα:

A. Random Walk and passive data collection
B. Partial Random Walk with Limited Multi-Hop Data propagation
Γ. Biased Random Walk with Passive Data Collection

Δ. Deterministic Walk with Multi-hop Data Propagation

Η διεξαγωγή των πειραμάτων των πιο πάνω πρωτοκόλλων έγινε με τις ακόλουθες προϋποθέσεις :

- Η περιοχή του δικτύου είναι τετραγωνική μεγέθους DxD.
- Οι τοποθεσίες των κόμβων αισθητήρων μέσα στο δίκτυο είναι τυχαίες ακολουθώντας μια ομοιόμορφη κατανομή.
- Οι περιοχές του δικτύου που έχουν ψηλή πυκνότητα σε κόμβους αισθητήρων ονομάζονται 'pockets' οι οποίες για λόγους απλοποίησης των σεναρίων παρουσιάζονται ως κυκλικές περιοχές ακτίνας r_p οι οποίες δεν επικαλύπτονται μεταξύ τους. Η πυκνότητα των κόμβων αισθητήρων σε αυτές τις περιοχές είναι d_p.
- Οι κόμβοι αισθητήρων είναι εξοπλισμένοι με διάφορους αισθητήρες για την μέτρηση διαφόρων περιβαλλοντολογικών καταστάσεων.
- Κάθε αισθητήρας έχει σταθερή εμβέλεια μετάδοσης R και παίρνει ενέργεια από μπαταρία
- Κάθε αισθητήρας έχει μνήμη γενικής χρήσης χωρητικότητας C.
- Κάθε αισθητήρας μπορεί να βρίσκεται σε μια από τρεις καταστάσεις (κατάσταση μετάδοσης μηνύματος, κατάσταση παράληψης μηνύματος, κατάσταση αίσθησης γεγονότος).
- Τα sinks έχουν μεγάλη υπολογιστική ισχύ, μνήμη και μεγάλα αποθέματα ενέργειας.
- Τα sinks μπορούν να υπολογίσουν με ακρίβεια την θέση τους (πχ με την χρήση GPS).
- Τα sinks μπορούν να αλλάξουν την ταχύτητα και την κατεύθυνση τους στιγμιαία.
- Οι κόμβοι αισθητήρων εκτελούν τρεις κατηγορίες εφαρμογών οι οποίες είναι :

- Periodic Sensing: Οι κόμβοι αισθητήρων παρακολουθούν το περιβάλλον όλη την ώρα και συνεχώς ενημερώνουν το sink για τις μετρήσεις που κατέγραψαν.
- Event driven: Οι κόμβοι αισθητήρων παρακολουθούν το περιβάλλον όλη την ώρα αλλά ενημερώνουν τα sinks μόλις καταγράψουν κάποιο γεγονός.
- Query based: Οι κόμβοι αισθητήρων παρακολουθούν το περιβάλλον όλη την ώρα αλλά ενημερώνουν τα sinks όταν αυτά τους το ζητήσουν.

Random Walk and passive data collection: Το πιο απλό σχέδιο κινητικότητας είναι αυτό του τυχαίου περιδιαβάσματος, κατά την οποία το κινητό sink μπορεί να κινηθεί χαοτικά προς οποιαδήποτε κατεύθυνση με μεταβαλλόμενη ταχύτητα. Με αυτού του είδους την κίνηση δεν χρειάζεται ειδική γνώση του δικτύου. Αυτή η τεχνική είναι καλή γιατί εγγυείται την επίσκεψη όλων των κόμβων του δικτύου μαζεύοντας πληροφορίες ακόμα και από αποσυνδεδεμένες περιοχές . Η συλλογή των δεδομένων γίνεται με παθητικό τρόπο. Περιοδικά γίνεται broadcast ένα μήνυμα από το sink και κάθε αισθητήρας ο οποίος παραλαμβάνει το μήνυμα στέλλει τα δεδομένα του σε αυτόν. Αυτή η διαδικασία όμως πιθανόν να δημιουργεί πολλές συγκρούσεις και γι αυτό το λόγο πρέπει να χρησιμοποιηθεί ένα MAC πρωτόκολλο με τη δυνατότητα να κάνει back off όπου πρέπει.

Αυτό το πρωτόκολλο ελαχιστοποιεί την κατανάλωση ενέργειας επειδή γίνεται μόνο μία μετάδοση ανά γεγονός που αισθάνθηκε. Όμως, όσο αφορά το χρόνο, δεν είναι ιδιαίτερα αποδοτικός, λόγω του μεγάλου χρόνου για την επίσκεψη όλων των κόμβων αισθητήρων.

Partial Random Walk with Limited Multi-Hop Data propagation: Είναι σχέδιο κινητικότητας τυχαίου περιδιαβάσματος κατά το οποίο χρησιμοποιούνται προκαθορισμένες περιοχές και τυχαία μετάβαση μεταξύ των περιοχών, ανάλογα με την συνδεσιμότητά τους. Η ταχύτητα της κίνησης είναι σταθερή και προκαθορισμένη. Το πρωτόκολλο συλλογής των

δεδομένων μπορεί να χρειαστεί να επικοινωνήσει με κόμβους αισθητήρων εκτός της εμβέλειάς του. Έτσι, το πρωτόκολλο δημιουργεί propagation trees με το sink να αντιπροσωπεύει τη ρίζα. Οι κόμβοι αισθητήρων που ανήκουν στο propagation tree μπορούν να στέλνουν αμέσως τα δεδομένα τους στο sink. Το βάθος αυτών των δέντρων καθορίζεται από την τιμή του ttl, η οποία είναι μια λειτουργική παράμετρος του κινητού sink. Με αυτό τον τρόπο ο διαχειριστής του δικτύου μπορεί να κάνει trade-off μεταξύ της μείωσης της καθυστέρησης και αύξησης της κατανάλωσης ενέργειας. Όταν ο κόμβος αισθητήρα, ο οποίος είναι το άμεσο παιδί του sink, δεν μπορεί πλέον να επικοινωνήσει με το sink, τότε αποθηκεύει όλα τα δεδομένα που πρέπει να στείλει στο sink (αυτά που παράγει και αυτά που παραλαμβάνει από τους άλλους κόμβους του δέντρου) περιμένοντας για ένα συγκεκριμένο χρονικό διάστημα ώστε να ξαναγίνει το πρώτο παιδί του sink για να του στείλει όλα τα αποθηκευμένα δεδομένα. Αν το χρονικό διάστημα περάσει τότε ο κόμβος αυτός μπορεί να συμμετάσχει σε ένα άλλο δέντρο. Αυτό το πρωτόκολλο υποθέτει γνώση του δικτύου και η απόσταση που διανύει το sink είναι μικρότερη σε σχέση με την απόσταση που διανύεται από το sink με τη χρήση του αλγόριθμου Random Walk and passive data collection. Όμως το κόστος επικοινωνίας και των υπολογισμών είναι μεγάλο.

Biased Random Walk with Passive Data Collection: Σε αυτό το σχέδιο κινητικότητας χρησιμοποιείται ένας λογικός γράφος ο οποίος έχει επεκταθεί έτσι ώστε να ευνοεί συγκεκριμένες περιοχές του δικτύου για τη βελτίωση της συλλογής δεδομένων και την αντιμετώπιση προβλημάτων τα οποία μπορεί να δημιουργούνται λόγω της τοπολογίας. Η συλλογή των δεδομένων γίνεται με παθητικό τρόπο όπως αυτόν του πρωτοκόλλου Random Walk and passive data collection. Το πρωτόκολλο αυτό χρησιμοποιεί γνώση η οποία μαζεύεται από το sink για να αυξήσει την κάλυψη σε νέες περιοχές ή για να αυξήσει την παράδοση των δεδομένων σε περιοχές στις οποίες υπάρχουν πολλοί κόμβοι αισθητήρων. Σε σύγκριση με το πρωτόκολλο Random Walk and passive data collection,

πετυχαίνει γρηγορότερη κάλυψη του δικτύου, ψηλότερα ποσοστά παράδοσης δεδομένων και λιγότερη καθυστέρηση με μιαν αύξηση των υπολογισμών στο sink.

Deterministic Walk with Multi-hop Data Propagation: Σε αυτό το σχέδιο κινητικότητας χρησιμοποιείται μια απλή μορφή ελεγχόμενης κινητικότητας κατά την οποία το sink μετακινείται βάση μιας προκαθορισμένης τροχιάς (κυκλικής ή γραμμικής). Επειδή το sink καλύπτει μια μικρή περιοχή του δικτύου, είναι αναγκαίο να μαζεύει τα δεδομένα με τη χρήση ενός multi-hop propagation πρωτοκόλλου παρόμοιου με αυτό του πρωτοκόλλου Partial Random Walk with Limited Multi-Hop Data propagation, χωρίς όμως μηχανισμούς timeout και time-to-live. Γι αυτό, τα μονοπάτια δημιουργούνται σύμφωνα με την ελάχιστη απόσταση των hop και αναπτύσσονται σε όλη την περιοχή του δικτύου. Όταν ο κόμβος αισθητήρα, ο οποίος είναι το άμεσο παιδί του sink, δεν μπορεί πλέον να επικοινωνήσει με το sink, τότε αποθηκεύει όλα δεδομένα που πρέπει να στείλει στο sink (αυτά που παράγει και αυτά που παραλαμβάνει από τους άλλους κόμβους του δέντρου) περιμένοντας ώστε να ξαναγίνει το πρώτο παιδί του sink για να του στείλει όλα τα αποθηκευμένα δεδομένα. Μια άλλη προσέγγιση είναι να αποσυνδεθεί το δέντρο αυτό και με τα νέα δεδομένα να δημιουργηθεί ένα νέο. Όμως αυτή η προσέγγιση έχει μέγιστη απόσταση hop ίση με αυτή της προσέγγισης με την χρήση στατικού sink. Η υλοποίηση αυτού του πρωτοκόλλου προϋποθέτει ένα ψηλό κόστος για τους κόμβους αισθητήρων οι οποίοι εκτελούν υπολογισμούς για τη δημιουργία του δέντρου και για το multi-hop propagation. Παρόλα αυτά, έχει λιγότερες καθυστερήσεις από όλους τους προαναφερθέντες αλγόριθμους. Επιπρόσθετα, η επιλογή της τροχιάς μεγέθους L δημιουργεί ένα trade-off μεταξύ του κόστους στο sink και του κόστους στους κόμβους αισθητήρων (πχ αυξάνοντας το μήκος L, το sink κινείται περισσότερο και να πλησιάζει περισσότερο τους αισθητήρες).

3.3 Καθορισμός Θέσης με τη χρήση κινητών Sink

Το localization είναι μια μεγάλη περιοχή έρευνας στα πλαίσια των ασύρματων δικτύων αισθητήρων. Σε ένα αυτορυθμιζόμενο ad-hoc ασύρματο δίκτυο αισθητήρων είναι πολύ σημαντικό οι κόμβοι αισθητήρων να γνωρίζουν την τοποθεσία τους μέσα σε μια συγκεκριμένη γεωγραφική περιοχή. Γνωρίζοντας την τοποθεσία τους, οι κόμβοι αισθητήρων, μπορούν να παράγουν πληροφορίες με περισσότερο νόημα, αφού μπορούν να πληροφορήσουν τον τελικό χρήστη για την ακριβή τοποθεσία εντός του δικτύου, εκεί ακριβώς καταγράφηκε το συγκεκριμένο γεγονός που ανιχνεύτηκε. Ακόμη μπορούν να βοηθήσουν στη συντήρηση του δικτύου σε περίπτωση που χαλάσει ένας ή ένα σύνολο από κόμβους αισθητήρων σε μια συγκεκριμένη περιοχή. Ο σωστός καθορισμός της θέσης των αισθητήρων βοηθά και στη δημιουργία αποτελεσματικών μονοπατιών δρομολόγησης προς το sink, το οποίο μπορεί να είναι στατικό ή κινητό, περιορίζοντας την κατανάλωση ενέργειας και μειώνοντας τον απαιτούμενο χρόνο.

Οι δυνατότητες ενός μεμονωμένου κόμβου αισθητήρα είναι πάρα πολύ περιορισμένες και γι αυτό το λόγο απαιτείται συνεργασία μεταξύ των κόμβων αισθητήρων και των sink με την ελάχιστη κατανάλωση ενέργειας.

Ο καθορισμός θέσης μπορεί να γίνει είτε Range-Based, είτε Range-Free. Ο καθορισμός θέσης βασισμένος σε Range-Based μεθόδους θεωρείται ότι υπολογίζεται με ψηλή ακρίβεια και βασίζεται στις αποστάσεις και τις γωνιές που σχηματίζουν μεταξύ τους οι κόμβοι αισθητήρων. Το localization βασισμένο σε Range-free μεθόδους θεωρείται ότι υπολογίζεται με λιγότερη ακρίβεια αλλά παρέχει μια οικονομική εναλλακτική λύση.

Η χρήση κινητών sink για τον εντοπισμό της τοποθεσίας των ασύρματων κόμβων αισθητήρων έχει μελετηθεί αρκετά. Όμως τα περισσότερα μοντέλα βασίζονται στην προϋπόθεση ότι τα κινητά sink είναι ενσωματωμένα με

επιπλέον υλικό (π.χ GPS) έτσι ώστε να μπορούν να υπολογίσουν της θέσης τους ανά πάσα στιγμή. Αυτά τα μοντέλα υποφέρουν από ανακρίβεια όπως για παράδειγμα στην περίπτωση που χρησιμοποιείται GPS μέσα σε κλειστά κτήρια και σπίτια.

Range-Based μέθοδοι: Μερικές από τις προσεγγίσεις που χρησιμοποιούνται για τον καθορισμό της θέσης είναι: Received Signal Strength Indication (RSSI), Angle of Arrival (AoA), Time of Arrival (ToA) and Time Difference of Arrival (TDoA). Αρκετές από αυτές τις μεθόδους χρησιμοποιούν μια τεχνική (multi-lateration) κατά την οποία οι αποστάσεις υποθέτονται (πχ βασισμένες στο RSSI) και με τη βοήθεια των γνωστών αποστάσεων μπορεί να υπολογιστεί η τοποθεσία των κόμβων αισθητήρων. Εφόσον οι μετρήσεις με τη χρήση RSSI είναι επιρρεπείς σε λάθη, ένα μικρό λάθος το οποίο μπορεί να δημιουργηθεί από ένα απλό υπολογισμό μπορεί να οδηγήσει σε μεγαλύτερου βαθμού λάθος με τηνχρήση των αποτελεσμάτων για τον υπολογισμό της τοπολογίας μεγάλου αριθμού κόμβων αισθητήρων.

Σε περίπτωση που το δίκτυο αισθητήρων αναπτύσσεται εντός κτιρίου ή οποιουδήποτε άλλου κλειστού χώρου, προτείνεται η χρήση συστημάτων RADAR τα οποία χρησιμοποιούν radio frequency signal strength (RF) για τον εντοπισμό και την παρακολούθηση. Άλλο μοντέλο που μπορεί να χρησιμοποιηθεί είναι το Floor Attenuation Factor propagation model εξαιτίας της απλότητας και της ακρίβειας του. Όμως το μοντέλο αυτό υποφέρει από ανακρίβειες που οφείλονται στην περιορισμένη χρήση του beacon base station. Εκτός από αυτό, είναι και ακριβό για να χρησιμοποιηθεί, επειδή χρειάζεται τουλάχιστον 3 beacons για να μπορεί να υλοποιηθεί. Ένα άλλο μοντέλο για εσωτερική ανάπτυξη είναι το Cricket location estimation system το οποίο χρησιμοποιεί συνδυασμό τεχνολογίας RF και ultrasound με τη βοήθεια των beacons. Ο λόγος για τον οποίο χρησιμοποιείται και ultrasound είναι γιατί μέσα στα κτίρια αλλοιώνεται η ακρίβεια των μετρήσεων με RSSI λόγω των τοίχων και άλλων εμποδίων. Το

Cricket χρησιμοποιεί χρονοπρογραμματισμό τυχαίας μετάδοσης και ένα receiver decoding algorithm για τον υπολογισμό της εκτίμησης του maximum likelihood των κόμβων αισθητήρων.

Η μέθοδος που ακολουθεί και έχει προταθεί στο [1] δεν είναι μόνο οικονομική, αλλά είναι και αρκετά απλή για να υλοποιηθεί και ταυτόχρονα αρκετά αποδοτική. Για τον καθορισμό της θέσης με αυτή τη μέθοδο χρησιμοποιείται ένας κινητός sink ο οποίος δεν έχει την δυνατότητα να γνωρίζει τη δική του τοπολογία.

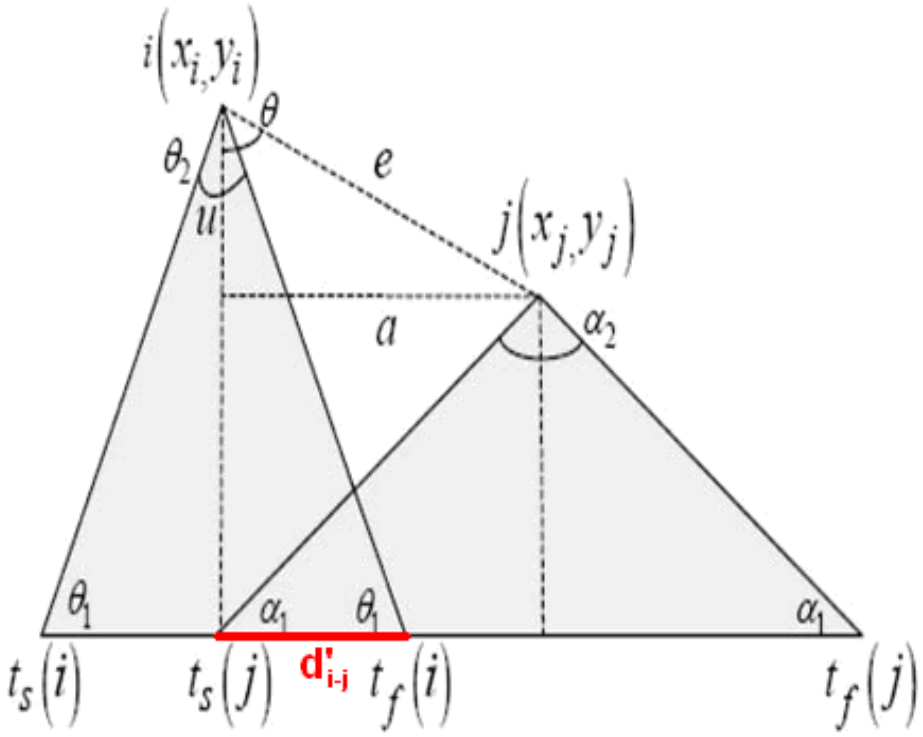
Η υλοποίηση της μεθόδου έγινε με τις ακόλουθες προϋποθέσεις:

- Το κινητό sink κινείται ευθεία, σταματά τυχαία και στη συνέχεια συνεχίζει τυχαία προς μια νέα κατεύθυνση.
- Η ταχύτητα του κινητού sink παραμένει σταθερή κατά την κίνηση του μέσα στο δίκτυο αισθητήρων.
- Η ανάπτυξη των αισθητήρων του δικτύου έγινε με τυχαίο τρόπο.

Το κινητό sink κινείται μέσα στο δίκτυο και υλοποιεί μια γεωμετρική αναπαράσταση της τοπολογίας των ασύρματων αισθητήρων βασιζόμενο στο επικοινωνιακό εύρος του κάθε κόμβου αισθητήρα (δηλαδή το χρονικό περιθώριο στον οποίο ένας κόμβος αισθητήρα μπορεί να επικοινωνήσει με επιτυχία με το κινητό sink, μέχρι το κινητό sink βγει έξω από το πεδίο επικοινωνίας του κόμβου αισθητήρα). Η απόσταση μεταξύ των κόμβων αισθητήρων υπολογίζεται από το κινητό sink με συσχετισμό με την γραμμή της κίνησης του sink.

Μεθοδολογία: Όταν το κινητό sink εισέρχεται στο δίκτυο κάνει broadcast beacon messages ζητώντας acknowledgement. Κάθε αισθητήρας που λαμβάνει το μήνυμα στέλλει πίσω στο sink acknowledgment καθώς και μια λίστα που περιέχει τους γειτονικούς του κόμβους αισθητήρων και ένα προαιρετικό τρέχον εύρος επικοινωνίας. Η λίστα με τους γειτονικούς

κόμβους αισθητήρων χρειάζεται έτσι ώστε το sink να μπορεί να τους τοποθετήσει σε μια από τις πλευρές της γραμμής κίνησης του (μετά από σύγκριση με τις λίστες άλλων κόμβων αισθητήρων). Όταν το κινητό sink παραλάβει πακέτο από κάποιο κόμβο αισθητήρα για πρώτη φορά καταγράφει το πρώτο και τελευταίο time-stamp. Τα time-stamps στη συνέχεια τοποθετούνται σε ένα γράφο ανάλογα με την γραμμή κινητικότητας του sink.



Σχήμα 3.1: Καθορισμός θέσης μεταξύ γειτονικών κόμβων [1]

Το πιο πάνω σχέδιο απεικονίζει δυο κόμβους αισθητήρων οι οποίοι είναι τοποθετημένοι σε σχέση με τη γραμμή της κίνησης του sink. Τα (Xi,Yi) και (Xj,Yj) είναι οι άγνωστες συντεταγμένες των δύο κόμβων αισθητήρων i και j.

Η απόσταση μεταξύ των δύο κόμβων μπορεί να υπολογιστεί με την διαφορά των time stamps (d'), όταν το sink μπαίνει και βγαίνει από την εμβέλεια τους.

Έτσι έχουμε:

$d'_{i-j} = (t_f(i) - t_s(j)) \times v$

όπου το v είναι η ταχύτητα του κινητού sink και είναι μια γνωστή σταθερά. Το επόμενο βήμα είναι να υπολογιστεί η προέκταση της απόστασης μεταξύ των τοποθεσιών των δύο κόμβων αισθητήρων, η οποία υπολογίζεται βάση της πιο κάτω εξίσωσης:

$\alpha = R(\cos\theta_1 + \cos\alpha_1) - d'_{i-j}$ (R = ακτίνα εμβέλειας μετάδοσης)

Το u υπολογίζεται με την εξίσωση:

$u = R(\sin\theta_1 - \sin\alpha_1)$

Έτσι, μπορεί να υπολογιστεί το συντομότερο μονοπάτι χρησιμοποιώντας τις πιο πάνω εξισώσεις:

$$e = \sqrt{u^2 + \alpha^2}$$

δηλαδή,

$$e = \sqrt{4R^2 \cos^2\left(\frac{\theta_1+\alpha_1}{2}\right) + d'^2_{i-j} - 4d'_{i-j} R \cos\left(\frac{\theta_1+\alpha_1}{2}\right)\cos\left(\frac{\theta_1-\alpha_1}{2}\right)}$$

Η πολική γωνία μεταξύ των δυο κόμβων αισθητήρων σε σχέση με την γραμμή της κίνησης του sink είναι:

$$\theta = \sin^{-1}\frac{\alpha}{u}$$

Το χρονικό διάστημα μεταξύ δύο διαδοχικές αποστολές πακέτων από το sink πρέπει να είναι αρκετά μικρό για να επιτύχουμε καλύτερη ακρίβεια.

Κεφαλαιο 4

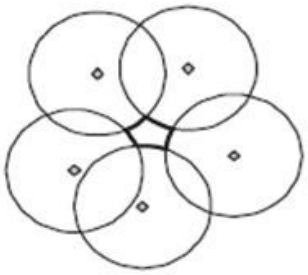
Έλεγχος Κάλυψης στα Ασύρματα Δίκτυα Αισθητήρων και πρωτόκολλα κινητικότητας

4.1 Γενικά

Πολλές ανωμαλίες μπορούν να συμβούν σε ένα δίκτυο αισθητήρων, οι οποίες μπορούν να προκαλέσουν προβλήματα στην λειτουργικότητα του. Η περιοχή που καλύπτεται από το δίκτυο αισθητήρων και υποτίθεται ότι καλύπτεται 100% από τους κόμβους αισθητήρων, μπορεί να έχει τρύπες κάλυψης. Οι τρύπες αυτές είναι περιοχές του δικτύου οι οποίες δεν καλύπτονται από κανένα κόμβο και δημιουργούνται από την τυχαία εναέρια ανάπτυξη του δικτύου, από την ύπαρξη εμποδίου στην περιοχή και πιο συχνά με το θάνατο ενός κόμβου. Παρόμοια, οι κόμβοι ίσως να μην μπορούν να επικοινωνήσουν σωστά αν υπάρχουν τρύπες δρομολόγησης στην ανεπτυγμένη τοπολογία του δικτύου. Έτσι το δίκτυο αποτυγχάνει να πετύχει το στόχο του αν μερικά από τα γεγονότα που

πρέπει να καταγραφούν δεν αισθάνονται ή δεν μπορούν να μεταφερθούν προς το sink.

4.2 Τρύπες κάλυψης

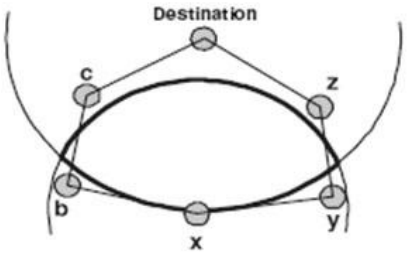


Σχήμα 4.1: Τρύπα κάλυψης μεταξύ γειτονικών κόμβων [2]

Δεδομένου ότι έχουμε ένα σύνολο από αισθητήρες και μια συγκεκριμένη περιοχή. Σύμφωνα με το [2], δεν υπάρχει τρύπα κάλυψης, αν κάθε σημείο της περιοχής καλύπτεται από τουλάχιστον k κόμβους αισθητήρων και όπου k είναι ο απαιτούμενος βαθμός κάλυψης για την συγκεκριμένη εφαρμογή. Η κάλυψη αίσθησης ενός κόμβου αισθητήρα συνήθως θεωρείται ομοιόμορφη προς όλες τις κατευθύνσεις και παρουσιάζεται ως ένας δίσκος με κέντρο του τον κόμβο αισθητήρα. Αυτό το ιδανικό μοντέλο βασίζεται στη μη ρεαλιστική υπόθεση ότι όλοι οι κόμβοι αισθητήρων έχουν δίσκο κάλυψης ιδίου μεγέθους. Κανονικά όμως η κάλυψη εξαρτάται από τις ικανότητες αίσθησης του κάθε κόμβου αισθητήρα και από τα χαρακτηριστικά του γεγονότος που πρέπει να ανιχνευτεί.

Οι τρύπες κάλυψης δεν πρέπει να συγχύζονται με τις τρύπες δρομολόγησης. Μια τρύπα δρομολόγησης είναι μια περιοχή ενός δικτύου αισθητήρων, στην οποία δεν υπάρχουν διαθέσιμοι κόμβοι ή οι διαθέσιμοι κόμβοι που υπάρχουν δεν μπορούν να συμμετάσχουν στη δρομολόγηση των δεδομένων για διάφορους λόγους. Οι τρύπες αυτές μπορούν να δημιουργηθούν είτε από τα κενά που θα δημιουργηθούν στην ανάπτυξη του δικτύου είτε από θάνατο κάποιου κόμβου, είτε από τη δημιουργία του local minimum phenomenon το οποίο παρουσιάζεται, όταν τα δεδομένα

δρομολογούνται με greedy forwarding (η δρομολόγηση των δεδομένων γίνεται βάση της τοποθεσίας του προορισμού των δεδομένων).



Σχήμα 4.2: Local minimum phenomenon [2]

Local minimum phenomenon: Ένας κόμβος αισθητήρα x προσπαθεί να στείλει τα δεδομένα σε έναν από τους 1- hop γείτονες του ο οποίος θα είναι γεωγραφικά πιο κοντά στον προορισμό των δεδομένων σε σχέση με τη δική του γεωγραφική θέση. Αυτή η διαδικασία δρομολόγησης σταματά, όταν ο κόμβος αισθητήρα x δεν μπορεί να βρει κάποιο γειτονικό κόμβο ενός hop που να βρίσκεται πιο κοντά στον προορισμό των δεδομένων σε σχέση με τη δική του θέση. Ο μόνος τρόπος για να φτάσουν τα δεδομένα στον προορισμό τους είναι με τη δρομολόγηση των δεδομένων στον κόμβο αισθητήρα b ή y, που είναι πιο μακριά από τον προορισμό των δεδομένων σε σχέση με τον κόμβο x.

4.3 Αντιμετώπιση των τρυπών κάλυψης στα δίκτυα ασύρματων κόμβων αισθητήρων

Αρκετοί αλγόριθμοι έχουν αναπτυχθεί για την αντιμετώπιση των τρυπών κάλυψης στα ασύρματα δίκτυα αισθητήρων. Ακολούθως παρουσιάζονται συνοπτικά οι πιο γνωστοί.

Το **Coverage Configuration Protocol** (CCP) που αναλύεται στο [50], παρέχει ελαστικότητα στη ρύθμιση του δικτύου των κόμβων αισθητήρων, ώστε να ρυθμιστεί αυτόματα σε διαφορετικού βαθμού καλύψεις. Κάθε αναπτυγμένος κόμβος τρέχει τον K_s - coverage αλγόριθμο, όταν $R_c >= 2R_s$,

όπου R_c είναι το εύρος επικοινωνίας και όπου R_s είναι το εύρος αίσθησης. Δεδομένου ότι ζητείται κάλυψη βαθμού K_s, ένας κόμβος αισθητήρα προγραμματίζεται να κοιμάται αν κάθε τοποθεσία μέσα στο εύρος κάλυψης του (R_s) είναι ήδη K_s, δηλαδή καλύπτεται από άλλους ενεργούς γειτονικούς κόμβους αισθητήρων. Σε περιπτώσεις που ισχύει ότι $R_c < 2R_s$, το πρωτόκολλο CCP δεν εγγυάται τη συνδεσιμότητα μαζί με την κάλυψη. Έτσι προτάθηκε ο συνδυασμός με το πρωτόκολλο SPAN για να του παρέχει τη συνδεσιμότητα του δικτύου. Το πρωτόκολλο **SPAN** [14] είναι μια κατανεμημένη τεχνική συντονισμού για multi-hop ad-hoc ασύρματα δίκτυα αισθητήρων που μειώνει την κατανάλωση ενέργειας χωρίς να μειώνει σημαντικά τη χωρητικότητα ή τη συνδεσιμότητα του δικτύου.

Έχουν προταθεί αλγόριθμοι πολυωνυμικού χρόνου στο [24] για τον έλεγχο αν κάθε κομμάτι της περιοχής του δικτύου είναι τουλάχιστον καλυμμένο με τον ελάχιστο αριθμό απαιτούμενων κόμβων αισθητήρα. Ο αλγόριθμος **k-UC** υποθέτει ένα ομοιόμορφο δίσκο αίσθησης, ενώ ο αλγόριθμος **k-NC** υποθέτει ένα μη ομοιόμορφο εύρος αίσθησης για κάθε κόμβο. Ο προτεινόμενος αλγόριθμος χρειάζεται μόνο της πληροφορίες τοποθεσίας για κάθε αναπτυγμένο κόμβο για να υπολογίσει την ύπαρξη της επιθυμητής κάλυψης. Στο πρωτόκολλο K-UC , κάθε κόμβος αισθητήρα υπολογίζει την κάλυψη του γειτονικού του κόμβου εάν αυτός ο γείτονας του βρίσκεται σε περιοχή μικρότερη από δύο φορές το εύρος αίσθησης του κόμβου. Στο πρωτόκολλο K-NC λαμβάνονται υπόψη διαφορετικοί κανόνες για το αν ένας κόμβος αισθητήρα πρέπει να υπολογίσει την κάλυψη γειτονικών του κόμβων και για το πότε ένας κόμβος βρίσκεται μέσα στο εύρος αίσθησης ενός άλλου κόμβου. Στη συνέχεια οι κόμβοι υπολογίζουν την κάλυψη της περιμέτρου τους από το εύρος αίσθησης γειτονικών κόμβων. Με αυτό τον τρόπο ο κάθε κόμβος ελέγχει αν η περίμετρος του είναι καλυμμένη από τον απαιτούμενο βαθμό κάλυψης. Για την ανίχνευση τρυπών κάλυψης προτείνεται η χρήση μιας κεντρικής οντότητας ελέγχου η οποία θα μαζεύει τις πληροφορίες για τις τρύπες κάλυψης που υπάρχουν

στο δίκτυο και θα στέλνει νέους κόμβους εκεί που χρειάζεται επιδιόρθωση. Αυτή η προσέγγιση όμως δεν είναι scalable.

Ένας κατανεμημένος αλγόριθμος έχει προταθεί στο [52]για τον έλεγχο της πυκνότητας των κόμβων ενός ασύρματου δικτύου αισθητήρων, ικανός να παρέχει διαφορετικές καλύψεις ανάλογα με τις διάφορες περιοχές ανάπτυξης του δικτύου, ονομάζεται **Differentiated**. Ο αλγόριθμος αυτός είναι βασισμένος στο συγχρονισμό του χρόνου μεταξύ των γειτονικών κόμβων. Στη φάση της αρχικοποίησης, οι κόμβοι παίρνουν την τοποθεσία τους και συγχρονίζονται με τους γειτονικούς τους κόμβους. Στη φάση αίσθησης, η οποία αποτελείται από πολλούς γύρους ίσων χρονικών διαστημάτων, κάθε κόμβος χωρίζει όλη την τοπική του περιοχή σε grids και διαφημίζει το reference point του μαζί με το χρόνο αρχής και τέλους που σχετίζονται με αυτό το reference point. Ο κάθε κόμβος αισθητήρα ταξινομεί τους γείτονες του που καλύπτουν ένα συγκεκριμένο grid σε αύξουσα σειρά με βάση τον αριθμό των reference points τους σε ένα γύρο. Βασισμένος στη χρονική ακολουθία που πάρθηκε, ένας κόμβος μπορεί να υπολογίσει πόσο χρόνο χρειάζεται να είναι ενεργός έτσι ώστε όλο το grid να έχει τον απαιτούμενο βαθμό κάλυψης.

Για απαιτήσεις κάλυψης ενός μόνο κόμβου έχει προταθεί στο [53]το **Optimal Geographical Density Control Protocol,** το οποίο προσπαθεί να μειώσει την επικάλυψη του εύρους αίσθησης του κάθε κόμβου όπου $R_c >= 2R_s$. Ο αλγόριθμος ξεκινά με όλους τους κόμβους να βρίσκονται στην κατάσταση ‘’UNDECIDED”. Ένας κόμβος με ικανοποιητικά αποθέματα ενέργειας επιλέγεται τυχαία για να αρχίσει τη διαδικασία επιλογής κόμβων. Έτσι, ο κόμβος αυτός κάνει broadcast ένα μήνυμα power-on. Οι κόμβοι που παραλαμβάνουν το μήνυμα αυτό ελέγχουν τα αποθέματα ενέργειας τους και την περιοχή κάλυψης εντός της περιοχής τους. Αν υπάρχουν ικανοποιητικά αποθέματα ενέργειας και η περιοχή τους δεν είναι πλήρως καλυμμένη, τότε ο κόμβος αυτός προσθέτει τον κόμβο που του είχε στείλει το power-on μήνυμα ως γείτονα του, αλλάζει την

κατάσταση του σε "ON" και αναμεταδίδει παντού το power-on μήνυμα ξανά. Η διαδικασία συνεχίζεται με ελάχιστα διαφορετική συμπεριφορά για τα μηνύματα power-on τα οποία λαμβάνονται από κόμβους starting και non-starting. Οι προσομοιώσεις του πρωτοκόλλου αυτού έδειξαν ότι δεν μπορεί πάντα να διατηρεί την αρχική κάλυψη του δικτύου όπως ακριβώς ήταν.

Ένα πρωτόκολλο που δημιουργήθηκε για να αντιμετωπίσει προβλήματα κάλυψης δικτύων με ανομοιόμορφου μεγέθους κάλυψης των αισθητήρων ονομάζεται **Sponsored Area protocol** [40] και δουλεύει ως εξής: Ένας κόμβος αποφασίζει να γίνει ανενεργός, αν ανακαλύψει ότι γειτονικοί του κόμβοι καλύπτουν πλήρως το εύρος αίσθησής του. Για να αποφευχθεί η πιθανότητα να γίνουν ανενεργοί περισσότεροι από ένας κόμβος στην ίδια περιοχή με αποτέλεσμα τη δημιουργία τρύπας κάλυψης, οι κόμβοι χρησιμοποιούν ένα τυχαίο αλγόριθμο back-off πριν να γίνουν ανενεργοί.

Μερικές παραλήψεις του πιο πάνω αλγόριθμου έχουν μελετηθεί και είναι οι πιο κάτω:

1. Οι γειτονικοί κόμβοι οι οποίοι βρίσκονται έξω από το εύρος αίσθησης δεν συμπεριλαμβάνονται παρόλο που μπορούν να συνεισφέρουν στην κάλυψη του κόμβου.
2. Ο υπολογισμός κάλυψης της περιοχής ο οποίος βασίζεται σε sectors, έχει ως αποτέλεσμα σε συντηρητικό υπολογισμό της περιοχής που μπορούν να καλύψουν οι γειτονικοί κόμβοι αισθητήρων.

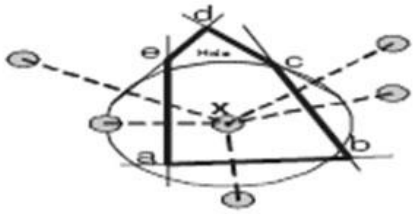
Για την αντιμετώπιση τους προτάθηκε στο [25] η έννοια των effective neighbor nodes για τον ακριβή υπολογισμό της κάλυψης των κόμβων και για την απόφαση εάν θα απενεργοποιηθούν περιττοί κόμβοι διατηρώντας την αρχική κάλυψη της περιοχής. Το σύνολο των γειτόνων περιλαμβάνει όλους τους κόμβους οι οποίοι βρίσκονται σε διπλάσια απόσταση από το

πεδίο αίσθησης του κάθε κόμβου. Το πρωτόκολλο αυτό ονομάστηκε **Extended – Sponsored Area**. Τα πειράματα που έγιναν έδειξαν ότι το πρωτόκολλο αυτό έχει 30% καλύτερα αποτελέσματα από το πρωτόκολλο Sponsored Area για τη μείωση του συνολικού αριθμού των κόμβων αισθητήρων που χρειάζονται ώστε να διατηρείται η αρχική κάλυψη του δικτύου.

4.4 Αντιμετώπιση των τρυπών κάλυψης σε δίκτυα κινητών ασύρματων κόμβων αισθητήρων

Πολλοί ερευνητές έχουν μελετήσει τεχνικές για να πετύχουν μέγιστη κάλυψη της περιοχής ενός ασύρματου δικτύου αισθητήρων χρησιμοποιώντας κινητούς κόμβους. Ένα τυπικό πρόβλημα για τέτοιου είδους σενάρια είναι η μεγιστοποίηση της κάλυψης του δικτύου με περιορισμούς τον χρόνο ανάπτυξης του δικτύου, την απόσταση που πρέπει να διανύσουν οι κόμβοι αισθητήρων για να καλύψουν τα κενά και την πολυπλοκότητα του πρωτοκόλλου.

Μια προτεινόμενη λύση είναι η χρήση διαγραμμάτων Voronoi για να ανακαλυφθεί η ύπαρξη τρυπών κάλυψης μετά την ανάπτυξη του δικτύου. Ο κάθε κόμβος χρειάζεται να γνωρίζει που βρίσκονται οι γειτονικοί του κόμβοι για να μπορεί να δημιουργήσει το δικό του διάγραμμα Voronoi. Αυτό σημαίνει, ότι είτε όλοι οι κόμβοι έχουν GPS, είτε χρησιμοποιούν κάποια τεχνική καθορισμού τοποθεσίας. Όλη η περιοχή του δικτύου χωρίζεται σε πολύγονα Voronoi και κάθε πολύγονο έχει μόνο έναν κόμβο αισθητήρα, ο οποίος έχει την ιδιότητα να βρίσκεται πιο κοντά σε κάθε σημείο του πολύγονου από οποιοδήποτε άλλο γειτονικό κόμβο αισθητήρα. Στη συνέχεια ο κάθε κόμβος αισθητήρα συγκρίνει το πεδίο αίσθησης του με την περιοχή που καταλαμβάνει το πολύγονο και υπολογίζει τις τοπικές τρύπες κάλυψης.



Σχήμα 4.2: Χρήση διαγραμμάτων Voronoi για την ανακάλυψη τρυπών κάλυψης [48]

Για το κλείσιμο τρυπών κάλυψης έχουν προταθεί τρεις κατανεμημένοι αλγόριθμοι στο [48].Ο **Vector Based** (VEC), ο **Voronoi Based** (VOR) και ο αλγόριθμος **Minmax**.

Κατά τον αλγόριθμο Vector Based, ένας κόμβος υπολογίζει την μέση απόσταση μεταξύ των γειτόνων του, υποθέτοντας ότι όλοι οι κόμβοι είναι ομοιόμορφα κατανεμημένοι στο δίκτυο και προσπαθεί να κρατήσει την απόσταση του από κάθε γειτονικό κόμβο περίπου ίση με αυτή την υπολογισμένη απόσταση.

Κατά τον αλγόριθμο Voronoi Based, όταν ανιχνευτεί μια τρύπα κάλυψης, ο κόμβος αισθητήρα κινείται προς την πιο μακρινή κορυφή του Voronoi πολυγώνου του, για να καλύψει την τοπική μέγιστη τρύπα κάλυψης.

Ο αλγόριθμος Minmax δουλεύει με παρόμοιο τρόπο, όπως το Voronoi Based. Επιπρόσθετα όμως με την κίνηση του κάθε κόμβου προς τη μακρινότερη κορυφή του Voronoi πολυγώνου του, λαμβάνει υπόψη του και την απόσταση του από τις άλλες κορυφές και έτσι βρίσκει την κατάλληλη θέση μέσα στο πολύγωνο, ούτως ώστε η απόσταση από τη μακρινότερη κορυφή να ελαχιστοποιηθεί.

Μέσα από προσομοιώσεις των πιο πάνω αλγορίθμων αποδείχτηκε ότι ο αλγόριθμος Minmax επιτυχαίνει καλύτερη μέγιστη κάλυψη από τους άλλους δυο προαναφερθέντες αλγόριθμους παρόλο που είναι ελάχιστα

υπολογιστικά πιο ακριβός και μετακινεί τους κόμβους σε μεγαλύτερες αποστάσεις. Γι αυτό το λόγο, προτάθηκε ως εναλλακτική λύση η προσομοίωση της κινητικότητας για τον απευθείας υπολογισμό της τελικής τοποθέτησης των κόμβων αισθητήρων με στόχο να μειωθεί η κίνηση των κόμβων αισθητήρων εις βάρος επιπλέον υπολογιστικής πολυπλοκότητας.

Ένα άλλο πρωτόκολλο που χρησιμοποιεί την κινητικότητα των κόμβων αισθητήρων έχει προταθεί στο [19] και ονομάζεται **Co-Fi**. Προτάθηκε για την κάλυψη των τρυπών κάλυψης της περιοχής του δικτύου, οι οποίες οφείλονται όμως στην εξάντληση ενέργειας κόμβων αισθητήρων. Οι κόμβοι οι οποίοι έχουν ελάχιστα αποθέματα ενέργειας κάνουν broadcast ένα "μήνυμα πανικού" και κόμβοι αισθητήρων με ικανοποιητικά αποθέματα ενέργειας τους απαντούν, αν μπορούν να μετακινηθούν στην περιοχή χωρίς να χάσουν την υπάρχουσα κάλυψη τους. Η απάντηση στο "μήνυμα πανικού" περιέχει τα αποθέματα ενέργειας του κόμβου που απάντησε και το κόστος μετακίνησης του (δηλαδή τη μικρότερη απαιτούμενη απόσταση που θα διανύσει για να φτάσει στον τελικό προορισμό). Ο κόμβος αισθητήρα που έστειλε το "μήνυμα πανικού" παραλαμβάνει τις απαντήσεις και επιλέγει τον πιο κατάλληλο κόμβο αισθητήρα και τον ειδοποιεί για να κινηθεί. Το πρωτόκολλο αυτό βασίζεται στο broadcasting μηνυμάτων πανικού και έτσι δεν θα δουλέψει σε περίπτωση που ένας κόμβος καταστραφεί από άλλους παράγοντες με αποτέλεσμα τη δημιουργία τρυπών κάλυψης που δεν μπορούν να διορθωθούν.

Τα **Potential Fields** χρησιμοποιούνται συχνά σε εφαρμογές κινητής ρομποτικής για πλοήγηση των robot και για αποφυγή των εμποδίων. Το πρωτόκολλο Potential Fields στα ασύρματα δίκτυα αισθητήρων, όπως αναφέρει το [22], βασίζεται μόνο σε μια προϋπόθεση. Προϋποθέτει ότι ο κάθε κόμβος αισθητήρα είναι εξοπλισμένος με έναν αισθητήρα, ο οποίος του επιτρέπει να καθορίζει την απόσταση και την κατεύθυνση προς όλους τους κόμβους αισθητήρων και τα εμπόδια που βρίσκονται γύρω από αυτόν. Χρησιμοποιώντας αυτές τις πληροφορίες, ο κάθε κόμβος αισθητήρα

μπορεί να καθορίσει τις εικονικές δυνάμεις οι οποίες ασκούνται πάνω του (οι κόμβοι αισθητήρων συμπεριφέρονται ως εικονικά μόρια και οι εικονικές δυνάμεις τους απωθούν τον ένα κόμβο αισθητήρα από τον άλλο και από άλλα εμπόδια) και να τις μετατρέψει σε ένα control vector, το οποίο θα χρησιμοποιήσει για να κινηθεί ανάλογα. Καμιά άλλη πληροφορία δεν χρειάζεται, επομένως το πρωτόκολλο αυτό έχει το πλεονέκτημα ότι δεν χρειάζεται να εφαρμοστεί centralized έλεγχος, πρωτόκολλο καθορισμού επικοινωνίας ή να ανταλλάξουν πληροφορίες οι κόμβοι αισθητήρων μεταξύ τους. Αυτό έχει ως αποτέλεσμα το πρωτόκολλο αυτό να είναι εύρωστο και επεκτάσιμο.

Δύο άλλοι αλγόριθμοι έχουν προταθεί στο [21] για την επίλυση του προβλήματος κάλυψης από ένα μόνο κόμβο αισθητήρα. Ο πρώτος αλγόριθμος ονομάζεται **Distributed Self-Spreading algorithm** (DSS), κατά τον οποίο οι κόμβοι αισθητήρων είναι αρχικά ανεπτυγμένοι τυχαία και στη συνέχεια αρχίζουν να κινούνται βάση εικονικών δυνάμεων που ασκούν σε αυτούς οι γειτονικοί κόμβοι (πχ όπως στον VEC). Οι δυνάμεις που ασκούνται σε κάθε κόμβο αισθητήρα από τους γείτονες του εξαρτώνται από την τοπική πυκνότητα των κόμβων αισθητήρων και από την απόστασή των κόμβων από τους γείτονές τους. Ο δεύτερος αλγόριθμος ονομάζεται **Intelligent Deployment and Clustering Algorithm (IDCA)** και εκμεταλλεύεται την ελάχιστη κατανάλωση ενέργειας που επιτυγχάνει το clustering. Η τοπική πυκνότητα συγκρίνεται με την πυκνότητα που αναμένεται να υπάρχει σε περίπτωση που το δίκτυο αναπτυσσόταν με ομοιόμορφη πυκνότητα. Βάση του αποτελέσματος ένας αισθητήρας επιλέγει την κατάλληλη μέθοδο clustering. Σε αυτή τη μέθοδο, η σχετική υπόλοιπη ενέργεια των αισθητήρων υπαγορεύει, εάν ένας κόμβος πρέπει να κινηθεί ή όχι. Η ιδέα είναι να μειωθεί η διαφορά στην εναπομένουσα ενέργεια, όταν όλοι οι κόμβοι διανεμηθούν ομοιόμορφα στην περιοχή του δικτύου.

Για το κλείσιμο των τρυπών κάλυψης με κινητούς εφεδρικούς κόμβους που δουλεύουν σωστά, προτάθηκε στο [26] μια διαδικασία η οποία ονομάστηκε **snake-like cascading replacement process**. Πρώτα η περιοχή του δικτύου χωρίζεται σε πολλά μικρά τετράγωνα (r x r). Σε κάθε τετράγωνο επιλέγεται ως Head ένας κόμβος ο οποίος δουλεύει σωστά και είναι υπεύθυνος για την επικοινωνία με τα γειτονικά τετράγωνα. Τον ρόλο του Head ενός τετραγώνου τον αναλαμβάνουν οι κόμβοι του τετραγώνου που δουλεύουν σωστά εκ περιτροπής. Κάθε κόμβος Head γνωρίζει την τοποθεσία του τετραγώνου σε σχέση με το δίκτυο και τον αριθμό των κόμβων που δουλεύουν σωστά μέσα στο τετράγωνο καθώς και την τοποθεσία τους. Η πλήρης συνδεσιμότητα και κάλυψη της περιοχής του δικτύου είναι εγγυημένη, όταν σε κάθε τετράγωνο υπάρχει ένας κόμβος Head. Όταν ένα τετράγωνο ανιχνευτεί πως δεν έχει έναν κόμβο Head, αρχίζει η διαδικασία για τη μεταφορά ενός κόμβου που δουλεύει σωστά από τη γειτονική περιοχή για να γίνει Head του τετράγωνου αυτού. Για την εύρεση άδειων τετραγώνων παρουσιάζονται δύο μοντέλα, το παθητικό και το ενεργητικό.

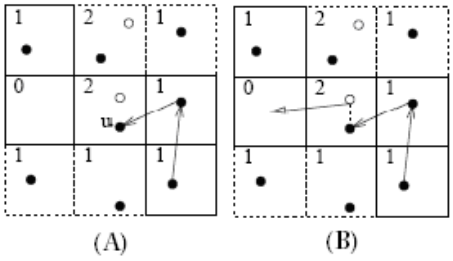
Κατά το παθητικό μοντέλο, το άδειο τετράγωνο ανιχνεύεται μόνο όταν μια ροή επικοινωνίας χρειάζεται να περάσει μέσα από αυτό. Κατά το ενεργητικό μοντέλο, κάθε κόμβος, ο οποίος είναι το head του τετραγώνου, παρακολουθεί την περιοχή με τα γειτονικά του τετράγωνα και ανιχνεύει το τετράγωνο που δεν έχει κόμβο head. Όταν βρεθεί άδειο τετράγωνο, αρχίζει η διαδικασία αντικατάστασης από τα γειτονικά τετράγωνα. Η μεθοδολογία snake-like cascading replacement process προϋποθέτει ότι όλοι οι κόμβοι έχουν την ίδια εμβέλεια επικοινωνίας R και οι κόμβοι οι οποίοι είναι μέσα σε αυτή την εμβέλεια θεωρούνται γείτονες και επικοινωνούν απευθείας. Ακόμη, οι πληροφορίες τοποθεσίας των κόμβων μπορούν να βρεθούν είτε έχοντας Global Positioning System receivers σε κάποιους κόμβους ή ένα κινητό κόμβο beacon ή απλά να υπολογιστούν σε σχετικό σύστημα συντεταγμένων. Χρησιμοποιώντας πληροφορίες γειτνίασης μόνο ενός hop και με το κατανεμημένο σχέδιο ελέγχου που υλοποιείται, κάνουν

την μεθοδολογία αυτή επεκτάσιμη και να συγκλίνει γρήγορα με ένα ελαχιστοποιημένο κόστος.

Αλγόριθμος παθητικού μοντέλου:

Βήμα 1: Στον κόμβο, στον οποίο σταμάτησε η ροή της επικοινωνίας (u), θα ξεκινήσει η διαδικασία αντικατάστασης, αφού ο κόμβος – head u δεν μπορεί να βρει τον επόμενο κόμβο για να μεταβιβάσει τα δεδομένα στο επόμενο τετράγωνο. (Αυτό σημαίνει ότι ανιχνεύτηκε κενό τετράγωνο στην κατεύθυνση προώθησης των δεδομένων).

Βήμα 2: Εύρεση εφεδρικού και έμπιστου κόμβου μέσα στο τετράγωνο του u (κόμβος v) για να μετακινηθεί μέσα στο κενό τετράγωνο πριν την έναρξη του επόμενου γύρου.

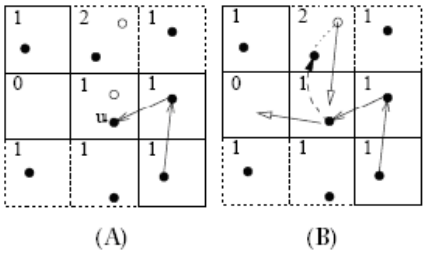


Σχήμα 4.3.1: Snake-like Cascading Replacement Process - Παθητική μέθοδος. (A) ανίχνευση άδειου τετραγώνου, (B) Μετακίνηση του u στο άδειο τετράγωνο[26].

Βήμα 3: Αν το πιο πάνω βήμα αποτύχει, επαναλαμβάνονται τα ακόλουθα μέχρι ο ειδοποιημένος κόμβος u να μπορεί να βρει ένα εφεδρικό και έμπιστο κόμβο v στο πιο πάνω βήμα.

- Επιλογή ενός γειτονικού τετραγώνου εκτός από το άδειο. Τα τετράγωνα με εφεδρικούς και έμπιστους κόμβους προτιμούνται.

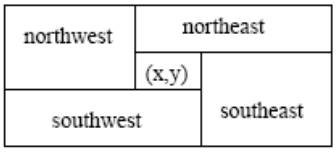
- Αποστολή ειδοποίησης στο head του τετραγώνου που επιλέκτηκε για αντικατάσταση του u από εφεδρικό και έμπιστο κόμβο του τετραγώνου του.
- Μετακίνηση του u στο άδειο τετράγωνο πριν από την έναρξη του επόμενου γύρου, αφήνοντας το τετράγωνο του άδειο για αντικατάσταση από κόμβο του γειτονικού τετραγώνου που επιλέκτηκε. u θεωρείται τώρα ο κόμβος – head του γειτονικού κόμβου που ειδοποιήθηκε και ο αλγόριθμος συνεχίζει από το βήμα 2.



Σχήμα 4.3.2: Snake-like Cascading Replacement Process - Παθητική μέθοδος. (A) Ανίχνευση άδειου τετραγώνου, (B) Μετακίνηση του u στο άδειο τετράγωνο και μετακίνηση εφεδρικού κόμβου από γειτονικό τετράγωνο για να καλύψει την τρύπα που θα δημιουργήσει η μετακίνηση του κόμβου u [26].

Αλγόριθμος ενεργητικού μοντέλου:

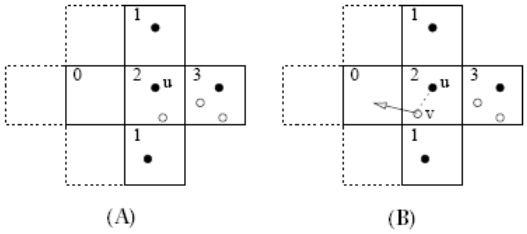
Βήμα 1: Για τον κόμβο - head κάθε τετραγώνου ο οποίος ανιχνεύει ένα άδειο γειτονικό τετράγωνο, αν δεν έχει πάρει ειδοποίηση από τον προηγούμενο γύρο από αυτή την περιοχή, ξεκινά τη διαδικασία αντικατάστασης μετά από τον καθορισμό της περιοχής του τετραγώνου Δ.



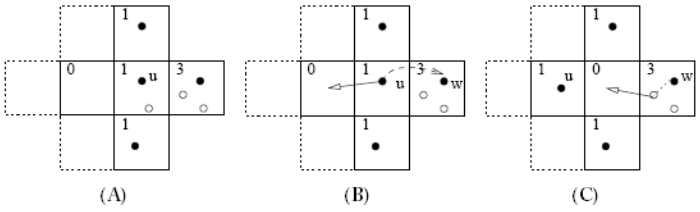
Σχήμα 4.4.1: Snake-like Cascading Replacement Process – Ενεργητική μέθοδος. Διαχωρισμός περιοχής του grid σύμφωνα με ένα άδειο τετράγωνο [26].

Βήμα 2: Για κάθε κόμβο – head u πού έχει αρχίσει τη διαδικασία αντικατάστασης ή έχει ειδοποιηθεί για cascading διαδικασία αντικατάστασης, γίνεται εύρεση ενός εφεδρικού και έμπιστου κόμβου v από το τετράγωνό τους, για να μετακινηθεί στο άδειο τετράγωνο πριν την έναρξη του επόμενου γύρου.

Βήμα 3: Αν δεν βρεθεί ένας κόμβος v από το αντίστοιχο τετράγωνο, επιλέγεται ένα γειτονικό τετράγωνο εκτός από το άδειο τετράγωνο το οποίο να βρίσκεται στην περιοχή Δ. Ακολουθεί αποστολή ειδοποίησης για την αντικατάσταση του u, αποστέλλοντας και πληροφορίες για το Δ. Τα τετράγωνα με εφεδρικούς και έμπιστους κόμβους προτιμούνται. Μετά από την ειδοποίηση ακολουθεί η μετακίνηση του u στην άδεια περιοχή πριν από την έναρξη του επόμενου γύρου.



Σχήμα 4.4.2: Snake-like Cascading Replacement Process – Ενεργητική μέθοδος. (A) Ανίχνευση άδειου τετραγώνου. (B) Μετακίνηση εφεδρικού κόμβου από το τετράγωνο του u [26].



Σχήμα 4.4.3: Snake-like Cascading Replacement Process – Ενεργητική μέθοδος. (A) Ανίχνευση άδειου τετραγώνου. (B) Αποστολή ειδοποίησης για αντικατάσταση και μετακίνηση του u στο κενό τετράγωνο, (C) Cascading μετακίνηση [26].

4.5 Αντιμετώπιση των τρυπών κάλυψης σε υβριδικά δίκτυα κινητών και στατικών ασύρματων κόμβων αισθητήρων

Με την χρήση **unmanned Aerial Vehicle** (UAV) έχει μελετηθεί στο [16] το ζήτημα της ανάπτυξης ενός δικτύου με ικανοποιητική συνδεσιμότητα. Το robot με την ικανότητα να πετά και ονομάζεται **AVATAR**, αναπτύσσει το δίκτυο με βάση προϋπολογισμένη τοπολογία του δικτύου. Μόλις αναπτυχθούν οι αισθητήρες, υπολογίζουν το χάρτη συνδεσιμότητας τους και τον στέλνουν στο robot. Η υπάρχουσα συνδεσιμότητα του δικτύου συγκρίνεται με την απαιτούμενη τοπολογία και ανιχνεύονται οι ασύνδετες περιοχές. Έτσι υπολογίζονται τα σημεία ανάπτυξης επιπλέον κόμβων για την επιδιόρθωση των ασύνδετων περιοχών και οι κόμβοι αυτοί μεταφέρονται εκεί από το robot. Η ανάπτυξη επιπρόσθετων κόμβων αυξάνει την πυκνότητα των κόμβων αισθητήρων, πετυχαίνει πολλαπλή κάλυψη και συνδεσιμότητα ή επιδιορθώνει τρύπες που δημιουργούνται από χαλασμένους κόμβους αισθητήρων. Όμως τα πειραματικά αποτελέσματα αυτής της μεθοδολογίας έδειξαν ότι είναι πάρα πολύ δύσκολο να επιτευχθεί η επιθυμητή τοπολογία ενός δικτύου με εναέρια ανάπτυξη.

Σχήμα 4.5: Unmanned Aerial Vehicle [16].

Το πρωτόκολλο **single robot** [7] λύνει ταυτόχρονα προβλήματα εξερεύνησης και κάλυψης μιας δεδομένης περιοχής. Το πρόβλημα της κάλυψης λύνεται με τη βοήθεια ενός robot το οποίο κινείται συνέχεια στην δεδομένη περιοχή. Αρχικά, το κινητό robot αναπτύσσει το δίκτυο καθώς εξερευνά το άγνωστο περιβάλλον. Στη συνέχεια το ανεπτυγμένο δίκτυο με τις τοπικές του μετρήσεις οδηγεί το robot σε περιοχές με προβλήματα κάλυψης. Με αυτό τον τρόπο το κινητό robot, χρησιμοποιώντας τα τοπικά δεδομένα αίσθησης και την προτεινόμενη κατεύθυνση από τους ανεπτυγμένους κόμβους αισθητήρων αποφασίζει την μελλοντική του κατεύθυνση για εξερεύνηση. Όταν το robot δεν μπορεί να λάβει μηνύματα για προτεινόμενη κατεύθυνση, μετά από συγκεκριμένη απόσταση προς μια κατεύθυνση, αναπτύσσει ένα ακόμη κόμβο αισθητήρα για να βελτιώσει την τοπική κάλυψη της περιοχής. Ο αλγόριθμος αυτός δεν προϋποθέτει την επικοινωνία μεταξύ των ανεπτυγμένων κόμβων. Όλες οι αποφάσεις γίνονται από το robot το οποίο επικοινωνεί άμεσα με τους γειτονικούς κόμβους αισθητήρων. Για την επιστροφή του σε μια συγκεκριμένη περιοχή έχει προταθεί η χρήση κατανεμημένου υπολογισμού. Η στρατηγική ανάπτυξης και η πολιτική επιδιόρθωσης του δικτύου μπορούν να βελτιωθούν, αν χρησιμοποιηθεί multi-hop επικοινωνία μεταξύ των κόμβων αισθητήρων.

Στο πρωτόκολλο **Bidding** [49], οι στατικοί αισθητήρες υπολογίζουν τις τοπικές τους τρύπες κάλυψης χρησιμοποιώντας διαγράμματα Voronoi, ενώ οι κινητοί αισθητήρες υπολογίζουν τις τρύπες κάλυψης τους στις τρέχουσες τοποθεσίες τους μόνο αν αποφασίσουν να μετακινηθούν. Οι στατικοί κόμβοι πλειοδοτούν τους κινητούς κόμβους αισθητήρων σύμφωνα με το μέγεθος των τοπικών τους τρυπών κάλυψης. Ακολούθως ο κινητός κόμβος συγκρίνει τις πλειοδοτήσεις και αποφασίζει να κινηθεί μόνο αν η ψηλότερη πλειοδοσία που παραλήφθηκε έχει μέγεθος τρύπας κάλυψης μεγαλύτερο από αυτήν που θα δημιουργηθεί σε περίπτωση που κινηθεί ο κινητός κόμβος. Οι πλειοδοσίες γίνονται broadcast τοπικά μέχρι και δύο

hops και οι στατικοί κόμβοι αισθητήρων στέλνουν τους κινητούς κόμβους στις μακρινότερες κορυφές του Voronoi πολυγώνου τους.

Το πρωτόκολλο **COVEN** [4] ακολουθεί διαδικασία ανάπτυξης δυο βημάτων και παρέχει μια στρατηγική για τη συνεργασία των στατικών κόμβων αισθητήρων για τον υπολογισμό του επιπρόσθετου αριθμού των κινητών κόμβων που χρειάζονται να αναπτυχθούν σε βέλτιστες τοποθεσίες έτσι ώστε να βελτιστοποιηθεί η κάλυψη του μικτού δικτύου ασύρματων αισθητήρων. Στην πρώτη φάση, κατασκευάζει ένα Voronoi διάγραμμα που καλύπτει την περιοχή του δικτύου. Αν ένας κόμβος βρίσκεται πολύ κοντά στα όρια του πεδίου αίσθησης του δικτύου (το πρωτόκολλο προϋποθέτει ο κάθε κόμβος να γνωρίζει τη θέση του), βρίσκει την τοποθεσία των ορίων του πεδίου αίσθησης του δικτύου και θεωρεί τη γραμμή που αντιπροσωπεύει τα όρια ως την άκρια του Voronoi πολυγώνου του. Στη δεύτερη φάση, το πρωτόκολλο υπολογίζει πόσοι επιπρόσθετοι κινητοί κόμβοι χρειάζονται να εισαχθούν στο δίκτυο για να υπάρξει ικανοποιητική κάλυψη. Το πρωτόκολλο COVEN δεν είναι αρκετά εξελίξιμο σε περίπτωση που η αρχική τοποθεσία των στατικών κόμβων είναι αραιή. Σε αυτή την περίπτωση, μετά την ανάπτυξη των επιπρόσθετων κινητών κόμβων αισθητήρων, μερικοί από αυτούς επιλέγονται για να κινηθούν μέσα στο δίκτυο και να εντοπίσουν τους απομονωμένους κόμβους. Όταν τους εντοπίσουν μπορούν να ζητηθούν από την εφαρμογή για επιπλέον κινητούς κόμβους αισθητήρων σε αυτές τις τοποθεσίες.

Οι περισσότερες από τις προτεινόμενες λύσεις υποθέτουν απλοποιημένο εύρος ομοιόμορφης αίσθησης για τους κόμβους αισθητήρων (μορφή δίσκων) και ότι όλοι οι κόμβοι γνωρίζουν την τοποθεσία τους. Το μοντέλο με κυκλικό εύρος αίσθησης δεν είναι κατάλληλο για όλων των ειδών μετρήσεων των αισθητήρων(πχ θερμοκρασία, υγρασία, θόρυβος κλπ).

Category	**Approach**	**Proposed Solution**	**Main Assumptions**	**Characteristics**
Mobile Sensors	Computational Geometry	VEC, VOR, Minmax [48]	Location Information	Localized, scalable, distributed
		Co-Fi [19]	Location Information, Nodes can predict their death	Single coverage based, Residential energy considerations
	Virtual Forces	Potential Fields[22]	Range and Bearing	Scalable, distributed. No local communication required for localization or movement
		DSS, IDCA [21]	Location information	Scalable distributed, Residual energy
	Sequential	Incremental [23]	Line of sigh for Localization	Centralized. Bidirectional communication with base station.
	Grid	snake-like cascading replacement [26]	Location information	Scalable, distributed, low cost solution
Hybrid Sensors	Single mobile Sensor	UAV [16]	Predetermined topology information	Flying robot for deployment and network repair. Inaccuracies using aerial deployment
		Single Robot [7]	Location information	Distributed, no multi-hop communication for network deployment and repair.
	Multiple Mobile Sensors	Bidding Protocol[49]	Location information	Uses Voronoi diagram for single coverage requirement
		COVEN [4]	Location information	Uses Voronoi diagram for single coverage requirement
Static Sensors	Multiple coverage	CCP [50]	Location information, uniform sensing disk	Configurable degree of coverage, calculated by intersection points of sensing circles
		k-UC, k-NC [24]	Location information	Perimeter coverage, non – disk sensing model supported
		Differentiated [27]	Location information, time Synchronization	Grid based differentiated degree of coverage
	Single coverage	OGDC [29]	Location information, uniform sensing disk	Residual energy consideration
		Sponsored Area [40]	Location information	Sensor based coverage calculations, non – disk sensing model supported
		Extended Sponsored Area [25]	Location information, time Synchronization	Uniform disk sensing model

Πίνακας 4.1: Προτεινόμενες λύσεις που αφορούν προβλήματα τρυπών κάλυψης

Κεφαλαιο 5

Ανάπτυξη αλγορίθμου ανίχνευσης και ελαχιστοποίησης τρυπών κάλυψης με την προσθήκη επιπρόσθετων κινητών κόμβων αισθητήρων

5.1 Παρουσίαση Προβλήματος

Σε μια περιοχή N x M αναπτύσσονται τυχαία κόμβοι αισθητήρων. Οι κόμβοι αισθητήρων έχουν κυκλική ομοιόμορφη εμβέλεια αίσθησης R_{SENSE} και εύρος ασύρματης επικοινωνίας $R_{RF} > 2R_{SENSE}$. Στο δίκτυο υπάρχει και ένα sink, στο οποίο οι κόμβοι αισθητήρων στέλλουν ανά τακτά χρονικά διαστήματα πληροφορίες.

Επειδή η ανάπτυξη του δικτύου έγινε με τυχαίο τρόπο, υπάρχουν κάποιες περιοχές μέσα στο δίκτυο στις οποίες δεν υπάρχει κάποιος κόμβος αισθητήρα για την ανίχνευση γεγονότων. Δηλαδή στο δίκτυο υπάρχουν τρύπες κάλυψης. Για τη σωστή μελέτη της περιοχής στην οποία αναπτύχθηκαν οι κόμβοι, πρέπει να εξαλειφθούν οι τρύπες κάλυψης που υπάρχουν.

5.2 Η γενική ιδέα

Με κάποιο τρόπο θα στέλνονται οι συντεταγμένες και η εμβέλεια αίσθησης R_{SENSE} του κάθε κόμβου αισθητήρα στο sink και αυτό με τη σειρά του, θα υπολογίζει τις βέλτιστες θέσεις μέσα στο δίκτυο, στις οποίες με την ανάπτυξη επιπρόσθετου κόμβου θα ελαχιστοποιούνται οι τρύπες κάλυψης. Στη συνέχεια, το sink θα ενημερώνει τους κινητούς κόμβους με τις βέλτιστες συντεταγμένες ανάπτυξης που υπολόγισε και αυτοί με τη σειρά τους θα υπολογίζουν το δρομολόγιο που θα ακολουθήσουν και θα κατευθύνονται εκεί.

5.3 Περιορισμοί – Προϋποθέσεις

1. Όλοι οι κόμβοι αισθητήρων έχουν ένα ομοιόμορφο δίσκο αίσθησης.
2. Το Sink γνωρίζει τις διαστάσεις της περιοχής του δικτύου.
3. Κάθε κόμβος γνωρίζει τη θέση του σε σχέση με την περιοχή του δικτύου.
4. Στην περιοχή ανάπτυξης του δικτύου δεν υπάρχουν φυσικά εμπόδια ή αν υπάρχουν είναι γνωστά από το sink.
5. Κάθε αισθητήρας μπορεί να βρίσκεται σε μια από τρεις καταστάσεις (κατάσταση μετάδοσης μηνύματος, κατάσταση παράληψης μηνύματος, κατάσταση αίσθησης γεγονότος).
6. Ο κάθε κινητός κόμβος αισθητήρας μπορεί να βρίσκεται σε μια από τέσσερις καταστάσεις (κατάσταση μετάδοσης μηνύματος, κατάσταση παράληψης μηνύματος, κατάσταση αίσθησης γεγονότος, κατάταση μετακίνησης).
7. Το δίκτυο δεν υποφέρει από τρύπες δρομολόγησης.

8. Τα sinks έχουν μεγάλη υπολογιστική ισχύ, μνήμη και μεγάλα αποθέματα ενέργειας.

5.4 Εντοπισμός τρυπών κάλυψης και βέλτιστων θέσεων για την ανάπτυξη επιπρόσθετων κόμβων αισθητήρων

Είναι αδύνατον όλοι οι κόμβοι αισθητήρων να μπορούν να επικοινωνούν άμεσα με το sink. Γι’ αυτό τον λόγο οι κόμβοι πρέπει να επικοινωνούν με το sink βάση κάποιου πρωτοκόλλου multi-hop δρομολόγησης πακέτων.

Το sink γνωρίζει τις διαστάσεις της περιοχής του δικτύου. Έτσι λαμβάνοντας τις συντεταγμένες και την εμβέλεια αίσθησης του κάθε κόμβου μπορεί να σχεδιάσει το χάρτη κάλυψης του δικτύου. Ο χάρτης στο sink θα αναπαριστάται από ένα grid N x M το οποίο θα είναι αρχικοποιημένο με 0 στην κάθε θέση.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Σχήμα 5.1: Αρχικοποιημένο grid N x M

Για κάθε κόμβο αισθητήρα που λαμβάνει τα στοιχεία του, το sink θα σχεδιάζει την αντίστοιχη του θέση πάνω στο grid αλλάζοντας τη θέση αυτή από 0 σε 1. Επίσης θα σχεδιάζει την εμβέλεια κάλυψής του με τον ίδιο τρόπο. Δηλαδή όσες θέσεις του grid βρίσκονται σε απόσταση μικρότερη ή ίση με R_{SENSE} θα μετατρέπονται σε 1.

Έστω ότι το sink λάβει τις συντεταγμένες και την εμβέλεια αίσθησης των πιο κάτω κόμβων:

Node 1: συντεταγμένες: 2, 9 εμβέλεια αίσθησης 3
Node 2: συντεταγμένες: 3, 3 εμβέλεια αίσθησης 3
Node 3: συντεταγμένες: 7, 8 εμβέλεια αίσθησης 3
Node 4: συντεταγμένες: 8, 4 εμβέλεια αίσθησης 3
Node 5: συντεταγμένες: 14, 9 εμβέλεια αίσθησης 3

Τότε το sink θα σχεδιάσει τον ακόλουθο χάρτη:

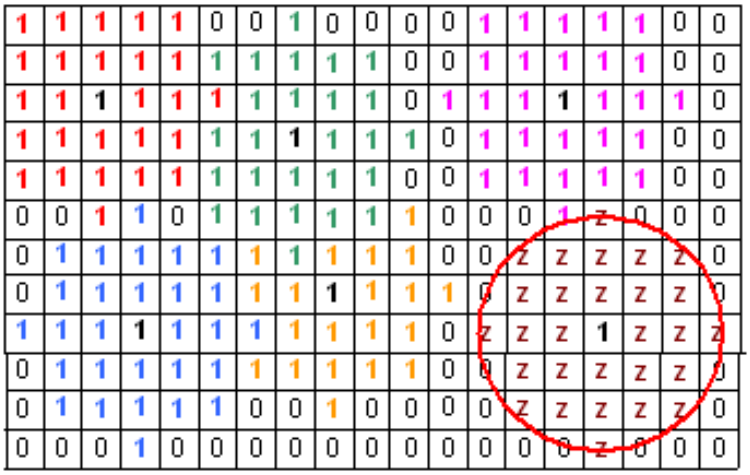
1	1	1	1	1	0	0	1	0	0	0	0	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0
0	0	1	1	0	1	1	1	1	1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Σχήμα 5.2: Ο χάρτης που θα σχεδιάσει το sink πάνω στο grid N x M

Με αυτό τον τρόπο φαίνονται οι τρύπες κάλυψης του δικτύου. Είναι οι περιοχές για τις οποίες αντιστοιχεί η τιμή 0 στην αντίστοιχη θέση του N x M grid.

Για τον εντοπισμό της βέλτιστης θέσης ανάπτυξης επιπρόσθετου κόμβου λαμβάνεται υπόψη η ακτίνα κάλυψης του κινητού κόμβου που θα προστεθεί. Υπολογίζονται δηλαδή πόσες θέσεις του grid μπορούν να καλυφθούν από την ακτίνα κάλυψης του κινητού κόμβου. Ακολούθως το sink ψάχνει να βρει τις συντεταγμένες της θέσης του grid, στις οποίες αν τοποθετηθεί ο επιπρόσθετος κόμβος θα μετατραπούν οι περισσότερες γύρω θέσεις από 0 σε 1.

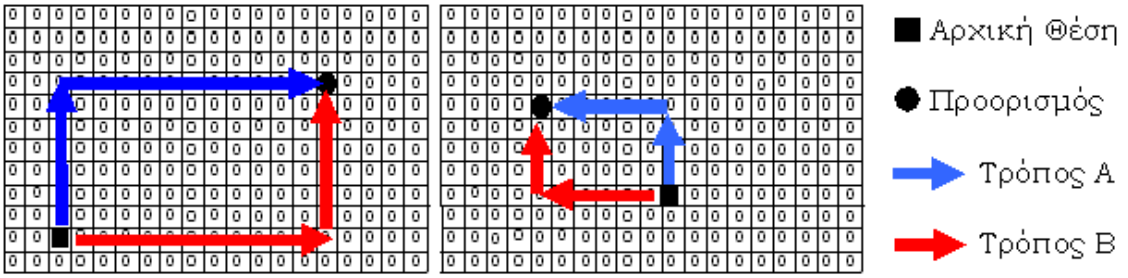
Αν για παράδειγμα η εμβέλεια κάλυψης του κινητού κόμβου που θα προστεθεί είναι 3, τότε η βέλτιστη θέση τοποθέτησης του σύμφωνα με το πιο πάνω παράδειγμα είναι στις συντεταγμένες (15,3) αν θεωρήσουμε ότι το πρώτο τετράγωνο κάτω αριστερά έχει συντεταγμένες (0,0).



Σχήμα 5.3: Βέλτιστη θέση για την ανάπτυξη επιπρόσθετου κόμβου

Η τοποθέτηση του αυτή είναι βέλτιστη, γιατί καλύπτει τη μεγαλύτερη ακάλυπτη περιοχή του χάρτη. Δηλαδή σε αυτές τις συντεταγμένες, αν τοποθετηθεί επιπρόσθετος κόμβος μετατρέπονται από 0 σε 1 οι περισσότερες θέσεις του grid.

5.5 Μετακίνηση του κινητού κόμβου για κάλυψη της τρύπας κάλυψης που εντοπίστηκε



Σχήμα 5.4: Πορεία δύο κατευθύνσεων

Ο πιο εύκολος τρόπος για να πάει ο κινητός κόμβος στον προορισμό του, αν η θέση προορισμού του κινητού κόμβου δεν έχει έστω και μια από τις συντεταγμένες της την ίδια με την αντίστοιχη της αρχικής του θέσης, είναι ακολουθώντας την πορεία δύο κατευθύνσεων.

Πορεία δύο κατευθύνσεων σε μια περιοχή ονομάζω τη μετακίνηση παράλληλα με τους νοητούς άξονές της. Την ονομάζω δύο κατευθύνσεων γιατί για να μετακινηθούμε από ένα σημείο σε ένα άλλο σημείο, το οποίο δεν έχει μια από τις συντεταγμένες του την ίδια με την αντίστοιχη του πρώτου σημείου, πρέπει να ακολουθήσουμε πορεία πρώτα παράλληλη με τον ένα νοητό άξονα και στη συνέχεια πορεία παράλληλη με τον άλλο νοητό άξονα. Η μετακίνηση αυτή μπορεί να επιτευχθεί με δύο τρόπους. Είτε ξεκινούμε κατευθυνόμενοι παράλληλα πρώτα με τον άξονα των x μέχρι να έχουμε την ίδια τεταγμένη με την θέση προορισμού μας και ακολούθως κατευθυνόμαστε παράλληλα με τον άξονα των ψ μέχρι να έχουμε την ίδια τετμημένη, είτε ξεκινούμε κατευθυνόμενοι παράλληλα πρώτα με τον άξονα των ψ μέχρι να έχουμε την ίδια τετμημένη με τη θέση προορισμού μας και ακολούθως κατευθυνόμαστε παράλληλα με τον άξονα των x μέχρι να έχουμε την ίδια τεταγμένη.

Για να μπορούμε όμως να κινηθούμε με αυτό τον τρόπο πρέπει να μην υπάρχουν εμπόδια στην περιοχή. Αυτό όμως δεν ισχύει, γιατί ακόμα και αν δεν υπάρχουν φυσικά εμπόδια στην περιοχή ανάπτυξης του δικτύου, οι ίδιοι οι κόμβοι αισθητήρων αποτελούν εμπόδιο για τον κινητό κόμβο, αφού θα πρέπει να κινηθεί με τέτοιο τρόπο ώστε να μην συγκρουστεί μαζί τους ή να τους παρασύρει

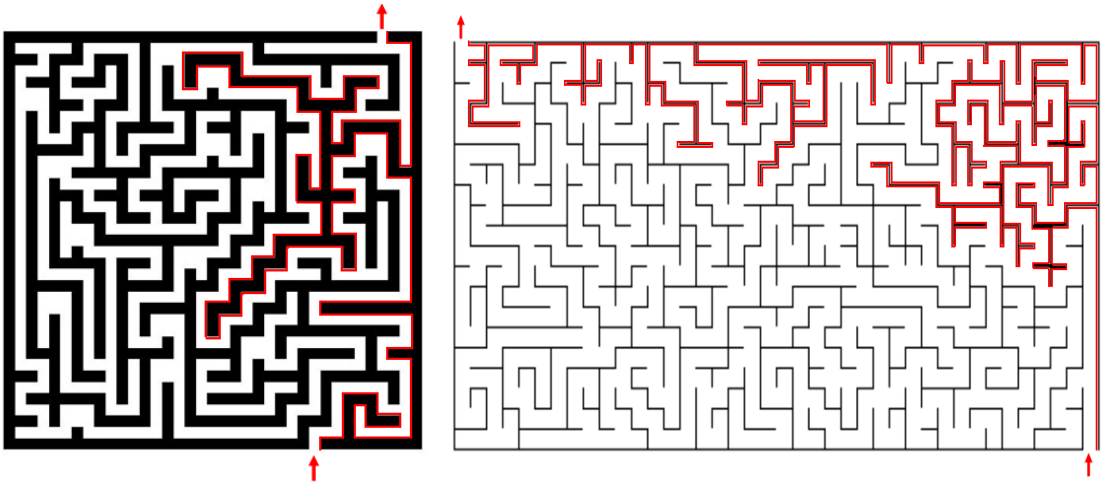
Ο κινητός κόμβος αισθητήρα δεν έχει τη δυνατότητα να ανιχνεύει τυχών εμπόδια τα οποία ίσως βρεθούν μπροστά του. Επομένως, πρέπει να μετακινηθεί με τέτοιο τρόπο ώστε να αποφύγει τα εμπόδια προχωρώντας προς τον προορισμό του.

Αφού το sink γνωρίζει ήδη την τοποθεσία του κάθε κόμβου, αυτό σημαίνει ότι γνωρίζει την τοποθεσία όλων των εμποδίων της περιοχής του δικτύου. Έτσι στέλλει στον κινητό κόμβο τις συντεταγμένες όλων των κόμβων που βρίσκονται αναπτυγμένοι στο δίκτυο καθώς και τις συντεταγμένες της τοποθεσίας που πρέπει να αναπτυχθεί. Έχοντας αυτές τις πληροφορίες, ο κινητός κόμβος, σχεδιάζει το χάρτη της περιοχής με τα εμπόδια για να μπορεί να υπολογίσει το δρομολόγιο του. Ο χάρτης του δρομολογίου θα αναπαριστάται από ένα grid N x M το οποίο θα είναι αρχικοποιημένο με 1 στη κάθε θέση. Στο grid θα τοποθετηθούν οι κόμβοι αισθητήρων της περιοχής, ανάλογα με τις συντεταγμένες τους στις αντίστοιχες θέσεις καθώς και χώρος γύρω από αυτούς ο οποίος θα αντιπροσωπεύει την περιοχή κινδύνου την οποία δεν πρέπει να πλησιάσει ο κινητός κόμβος, γιατί υπάρχει πιθανότητα σύγκρουσης. Στο grid οι θέσεις αυτές θα αναπαριστούνται με 0.

Αν είναι μεγάλη η πυκνότητα των κόμβων αισθητήρων σε αρκετές περιοχές του δικτύου, υπάρχει μεγάλη περίπτωση ο χάρτης για την εύρεση δρομολογίου που θα δημιουργηθεί να αναπαριστά ένα λαβύρινθο με πολλά αδιέξοδα.

Έτσι ο κινητός κόμβος πρέπει να είναι σε θέση να μετακινηθεί στον προορισμό του, αν υπάρχει έστω και ένα δρομολόγιο, πολύπλοκο ή μη, που να τον οδηγά εκεί.

Ένας αποτελεσματικός τρόπος, αλλά όχι και ο πιο αποδοτικός για την επίλυση ενός λαβύρινθου, είναι με τη μέθοδο του δεξιού χεριού. Δηλαδή το άτομο που πρέπει να διασχίσει το λαβύρινθο πρέπει να προχωρά και να αγγίζει συνέχεια το δεξί του χέρι στο δεξί τοίχο. Με αυτό τον τρόπο σίγουρα σε κάποια στιγμή στο μέλλον, θα φτάσει στην έξοδο του λαβύρινθου. Αυτή η μέθοδος είναι χρονοβόρα σε περίπτωση που ο λαβύρινθος είναι μεγάλος, αλλά εγγυάται την αποτελεσματικότητα της άμα υπάρχει έστω και μια έξοδος από το λαβύρινθο.



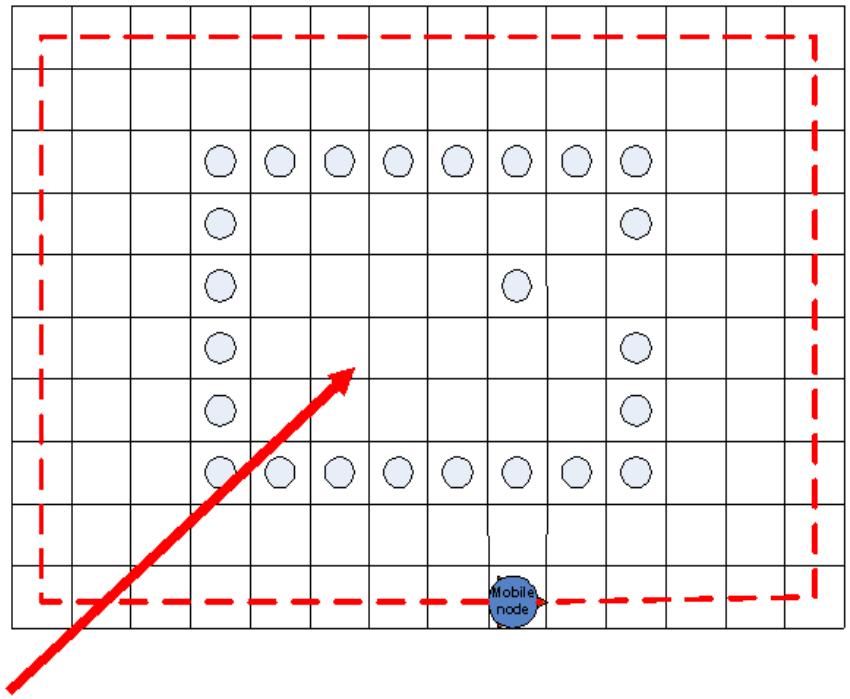
Σχήμα 5.5: Με τη μέθοδο του δεξιού χεριού μπορούμε να διασχίσουμε με επιτυχία οποιοδήποτε λαβύρινθο, αν υπάρχει έστω και μια έξοδος

Ο κινητός κόμβος θα κάνει τους υπολογισμούς για την εύρεση του δρομολογίου βασιζόμενο στην πιο πάνω μεθοδολογία. Ο κινητός κόμβος αισθητήρα θα ξεκινά από τις συντεταγμένες (0,0) και θα προσπαθεί να κινηθεί δεξιά σε κάθε διακριτό του βήμα. Εάν δεν μπορεί να κινηθεί προς τα δεξιά τότε θα προχωρά ένα διακριτό βήμα ευθεία. Αν δεν μπορεί να κινηθεί ούτε ευθεία, τότε στρίβει αριστερόστροφα 90 μοίρες έτσι ώστε η δεξιά του πλευρά να αντιμετωπίζει το εμπόδιο που βρισκόταν μπροστά του και επαναλαμβάνει τη διαδικασία προσπάθειας κίνησης από την αρχή.

Η μεθοδολογία αυτή από μόνη της δεν μπορεί να οδηγήσει τον κινητό κόμβο στον προορισμό του. Εγγυάται μόνο ότι ο κόμβος θα μπορεί να βρει διέξοδο από κάποιο λαβύρινθο.

Για να μπορεί ο κινητός κόμβος να φτάσει με επιτυχία στον προορισμό του πρέπει η πιο πάνω μεθοδολογία να συνδυαστεί με την πορεία δύο κατευθύνσεων. Δηλαδή για κάθε βήμα του κινητού κόμβου, πριν από την προσπάθεια μετακίνησης προς τα δεξιά, θα πρέπει να ελέγχεται αν μπορεί να φτάσει στον προορισμό του μέσο της πορείας δύο κατευθύνσεων.

Η μεθοδολογία αυτή δεν δουλεύει σε μεμονωμένες περιπτώσεις κατά τις οποίες ο τελικός προορισμός βρίσκεται στο κέντρο της περιοχής περιτριγυρισμένος από εμπόδια με κάπως ιδιόρρυθμο τρόπο.



Σχήμα 5.6: Περίπτωση όπου η μέθοδος θα πέσει σε ατέρμονα βρόχο.

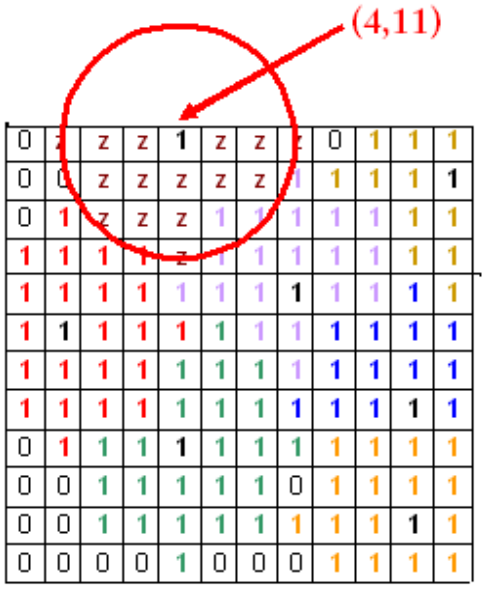
Μια λύση είναι ο κινητός κόμβος αν διαπιστώσει ότι έχει ξαναπεράσει από το ίδιο σημείο με την ίδια κατεύθυνση, να αρχίσει να κινείται τυχαία μέσα στο δίκτυο για μια μικρή χρονική περίοδο και μετά να ξαναρχίσει να κινείται χρησιμοποιώντας την μέθοδο δρομολόγησης που περιγράφηκε. Με αυτό τον τρόπο θα μπορέσει να βγει από ατέρμονες βρόχους και να φτάσει στον τελικό του προορισμό.

5.6 Παρουσίαση σεναρίου με πολύπλοκο δρομολόγιο κινητού κόμβου για κάλυψη τρύπας κάλυψης

Έστω ή περιοχή του δικτύου που παρακολουθείται είναι 12 x 12 και σε αυτήν είναι ανεπτυγμένοι 6 κόμβοι αισθητήρων με ακτίνα κάλυψης 3. Οι κόμβοι αισθητήρων θα επικοινωνήσουν με το sink με τη βοήθεια ενός multi-hop πρωτοκόλλου δρομολόγησης και θα του στείλουν τις συντεταγμένες τους και την ακτίνα κάλυψης τους :

Node 1: συντεταγμένες: 1, 6 εμβέλεια αίσθησης 3
Node 2: συντεταγμένες: 4, 3 εμβέλεια αίσθησης 3
Node 3: συντεταγμένες: 7, 7 εμβέλεια αίσθησης 3
Node 4: συντεταγμένες: 10, 1 εμβέλεια αίσθησης 3
Node 5: συντεταγμένες: 10, 4 εμβέλεια αίσθησης 3
Node 6: συντεταγμένες: 11, 10 εμβέλεια αίσθησης 3

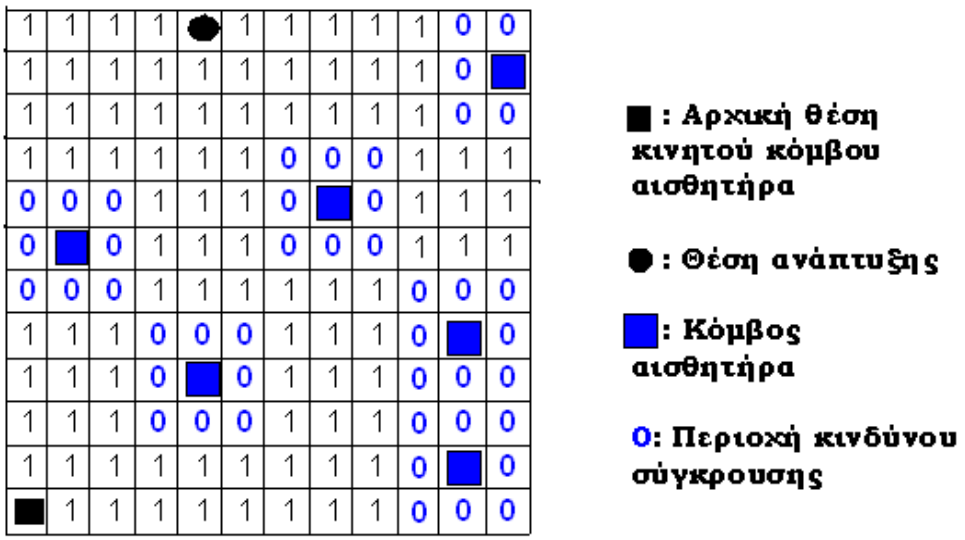
Έτσι, το sink θα σχεδιάσει το χάρτη κάλυψης και θα βρει τις τρύπες κάλυψης και θα υπολογίσει τη βέλτιστη θέση για την ανάπτυξη ενός κινητού κόμβου αισθητήρα, ο οποίος θα έχει ακτίνα κάλυψης 3.



Σχήμα 5.7: Η βέλτιστη θέση για την ανάπτυξη επιπρόσθετου κόμβου

Το sink θα υπολογίσει ότι η βέλτιστη θέση για την ανάπτυξη επιπρόσθετου κόμβου αισθητήρα είναι στις συντεταγμένες (4, 11) και θα στείλει μήνυμα στον κινητό κόμβο για να μετακινηθεί σε αυτή την τοποθεσία στέλνοντας του και τις συντεταγμένες όλων των κόμβων που είναι ανεπτυγμένα στο δίκτυο.

Έχοντας αυτές τις πληροφορίες, ο κινητός κόμβος θα σχεδιάσει το χάρτη δρομολόγησης, αφού τοποθετήσει σε αυτόν τις θέσεις των κόμβων αισθητήρων του δικτύου και προσθέσει γύρω από κάθε κόμβο τη δική του περιοχή κίνδυνου όπως φαίνεται πιο κάτω:



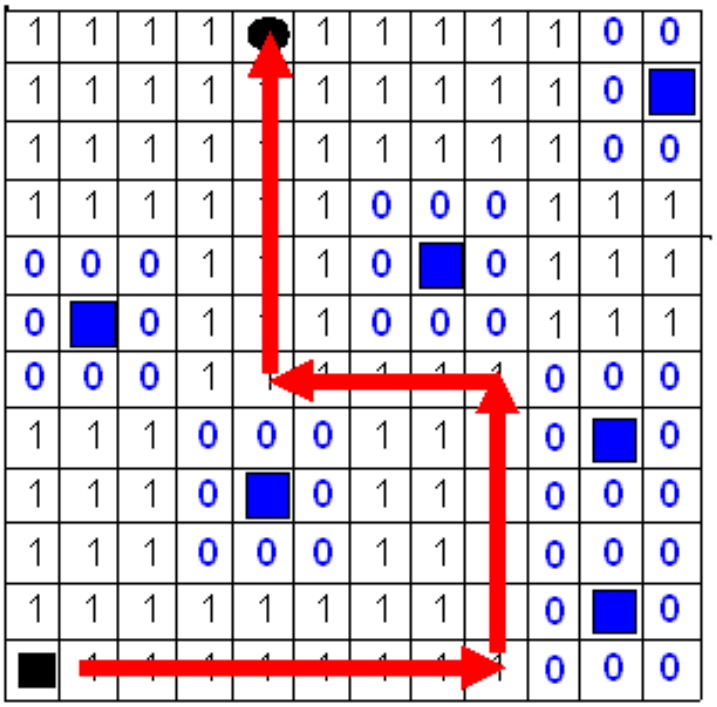
Σχήμα 5.8: Ο χάρτης δρομολόγησης. Στις θέσεις που υπάρχει 1 μπορεί να κινηθεί για να φτάσει στον προορισμό του ο κινητός κόμβος

Ο κινητός κόμβος αισθητήρα πρέπει να ξεκινήσει από την θέση με συντεταγμένες (0,0) και να κατευθυνθεί στη θέση με συντεταγμένες (4, 11) χωρίς να περάσει πάνω από την περιοχή κινδύνου σύγκρουσης κανενός από τους ανεπτυγμένους κόμβους. Με πιο απλά λόγια, ο κινητός κόμβος αισθητήρα δικαιούται να κινηθεί σε θέσεις στις οποίες αντιστοιχεί ο αριθμός 1 μέσα στο grid.

Όταν ο κινητός κόμβος σχεδιάσει το χάρτη δρομολόγησης, προσομοιώνει την κίνηση του μέσα στην περιοχή του δικτύου, και συνδυάζοντας τη μέθοδο

πορείας δύο κατευθύνσεων με τη μέθοδο του δεξιού χεριού για έξοδο από λαβύρινθο υπολογίζει το δρομολόγιο που πρέπει να ακολουθήσει.

Πιο κάτω παρουσιάζεται το δρομολόγιο που πρέπει να ακολουθήσει ο κινητός κόμβος αισθητήρα σύμφωνα με τους υπολογισμούς του:



Σχήμα 5.9: Το μονοπάτι που υπολόγισε το sink για τη μετακίνηση του κινητού κόμβου στον προορισμό του.

Εφόσον ο κινητός κόμβος έχει υπολογίσει με επιτυχία το δρομολόγιο που πρέπει να ακολουθήσει για να μεταβεί από τη θέση με συντεταγμένες (0, 0) στη θέση με συντεταγμένες (4, 11), το μόνο που μένει είναι να επικοινωνήσει με την κινητή πλατφόρμα στην οποία είναι ενσωματωμένος και να της μεταβιβάσει το δρομολόγιο. Αφού φτάσει ο κινητός κόμβος στη τελική του θέση, συμπεριφέρεται όπως και οι στατικοί κόμβοι του δικτύου. Ανιχνεύει δηλαδή την περιοχή του και μαζί με τα δεδομένα ανίχνευσης του στέλλει και τις συντεταγμένες του.

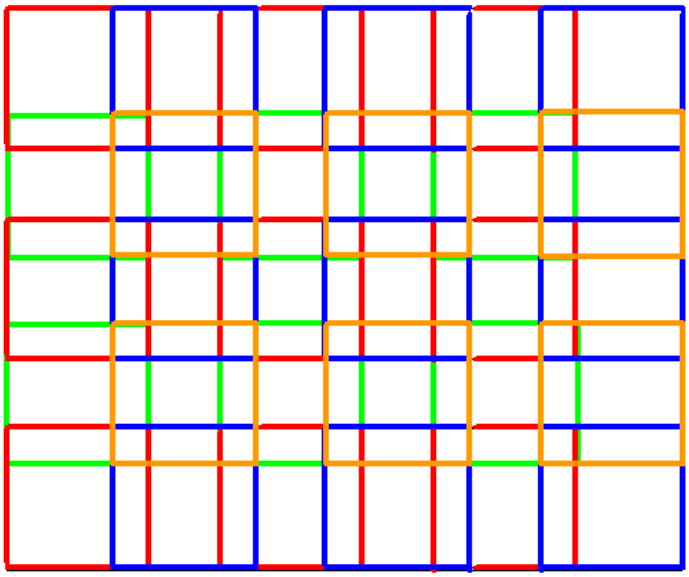
Ο αλγόριθμος επαναλαμβάνεται μέχρι το δίκτυο να είναι καλυμμένο στον επιθυμητό βαθμό κάλυψης (μέχρι δηλαδή το μέγεθος των υπαρκτών τρύπων κάλυψης να είναι μικρότερο από ένα threshold).

Επιπλέον ο αλγόριθμος τρέχει περιοδικά ανά διαστήματα, ώστε σε περίπτωση που παρουσιαστεί μια νέα τρύπα κάλυψης από θάνατο οποιουδήποτε κόμβου του δικτύου, να εντοπιστεί και να σταλεί νέος κόμβος για να τον αντικαταστήσει.

5.7 Επέκταση του αλγόριθμου.

Ο αλγόριθμος αυτός είναι επεκτάσιμος ανάλογα με τις υπολογιστηκές ικανότητες του sink. Θεωρητικά το sink θα έχει την δυνατότητα να επεξεργαστεί και να κάνει τέτοιου είδους υπολογισμούς, αφού πρέπει να έχει την υπολογιστηκή ικανότητα να επεξεργάζεται τα δεδομένα αίσθησης των αισθητήρων του δικτύου. Αν η περιοχή ανάπτυξης του δικτύου είναι πάρα πολύ μεγάλη και οι υπολογιστικές δυνατότητες του sink δεν αρκούν, ο πιο πάνω αλγόριθμος πρέπει να επεκταθεί έτσι ώστε να μπορεί να δουλέψει με επιτυχία. Ο αλγόριθμος μπορεί να τροποποιηθεί στον πιο κάτω:

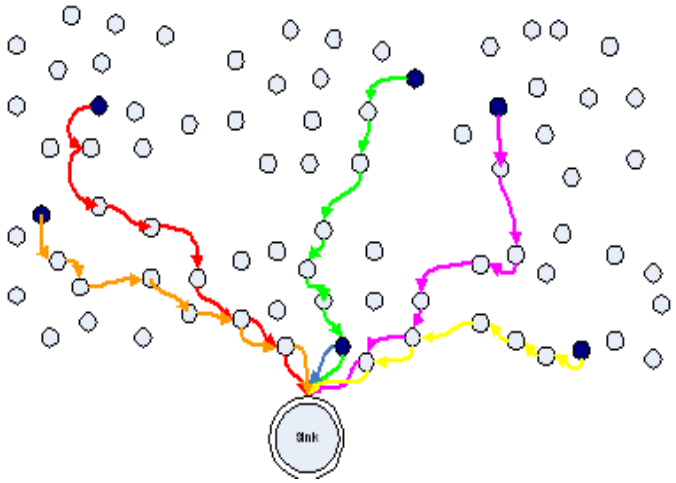
Το δίκτυο θα χωριστεί σε μικρότερες υποπεριοχές, οι οποίες θα υπερκαλύπτονται στα όρια τους με τις γειτονικές τους υποπεριοχές. Το Sink κάνει broadcast τα όρια των υποπεριοχών, ώστε ο κάθε κόμβος να γνωρίζει σε ποια υποπεριοχή βρίσκεται. Για κάθε υποπεριοχή, ο κόμβος με τα μεγαλύτερα αποθέματα ενέργειας θα ορίζεται αρχηγός της. Η επιλογή του αρχηγού της κάθε υποπεριοχής θα γίνεται σε συγκεκριμένες χρονικές περιόδους σε όλες τις υποπεριοχές. Οι υπόλοιποι κόμβοι της υποπεριοχής θα στέλνουν τις πληροφορίες τους στον αρχηγό της κάθε υποπεριοχής (με ένα πρωτόκολλο multihop δρομολόγησης όπως το CTP [18]) και αυτός με τη σειρά του θα υπολογίζει τις τρύπες κάλυψης της υποπεριοχής που είναι υπεύθυνος.



Σχήμα 5.10: Η περιοχή του δικτύου χωρίζεται σε υποπεριοχές οι οποίες υπερκαλύπτονται.

Για να γνωρίζει το sink ποιοι κόμβοι είναι αρχηγοί των υποπεριοχών, πρέπει οι κόμβοι – αρχηγοί των υποπεριοχών, μετά τον ορισμό τους να ενημερώνουν το sink με “leadership message”. Παραλαμβάνοντας το sink “leadership message” που περιέχει το node Id του κόμβου αρχηγού της κάθε υποπεριοχής και την υποπεριοχή στην οποία είναι υπεύθυνος, ενημερώνεται για τους αρχηγούς των υποπεριοχών του δικτύου.

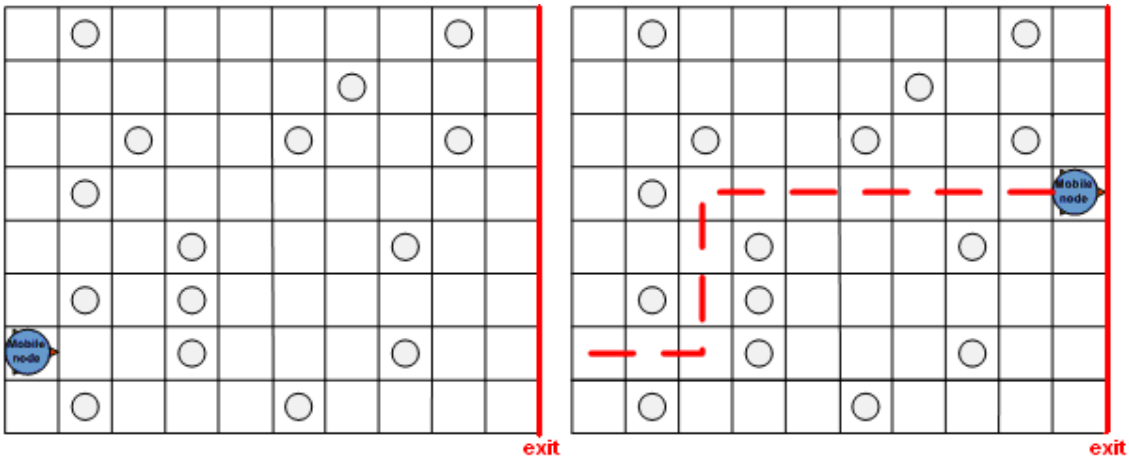
Ο αρχηγός της κάθε υποπεριοχής, αφού υπολογίσει τις τρύπες κάλυψης της υποπεριοχής του με τον αλγόριθμο που ανάλυσα πιο πάνω, στέλνει στο sink τις συντεταγμένες ανάπτυξης επιπρόσθετου κόμβου. Το sink τότε ενημερώνει ένα κινητό κόμβο για τις συντεταγμένες ανάπτυξης του και για τον αρχηγό της κάθε υποπεριοχής



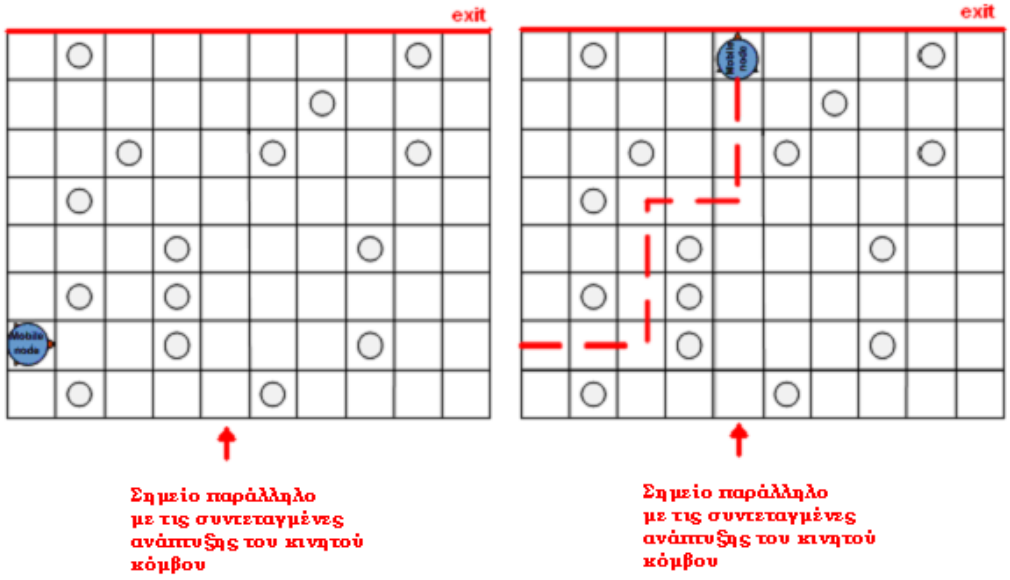
Σχήμα 5.11: Ο αρχηγός της κάθε υποπεριοχής στέλνει“leadership message” στο sink.

Ο κινητός κόμβος θα προσπαθεί να φτάσει στον προορισμό του ακολουθώντας πορεία παράλληλη με τον άξονα των x, με στόχο να φτάσει στην υποπεριοχή που πρέπει να αναπτυχθεί. Αν φτάσει σε σημείο που να έχει το ίδιο x με το σημείο ανάπτυξής του, αλλά δεν βρίσκεται στην υποπεριοχή ανάπτυξης του, τότε αλλάζει πορεία και αρχίζει να κατευθύνεται με πορεία παράλληλη με τον άξονα των y προς την υποπεριοχή που πρέπει να αναπτυχθεί. Αν φτάσει σε σημείο που να έχει το ίδιο y με το σημείο ανάπτυξής του, αλλά δεν βρίσκεται στην υποπεριοχή ανάπτυξής του, τότε αλλάζει πάλι πορεία, και ξανα-αρχίζει πορεία παράλληλη με τον άξονα των x προς την υποπεριοχή ανάπτυξης του.
Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου ο κινητός κόμβος να φτάσει στην υποπεριοχή ανάπτυξής του. Με τον τρόπο αυτό, ο κινητός κόμβος θα πορεύεται διαμέσου των υποπεριοχών για να μεταβεί στον προορισμό του. Με την είσοδο του σε μια υποπεριοχή, θα σταματά την πορεία του και θα ζητά από τον αρχηγό της υποπεριοχής πληροφορίες για την υποπεριοχή του (τοποθεσίες εμποδίων – κόμβων αισθητήρων). Αφού πληροφορηθεί και ανανεώσει τον χάρτη του, ο κινητός κόμβος θα διασχίζει την υποπεριοχή για να μεταβεί στην επόμενη υποπεριοχή αποφεύγοντας τα εμπόδια που θα συναντά. Αν για παράδειγμα μπει από το αριστερό τμήμα μιας υποπεριοχής, τότε στόχος του κινητού κόμβου είναι να βγεί από το δεξιό τμήμα της υποπεριοχής, εκτός αν φτάσει σε σημείο το οποίο να έχει το ίδιο x ή y με το

σημείο ανάπτυξής του. Αν φτάσει σε τέτοιο σημείο, τότε ο στόχος του κινητού κόμβου αλλάζει και πρέπει να βγεί είτε από το πάνω, είτε από το κάτω τμήμα της υποπεριοχής ανάλογα με τις συντεταγμένες που πρέπει να αναπτυχθεί. Τα εμπόδια θα τα αποφεύγει με την μέθοδο του δεξιού χεριού σε συνδιασμό με την πορείας δύο κατευθύνσεων.



Σχήμα 5.12: Διάσχιση κινητού κόμβου από υποπεριοχή του δικτύου προς την κατεύθυνση της υποπεριοχής που θα αναπτυχτεί



Σχήμα 5.12: Διάσχιση κινητού κόμβου από υποπεριοχή του δικτύου προς την κατεύθυνση της υποπεριοχής που θα αναπτυχτεί με αλλαγή στόχου διάσχισης

Όταν φτάσει στην υποπεριοχή ανάπτυξής του ακολουθεί την μέθοδο του δεξιού χεριού σε συνδιασμό με την πορείας δύο κατευθύνσεων με στόχο την θέση ανάπτυξής του.

Με αυτή την επέκταση, ο αλγόριθμος μπορεί να τρέξει κατανεμημένα σε μεγαλύτερου μεγέθους δίκτυα.

5.8 Ανάλυση πολυπλοκότητας του αλγορίθμου και της επέκτασής του

Ένας αλγόριθμος πρέπει να εργάζεται σωστά για κάθε σύνολο δεδομένων εισόδου, δηλαδή για κάθε στιγμιότυπο του πεδίου ορισμού του προβλήματος που λύνει και πρέπει να είναι αποδοτικός

Εδώ θα ασχοληθούμε με την ανάλυση και τον υπολογισμό της πολυπλοκότητας χρόνου και χώρου του αλγορίθμου που παρουσιάστηκε στο κεφάλαιο αυτό. Κάθε αλγόριθμος μπορεί να μελετηθεί εμπειρικά μετρώντας το χρόνο και το χώρο εκτέλεσης του σε συγκεκριμένο υπολογιστή. Θεωρητικά μπορούμε να υπολογίσουμε το χρόνο και το χώρο που απαιτεί ο αλγόριθμος σαν συνάρτηση του μεγέθους των εξεταζομένων στιγμιότυπων.

Η θεωρητική προσέγγιση υπολογισμού της αποτελεσματικότητας ενός αλγορίθμου δεν εξαρτάται από το υλικό του υπολογιστή (μνήμη, cashe κλπ), από τη γλώσσα προγραμματισμού, ούτε από το μεταφραστή. Γι' αυτόν το λόγο θεωρείται ότι είναι γενική.

Θεωρούμε ότι στην περιοχή του δικτύου μπορούν να χωρέσουν D_1 κόμβοι αισθητήρων κατά μήκος του χώρου και D_2 κόμβοι αισθητήρων κατά πλάτος. Δηλαδή συνολικά μπορούν να χωρέσουν D_1 x D_2 κόμβοι στο δίκτυο. Στην περιοχή του δικτύου είναι αναπτυγμένοι N κόμβοι αισθητήρων σε τυχαίες θέσεις και ο καθένας από αυτούς έχει διάμετρο αίσθησης R.

Σύμφωνα με τον αλγόριθμο θα αποσταλθούν στο sink N μηνύματα τα οποία θα περιέχουν τις πληροφορίες του κάθε κόμβου. Για κάθε μήνυμα που παραλαμβάνει το sink, διαβάζει τις πληροφορίες του κάθε κόμβου και τον σχεδιάζει στο χάρτη. Μέχρι εδώ έχουμε χρόνο εκτέλεσης της τάξεως του O(N x R x R) για την σχεδίαση του κάθε κόμβου και της εμβέλειας του στο χάρτη. Στη συνέχεια θα εκτελεστεί η συνάρτηση για εύρεση των τρυπών κάλυψης. Στη χειρότερη περίπτωση ο έλεγχος θα γίνει για όλες τις θέσεις του πίνακα ώστε να βρεθεί η μεγαλύτερη τρύπα κάλυψης. Επομένως, αυτή η διαδικασία θα έχει χρόνο εκτέλεσης της τάξεως του $O(D_1 \times D_2 \times R^2)$ για την εύρεση της μεγαλύτερης τρύπας κάλυψης που μπορεί να καλυφθεί με ένα κόμβο. Άρα, συνολικά ο χρόνος εκτέλεσης θα είναι $O(N \times R \times R) + O(D_1 \times D_2 \times R^2)$.

Το sink θα στείλει τις συντεταγμένες για ανάπτυξη επιπρόσθετου κόμβου και τις συντεταγμένες του κάθε κόμβου του δικτύου στον κινητό κόμβο. Ο κινητός κόμβος θα δημιουργήσει τον χάρτη δρομολόγησης του βασιζόμενος στις συντεταγμένες του κάθε κόμβου του δικτύου. Η διαδικασία αυτή έχει χρόνο εκτέλεσης O(N). Αφού δημιουργηθεί ο χάρτης δρομολόγησης, θα εκτελεστεί ο αλγόριθμος για την εύρεση του μονοπατιού που πρέπει να ακολουθήσει ο κινητός κόμβος. Η μέθοδος της πορείας δύο κατευθύνσεων σε συνδυασμό με την μέθοδο του δεξιού χεριού έχουν χρονική πολυπλοκότητα $O(D_1) + O(D_2)$. Αυτό γιατί στη χείριστη περίπτωση ο κινητός κόμβος πρέπει να κινηθεί στην άλλη μεριά του χάρτη. Έτσι ο συνολικός χρόνος εκτέλεσης θα είναι $O(D_1) + O(D_2) + O(N)$.

Για το κλείσιμο της μεγαλύτερης τρύπας κάλυψης που μπορεί να καλυφθεί από ένα επιπρόσθετο κόμβο χρειάζεται να σταλθούν N+1 μηνύματα.

Ο χώρος που χρειάζεται το sink και ο κινητός κόμβος για την εκτέλεση του αλγόριθμου τους είναι της τάξεως $O(N) + O(D_1 \times D_2)$. Χρειάζεται δηλαδή χώρος της τάξεως O(N) για την αποθήκευση των συντεταγμένων και της ακτίνας του κάθε κόμβου του δικτύου και χώρο της τάξεως $O(D_1 \times D_2)$ για την αποθήκευση του χάρτη της περιοχής του δικτύου.

Όσο αφορά την επέκταση του αλγορίθμου, το sink θα χωρίσει την περιοχή του δικτύου σε M υποπεριοχές και θα ενημερωθούν όλοι οι κόμβοι του δικτύου με τα όρια τις κάθε υποπεριοχής. Επομένως για την ενημέρωση όλων των κόμβων του δικτύου, θα χρειαστούν N μηνύματα. Κάθε αρχηγός της υποπεριοχής θα πρέπει να λάβει από την υποπεριοχή που είναι υπεύθυνος N_i μηνύματα από τους N_i κόμβους που βρίσκονται σε αυτή την υποπεριοχή (N_1 + N_2 +…..+ N_m > N αφού μερικοί κόμβοι μπορεί να ανήκουν ταυτόχρονα σε περισσότερες υποπεριοχές και N_i < N).

Έστω ότι η κάθε υποπεριοχή έχει διαστάσεις U_1 x U_2 όπου U_1 = D_1 / M και U_2 = D_2/M. Αυτό σημαίνει ότι ο κάθε αρχηγός της υποπεριοχής θα υπολογίσει την τρύπα κάλυψης της υποπεριοχής του σε χρόνο της τάξεως $O(N_i \times R^2)$ + $O(U_1 \times U_2 \times R^2)$.

Στη συνέχεια ο αρχηγός της κάθε υποπεριοχής θα ενημερώσει το sink με τις συντεταγμένες για την ανάπτυξη επιπρόσθετου κόμβου στην περιοχή του ή ότι δεν υπάρχει τρύπα κάλυψης. Έτσι το sink θα παραλάβει τα M μηνύματα των κόμβων αρχηγών και θα αρχίσει να στέλλει κινητούς κόμβους για να καλύψουν τα κενά στις συντεταγμένες που ζήτησαν οι κόμβοι αρχηγοί της κάθε υποπεριοχής.

Ο κάθε κινητός κόμβος σε κάθε υποπεριοχή που εισέρχεται θα ενημερώνει τον χάρτη του με τις πληροφορίες του μηνύματος από τον αρχηγό της υποπεριοχής. Για την ενημέρωση του χάρτη του, ο κάθε κινητός κόμβος θα χρειάζεται χρόνο της τάξεως $O(N_i)$. Για τον υπολογισμό του δρομολογίου μέσα σε κάθε υποπεριοχή χρειάζεται στη χειρότερη περίπτωση χρόνος της τάξεως $O(U_1)$ + $O(U_2)$. Έτσι ο συνολικός χρόνος εκτέλεσης θα είναι $O(M \times U_1)$ + $O(M \times U_2)$ + $O(M \times N_i)$. Αφού $M \times U_1 = D_1$, $M \times U_2 = D_2$ και $M \times N_i > N$, συνεπάγεται ότι οι υπολογισμοί στον κινητό κόμβο με την επέκταση του αλγορίθμου είναι περισσότεροι από τους υπολογισμούς του κινητού κόμβου με την πρώτη προσέγγιση.

Για το κλείσιμο της μεγαλύτερης τρύπας κάλυψης που μπορεί να καλυφθεί από ένα επιπρόσθετο κόμβο χρειάζεται να σταλθούν το πολύ $N+M*N_i+M+1+M$ μηνύματα.

Ο χώρος που χρειάζεται το sink για την εκτέλεση του αλγόριθμου είναι της τάξεως O(M). Χρειάζεται δηλαδή χώρος της τάξεως O(M) για την αποθήκευση των πληροφοριών των αρχηγών των υποπεριοχών και των ορίων που είναι υπεύθυνοι.

Ο χώρος που χρειάζεται ο κάθε αρχηγός των υποπεριοχών του δικτύου για την εκτέλεση του αλγόριθμου είναι της τάξεως $O(N) + O(U_1 \times U_2)$. Χρειάζεται δηλαδή χώρος της τάξεως O(N) για την αποθήκευση των συντεταγμένων του κάθε κόμβου της υποπεριοχής και χώρο της τάξεως $O(U_1 \times U_2)$ για την αποθήκευση του χάρτη της υποπεριοχής του δικτύου.

Ο χώρος που χρειάζεται ο κινητός κόμβος για την εκτέλεση του αλγόριθμου είναι της τάξεως $O(N) + O(U_1 \times U_2)$. Χρειάζεται δηλαδή χώρος της τάξεως O(N) για την αποθήκευση των συντεταγμένων του κάθε κόμβου της υποπεριοχής και χώρο της τάξεως $O(U_1 \times U_2)$ για την αποθήκευση του χάρτη της υποπεριοχής του δικτύου.

Στον υπολογισμό της πολυπλοκότητας της επέκτασης δεν λήφθηκε υπόψη η πολυπλοκότητα για την εκλογή του αρχηγού και τα μηνύματα που πρέπει να σταλούν για τον σκοπό αυτό.

5.9 Σύγκριση αλγορίθμων

Ένας άλλος αλγόριθμος, ο οποίος έχει ως στόχο τον εντοπισμό και την ελαχιστοποίηση των τρυπών κάλυψης παρουσιάζεται στο [54]. Ο αλγόριθμος αυτός χρειάζεται να γνωρίζει εκ των προτέρων τις διαστάσεις της ελεγχόμενης περιοχής, την εμβέλεια των κόμβων αισθητήρων και το ελάχιστο ποσοστό κάλυψης για το οποίο θεωρείται μια περιοχή καλυμμένη.

Ο αλγόριθμος δουλεύει με τον πιο κάτω τρόπο:

1. Επιλέγονται οι αισθητήρες των οποίων οι εμβέλειά τους είναι μέσα στην περιοχή που εξετάζεται.
2. Αφαιρείται από το συνολικό εμβαδό της περιοχής το εμβαδόν της εμβέλειας του κάθε αισθητήρα που επιλέχτηκε. Αν το ποσοστό της ακάλυπτης περιοχής είναι μεγαλύτερο από το επιτρεπτό ποσοστό ακάλυπτης περιοχής, τότε η περιοχή θεωρείται ακάλυπτη.
3. Η περιοχή που θεωρείται ακάλυπτη χωρίζεται στα 4 και για κάθε υποπεριοχή εκτελείται ξανά ο αλγόριθμος από το 1.

Αυτό επαναλαμβάνεται μέχρι η υποπεριοχή που εξετάζεται να θεωρείται καλυμμένη, είτε μέχρι οι διαστάσεις της υποπεριοχής να είναι οι μικρότερες επιτρεπτές.

Σε περίπτωση όπου υπάρχει υπερκάλυψη μιας περιοχής από δύο αισθητήρες λαμβάνονται μέτρα, ώστε να μην αφαιρεθεί ξανά εμβαδόν καλυμμένης περιοχής το οποίο αφαιρέθηκε ήδη από το ολικό εμβαδόν της περιοχής.

Σε περιπτώσεις που η εμβέλεια αισθητήρων βγαίνει έξω από τα όρια της περιοχής που εξετάζεται, ή που οι αισθητήρες είναι έξω από τα όρια της περιοχής που εξετάζεται, αλλά η εμβέλεια τους καλύπτει μέρος της περιοχής, τότε γίνονται οι κατάλληλοι υπολογισμοί, ώστε να προσθέτονται μόνο τα τμήματα της εμβέλειας των αισθητήρων τα οποία βρίσκονται μέσα στην περιοχή που εξετάζεται.

Αφού βρεθούν όλες οι θέσεις στις οποίες χρειάζεται να γίνει ανάπτυξη επιπρόσθετου κόμβου αισθητήρα, επιλέγεται μια από τις θέσεις αυτές για την ανάπτυξη του. Η επιλογή μπορεί να γίνει με δύο τρόπους. Είτε βάση των συντεταγμένων του robot, είτε ανάλογα με το κέντρο βάρους των ακάλυπτων περιοχών.

Ο αλγόριθμος αυτός υπερτερεί από τον αλγόριθμο που πρότεινα όσο αφορά την πολυπλοκότητα γιατί χρειάζεται χρόνο της τάξεως $O(Log_2(Min(D_1 , D_2)) \times N$

Όμως αν υπάρχει υπερκάλυψη μιας περιοχής από περισσότερους από δύο αισθητήρες, τότε το εμβαδόν της περιοχής που υπερκαλύπτεται αφαιρείται περισσότερες φορές από το ολικό εμβαδόν της περιοχής το οποίο μπορεί να οδηγήσει σε λάθος αποτέλεσμα, αφού μπορεί μια περιοχή που να είναι ακάλυπτη να θεωρηθεί καλυμμένη.

Ο αλγόριθμος που έχω προτείνει βρίσκει σε κάθε γύρο την βέλτιστη θέση ανάπτυξης ενός επιπρόσθετου κόμβου, ενώ αυτός ο αναδρομικός αλγόριθμος με το τέλος της εκτέλεσης του, βρίσκει τις θέσεις για όλες τις πιθανές θέσεις ανάπτυξης επιπρόσθετου κόμβου.

Ο κινητός κόμβος στον αλγόριθμό μου έχει την δυνατότητα να αποφεύγει κινητά εμπόδια χωρίς την χρήση επιπρόσθετων αισθητήρων, αλλά γνωρίζοντας μόνο τις συντεταγμένες τους σε αντίθεση με το robot που μεταφέρει τους κόμβους στον αναδρομικού αλγόριθμο που χρησιμοποιεί επιπρόσθετους αισθητήρες για την αποφυγή των εμποδίων.

5.10 Συμπεράσματα

Ό αλγόριθμος για την εύρεση τρυπών κάλυψης που παρουσιάστηκε προσπαθεί να μειώσει τις τρύπες κάλυψης όσο το δυνατόν γίνεται, με την προσθήκη μειωμένου αριθμού κινητών κόμβων αισθητήρων. Η μέθοδος για την δρομολόγηση του κινητού κόμβου στη βέλτιστη θέση ανάπτυξης, εγγυάται ότι ο κινητός κόμβος θα φτάσει με επιτυχία στον προορισμό του στις περισσότερες περιπτώσεις. Όμως, δεν εγγυάται ότι ο κινητός κόμβος θα ακολουθήσει το πιο σύντομο δρομολόγιο προς τον προορισμό του.

Η επέκταση του αλγόριθμου φορτώνει τους κόμβους αισθητήρων με επιπρόσθετη δουλειά, με αποτέλεσμα να καταναλώνεται η ενέργεια τους πιο

γρήγορα, αλλά αποφορτώνει το sink από υπολογισμούς σε περίπτωση που η υπολογιστική του ισχύ δεν είναι αρκετή. Γι' αυτό, όταν το δίκτυο έχει αρκετά αποθέματα ενέργειας, οι κόμβοι – αρχηγοί των υποπεριοχών θα μοιράζονται τους υπολογισμούς με το sink με τον κατανεμημένο υπολογισμό που προτάθηκε. Όταν το δίκτυο δε μπορεί να αντεπεξέλθει, λόγω μειωμένων αποθεμάτων σε ενέργεια, τότε το sink θα αναλαμβάνει όλους τους υπολογισμούς.

Κεφαλαιο 6

Υλοποίηση σε Πιλοτικό δίκτυο

6.1 Γενικά

Πρωταρχικός στόχος ήταν η χρήση του robot NXT [20,28] ως πλατφόρμα για τη δημιουργία κινητού κόμβου αισθητήρα. Το robot NXT έπρεπε με

κάποιο τρόπο να επικοινωνεί με έναν κόμβο αισθητήρα, ο οποίος θα ήταν ενσωματωμένος πάνω του και θα το καθοδηγούσε σύμφωνα με τις ανάγκες του δικτύου.

Σχήμα 6.1: Το NXT robot διαμορφωμένο για να χρησιμοποιηθεί ως πλατφόρμα μεταφοράς κόμβων αισθητήρων

Για τον προγραμματισμό του διαπίστωσα ότι η γλώσσα προγραμματισμού NXT-G (πρόσφερε προγραμματισμό σε γραφικό περιβάλλον με περιορισμένες δυνατότητες) δεν προσφερόταν για τους σκοπούς που ήθελα να χρησιμοποιήσω το robot. Το LEGO NXT robot μπορεί να προγραμματιστεί με NXT-G, RoboLab, NBC, NXC, RobotC, NI lab View toolkit, lejos NXT, pbLua και Lejos OSEK. Έτσι άλλαξα το firmware του ώστε να μπορώ να προγραμματίσω το robot σε γλώσσα lejos NXT [10], μια γλώσσα προγραμματισμού με σύνταξη, όπως της java. Ο προγραμματισμός του robot με αυτή τη γλώσσα δεν ήταν δύσκολος λόγω των ικανοποιητικών γνώσεων μου σε java και λόγω της πληθώρας των έτοιμων βιβλιοθηκών που παρείχε.

Μέσα από την έρευνα που έκανα, διαπίστωσα ότι ο μόνος τρόπος να επικοινωνήσει το robot NXT με τον κόμβο αισθητήρα micaz ήταν χρησιμοποιώντας το πρωτόκολλο I2C. Το robot NXT, μπορεί να υποστηρίξει τέτοιου είδους επικοινωνία, με την προϋπόθεση ότι θα την διαχειρίζεται αυτό, δηλαδή να είναι Master [11,12]. Επομένως,

υποχρεωτικά ο κόμβος micaz έπρεπει να ενεργεί ως slave. Για την εξοικείωση επικοινωνίας του robot με τον κόμβο αισθητήρα με τον τρόπο αυτό, αφού μελέτησα προσεκτικά το πρωτόκολλο, προσπάθησα να υλοποιήσω κάτι πιο απλό, δηλαδή την αποστολή μηνύματος από το robot NXT σε μια οθόνη LCD με τη χρήση του πρωτοκόλλου. Μετά την επίτευξη του πιο πάνω, έγινε έρευνα για το πώς μπορεί να προγραμματιστεί ο κόμβος αισθητήρα micaz ώστε να ενεργεί ως Slave. Δυστυχώς, μετά από αρκετή έρευνα και πολλές αποτυχημένες προσπάθειες, δεν κατάφερα να τον προγραμματίσω να ενεργεί ως Slave ούτε σε TinyOs1.x , ούτε σε TinyOs2.x. Μέχρι στιγμής δεν έχουν χρησιμοποιηθεί οι κόμβοι αυτοί ως slave με την χρήση αυτού του πρωτόκολλου I2C και γι' αυτό το λόγο δεν υπήρχαν κάπου οδηγίες για το πώς μπορεί να επιτευχθεί αυτό.

Έτσι έπρεπε να βρεθεί εναλλακτική πλατφόρμα για τη δημιουργία κινητού κόμβου. Το εργαστήριο διέθετε δύο robot boebot, έτσι ένα από αυτά τα robot χρησιμοποιήθηκε ως εναλλακτική λύση στο πρόβλημα την πλατφόρμας. Το robot αυτό έχει τη δυνατότητα να επικοινωνήσει σειριακά με έναν κόμβο αισθητήρα micaz. Για την εξοικείωση επικοινωνίας του robot με τον κόμβο αισθητήρα με τον τρόπο αυτό, έγραψα ένα απλό πρόγραμμα στη γλώσσα PBASIC για το robot boebot και ένα άλλο σε γλώσσα necC (για TinyOs1.x και για TinyOs2.x) για την πλοήγηση του boebot με τη χρήση κόμβου αισθητήρα micaz χρησιμοποιώντας RS232. Στη συνέχεια επέκτεινα τον κώδικά μου ώστε να γίνεται ασύρματη η πλοήγηση του boebot με τη χρήση δύο αισθητήρων micaz χρησιμοποιώντας RS232.

Αφού εξοικειώθηκα με την επικοινωνία του robot με ασύρματο κόμβο, μελέτησα αλγόριθμους για την multihop επικοινωνία των κόμβων αισθητήρων (Dissemination, Tymo, Collection και surge) [44]. Το πρωτόκολλο Dissemination [31] μεταφέρει αξιόπιστα μικρή ποσότητα δεδομένων σε κάθε κόμβο του δικτύου. Από την άλλη το πρωτόκολλο Collection [39] μεταφέρει μικρή ποσότητα δεδομένων από κάθε κόμβο του

δικτύου σε ένα συγκεκριμένο κόμβο – ρίζα. Το πρωτόκολλο Tymo [45] που βασίζεται στο πρωτόκολλο δρομολόγησης DYMO, επιτρέπει την αποστολή δεδομένων σε ένα συγκεκριμένο κόμβο. Το surge [43] χρησιμοποιεί ένα αλγόριθμο shortest-path-first με μόνο έναν κόμβο προορισμού και active two-way link estimation.

Επειδή το εργαστήριο διέθετε μικρό αριθμό κόμβων αισθητήρων micaz, έγιναν προσπάθειες ώστε να μπορούν να χρησιμοποιηθούν και οι τρείς κόμβοι αισθητήρων telosb που διέθετε. Αφού δοκίμασα να επικοινωνήσω με multihop επικοινωνία χρησιμοποιώντας micaz και telosb ξεχωριστά, προσπάθησα να αναπτύξω ένα ετερογενές δίκτυο ασύρματων κόμβων αισθητήρων αποτελούμενο από τους κόμβους αισθητήρων micaz και Telosb. Αυτό έγινε δυνατό μόνο σε TinyOs1.x. Για την ασύρματη επικοινωνία μεταξύ κόμβων micaz και κόμβων Telosb σε περιβάλλον TinyOs 2.x δοκίμασα το πρωτόκολλο IEEE 802.15.4 (zigbee), όμως δεν ήταν αξιόπιστο. Μετά από ολες αυτές τις πειραματικές δοκιμές, αποφάσισα να υλοποιήσω τον αλγόριθμο μου σε TinyOs2.x, χρησιμοποιώντας μόνο τους κόμβους αισθητήρων micaz. Ο κώδικας όλων των εφαρμογών που προγραμμάτισα βρίσκεται στα παραρτήματα Β και Δ.

Το κεφάλαιο αυτό ξεκινά με την επεξήγηση του πρωτοκόλλου I2C. Ακολουθεί η επεξήγηση για το πώς υλοποίησα την αποστολή μηνύματος από το NXT στην οθόνη LCD με τη χρήση του πρωτοκόλλου I2C. Στη συνέχεια γίνεται επεξήγηση του τρόπου υλοποίησης για την πλοήγηση του boebot με τη χρήση αισθητήρων micaz χρησιμοποιώντας RS232, του τρόπου υλοποίησης για την ασύρματη πλοήγηση BOE-BOT με τη χρήση δύο αισθητήρων micaz χρησιμοποιώντας RS232, της επίτευξης multihop επικοινωνίας μεταξύ αισθητήρων TelosB, της ανάπτυξης ετερογενούς δικτύου ασύρματων κόμβων αισθητήρων, της δημιουργίας επικοινωνίας ασύρματων κόμβων αισθητήρων μέσω του IEEE 802.15.4 (ZIGBEE) και της επίτευξης multihop επικοινωνίας μεταξύ αισθητήρων micaz. Το κεφάλαιο αυτό τελειώνει με την επεξήγηση της υλοποίησης αλγορίθμου

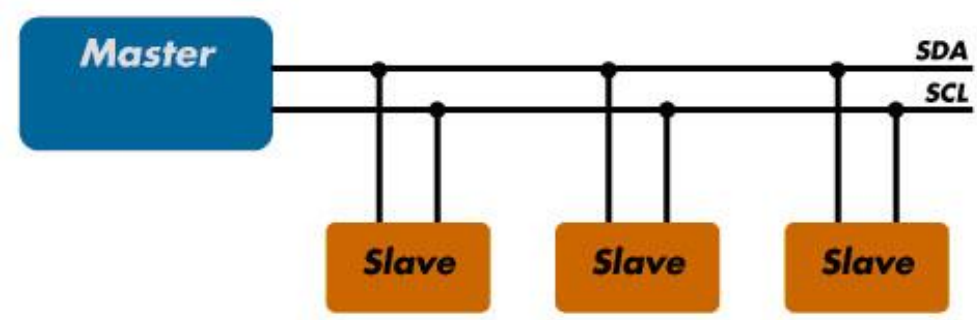
ανίχνευσης και ελαχιστοποίησης τρυπών κάλυψης με την προσθήκη επιπρόσθετων κινητών κόμβων αισθητήρων σε πιλοτικό δίκτυο.

6.2 Το πρωτόκολλο I2C

Σύμφωνα με τα [41, 46, 47, 51], ο δίαυλος I2C αποτελείται από 2 ενεργά καλώδια και μια γείωση. Τα ενεργά καλώδια ονομάζονται SDA (serial data line) και SCL (serial clock line) και είναι και τα δύο bi-directional.

Κάθε συσκευή η οποία είναι ενωμένη με το δίαυλο έχει τη δική της μοναδική διεύθυνση. Κάθε συσκευή μπορεί να δράσει είτε ως receiver ή και transceiver, ανάλογα με τις δυνατότητες του.

Ο δίαυλος I2C είναι ένας multi-master δίαυλος (Περισσότερα από ένα integrated circuits ικανά για την έναρξη μεταφοράς δεδομένων μπορούν να ενωθούν στον ίδιο δίαυλο). Το πρωτόκολλο I2C δηλώνει ότι το integrated circuit το οποίο αρχίζει τη μεταφορά δεδομένων στο δίαυλο θεωρείται το “Bus Master”. Έτσι όλα τα υπόλοιπα integrated circuits θεωρούνται ως “Bus Slaves”.

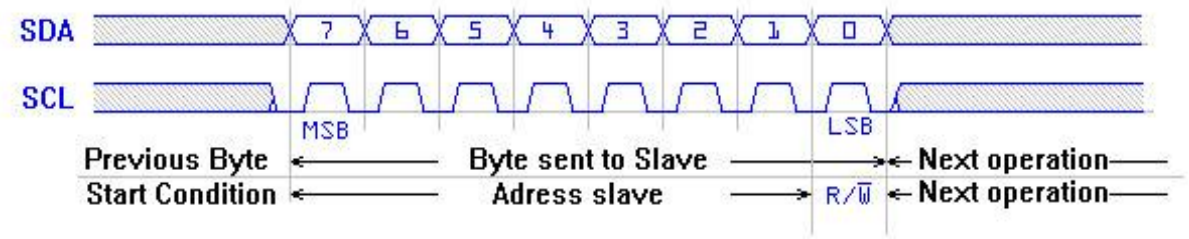


Σχήμα 6.2: Master ενωμένος με τρεις Slaves [47].

Αρχικά το MCU (Multipoint Control Unit) στέλνει στο δίαυλο ένα “Start” condition. Αυτό ενεργά ως ένα σήμα για να δώσουν προσοχή στο δίαυλο όλες οι συσκευές οι οποίες είναι ενωμένες μαζί του. Έτσι όλα τα integrated

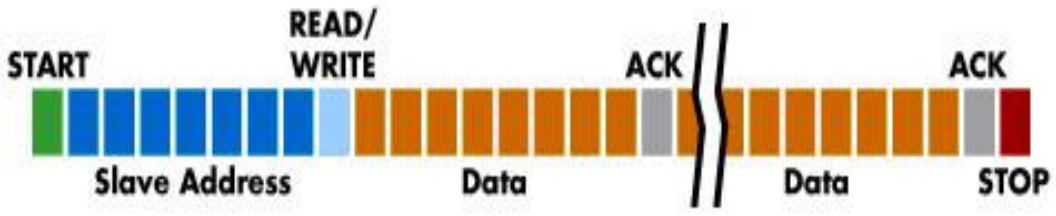
circuits που είναι ενωμένα με το δίαυλο θα ακούσουν για τα δεδομένα που θα σταλούν από τον Master.

Στη συνέχεια το MCU στέλνει τη διεύθυνση της συσκευής με την οποία θέλει να επικοινωνήσει μαζί με μαζί με μια ένδειξη για το αν θέλει να διαβάσει από τη συσκευή ή αν θέλει να γράψει σε αυτήν (1 byte του οποίου τα πρώτα 7 most significant bits υποδηλώνουν τη διεύθυνση και το list significant bit υποδηλώνει αν θα γίνει διάβασμα από τη συσκευή ή γράψιμο σε αυτή). Έχοντας πάρει τη διεύθυνση, όλα τα integrated circuits την συγκρίνουν με τη δική τους διεύθυνση και αν δεν ταιριάζει, περιμένουν το δίαυλο να ελευθερωθεί (όταν ο master στείλει "Stop" condition). Αν η διεύθυνση ταιριάζει τότε το integrated circuit θα στείλει στο Master acknowledge signal.



Σχήμα 6.3: Αναπαράσταση ενός byte που αποστέλλεται σε ένα Slave [51].

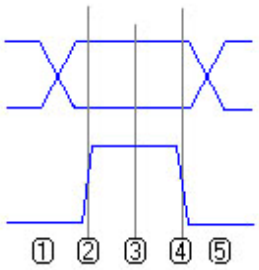
Όταν ο Master λάβει το acknowledgement, μπορεί να στείλει δεδομένα στη συσκευή ή να διαβάσει δεδομένα από αυτή. Με το τέλος αυτής της ενέργειας ο Master θα στείλει στο δίαυλο "Stop" condition. Με αυτό το σήμα υποδηλώνεται ότι ο δίαυλος έχει ελευθερωθεί.



Σχήμα 6.4: Ολοκληρωμένη επικοινωνία μεταξύ Master και Slave [47].

Receiving a byte from a slave

Η σύνταξη του πρωτοκόλλου είναι η ίδια με αυτή της μετάδοσης ενός byte σε ένα slave εκτός από το ότι σε αυτή την περίπτωση ο Master δεν δικαιούται να "αγγίξει" το SDA. Πριν από την αποστολή των 8 clock pulses στο SCL που χρειάζονται για το συγχρονισμό του byte που θα στείλει ο slave, ο Master απελευθερώνει το SDA και ο slave είναι υπεύθυνος για τον έλεγχο αυτής της γραμμής.



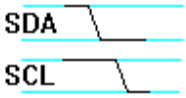
Σχήμα 6.5: Διάβασμα ενός byte που στάλθηκε από το Slave [51].

Το μόνο που έχει να κάνει ο Master είναι να μετατρέπει το SCL σε logical high κατάσταση (2), να διαβάζει το SDA (3) και στη συνέχεια να ρίχνει πάλι το SCL σε logical low κατάσταση(4). Ο slave δεν θα αλλάξει την κατάσταση του SDA κατά τη διάρκεια που το SCL βρίσκεται σε logical high κατάσταση. Κατά τη διάρκεια του (1) και του (5) ο slave μπορεί να αλλάξει την κατάσταση του SDA.

Αυτή η ακολουθία πρέπει να πραγματοποιηθεί 8 φορές για να ολοκληρωθεί η μεταφορά ενός byte το οποίο πάντα μεταδίδεται με τα most significant bits πρώτα.

Start condition

Η συσκευή που εκδίδει το start condition πρώτα ρίχνει το SDA σε logical low κατάσταση και ακολούθως ρίχνει το SCL σε logical low κατάσταση.



Σχήμα 6.6: Γραφική Αναπαράσταση Start Condition [51].

Stop condition

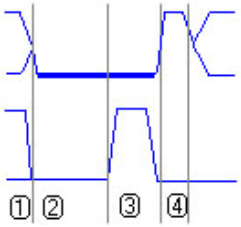
Ο bus Master πρώτα απελευθερώνει το SCL και μετά το SDA (logical high κατάσταση)

Σχήμα 6.7: Γραφική Αναπαράσταση Stop Condition [51].

Ένα stop condition πάντα δηλώνει το τέλος το τέλος της μετάδοσης των δεδομένων ακόμα και αν εκδοθεί στη μέση της μετάδοσης ή στη μέση ενός byte.

Acknowledge του slave στον Master

Ο slave ο οποίος θα στείλει πίσω στο Master ACK ρίχνει το SDA σε logical low κατάσταση αμέσως μετά την παραλαβή του όγδοου bit που μεταδόθηκε από τον Master ή σε περίπτωση που είναι address byte, αμέσως μετά του υπολογισμού της διεύθυνσης που παριστάνει.

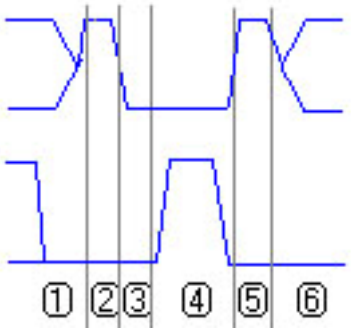


Σχήμα 6.8: Διάβασμα Acknowledge του Slave από το Master [51].

Αυτό σημαίνει ότι μόλις ο Master ρίξει το SCL σε logical low κατάσταση για την ολοκλήρωση της μετάδοσης του bit (1), ο slave θα ρίξει το SDA σε logical low κατάσταση (2). Ο Master στη συνέχεια στέλνει ένα clock pulse στο SCL (3) και ο slave ελευθερώνει το SDA (logical high κατάσταση) με την ολοκλήρωση του clock pulse (4).

Acknowledge του Master στον Slave

Με την παραλαβή ενός byte από ένα slave, ο Master πρέπει να στείλει acknowledge στον Slave.

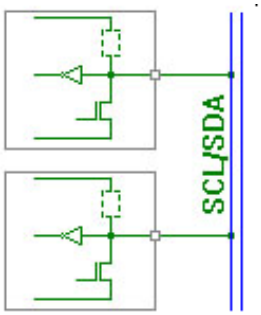


Σχήμα 6.9: Διάβασμα Acknowledge του Master από το Slave [51].

Μετά την μετάδοση του τελευταίου bit στο Master(1), ο slave απελευθερώνει το SDA και αλλάζει σε logical high κατάσταση(2). Στη συνέχεια ο Master θα μεταβάλει το SDA σε logical low κατάσταση(3) και θα στείλει clock pulse στο SCL(4). Μετά την ολοκλήρωση αυτού του clock pulse ο Master θα απελευθερώσει το SDA(5). Έτσι ο slave θα έχει και πάλι τον έλεγχο του SDA.

Ένα acknowledge ενός byte το οποίο λαμβάνει από ένα slave είναι πάντα αναγκαίο, εκτός για το τελευταίο byte της συναλλαγής.

The I2C Bus Hardware Structure



Σχήμα 6.10: απεικόνιση της δομής της κατασκευής του δίαυλου I2C [51].

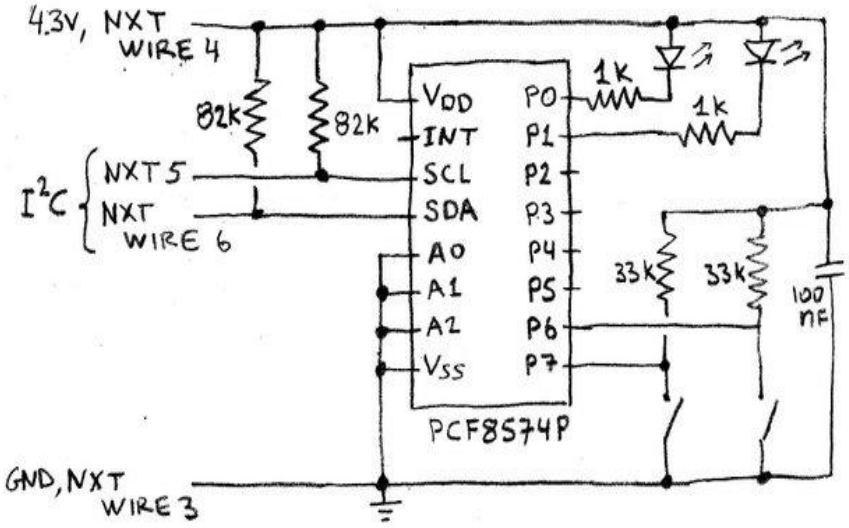
Το interface του δίαυλου είναι κτισμένο γύρω από ένα input buffer και ένα open drain ή ένα open collector transistor.

NXT και I2C

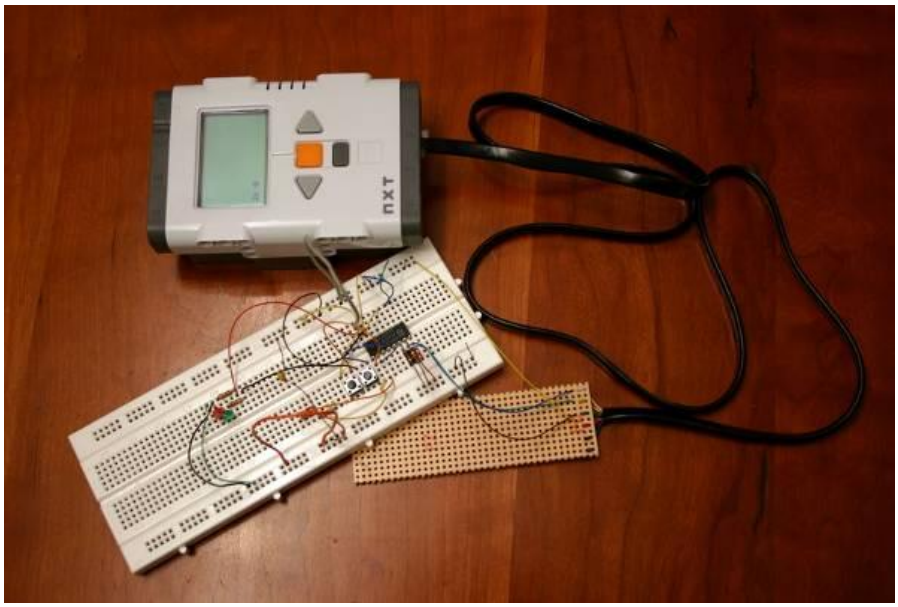
Χρώμα καλωδίου NXT	**Έξοδος καλωδίου**
Μπλε (pin6)	**Serial data (SDA)**
Κίτρινο (pin5)	**Serial Clock (SCL)**
Πράσινο (pin4)	**4.3 v**
Κόκκινο (pin3)	**Γείωση (GRD)**
Μαύρο (pin2)	**Γείωση (GRD)**
Άσπρο (pin1)	**RCX-type 4.9v analog sensor line**

Πίνακας 6.1: Αντιστοιχία χρωματισμού – εξόδου του καλωδίου NXT

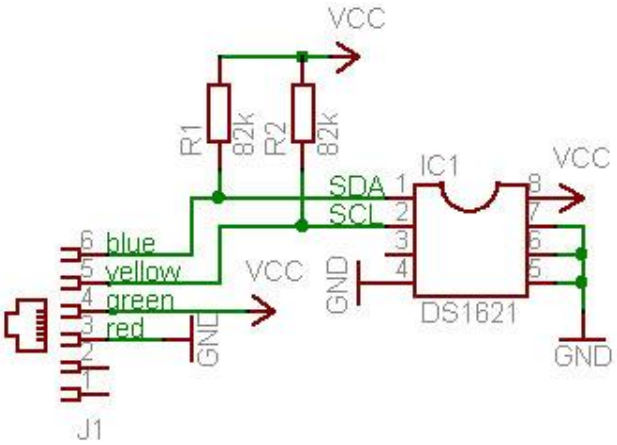
Τα sensor ports πάνω στο NXT υποστηρίζουν το πρωτόκολλο I2C. Σε μια I2C σύνδεση, το NXT θεωρείται πάντα ως η συσκευή Master. Δύο από τα 6 καλώδια ενός sensor port του NXT μεταφέρουν το clock και τα δεδομένα του I2C. Συγκεκριμένα το pin5 (κίτρινο καλώδιο) μεταφέρει το SCL και το pin6 (μπλε καλώδιο) μεταφέρει το SDA. Το pin4 (πράσινο καλώδιο) μεταφέρει ρεύμα (4.3V) και το pin1 (άσπρο καλώδιο) είναι μια RCX-type 4.9v analog sensor line. Τέλος, τα pin2 και 3 είναι γειωμένα.



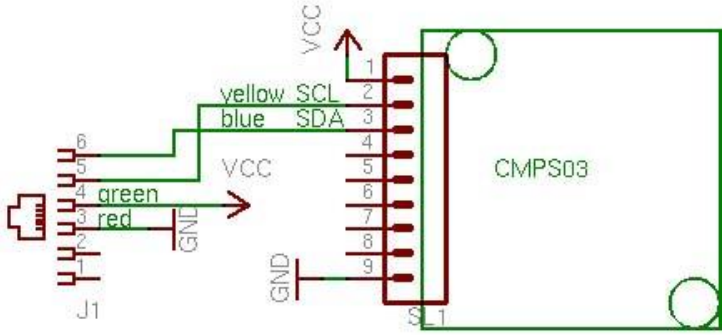
Σχήμα 6.11: Κύκλωμα για την προσθήκη Ψηφιακών I/O στο robot NXT χρησιμοποιώντας το πρωτοκολλο I2C [46].



Σχήμα 6.12: Προσθήκη Ψηφιακών I/O στο robot NXT χρησιμοποιώντας το πρωτοκολλο I2C [46].



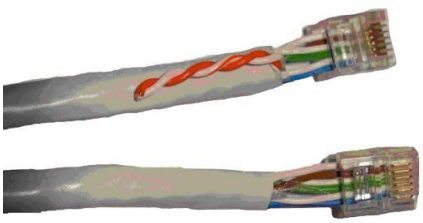
Σχήμα 6.13: NXT ενωμένο με DS1621 Digital Thermometer [35].



Σχήμα 6.14: NXT ενωμένο με CMPS03 magnetic compass [35].

6.3 Αποστολή μηνύματος από το NXT στην οθόνη LCD με την χρήση του πρωτοκόλλου I2C

Για να μπορέσει να επικοινωνήσει το Lego NXT με την οθόνη LCD03 πρέπει να ενωθεί το custom - made καλώδιο που έφτιαξα μαζί της.

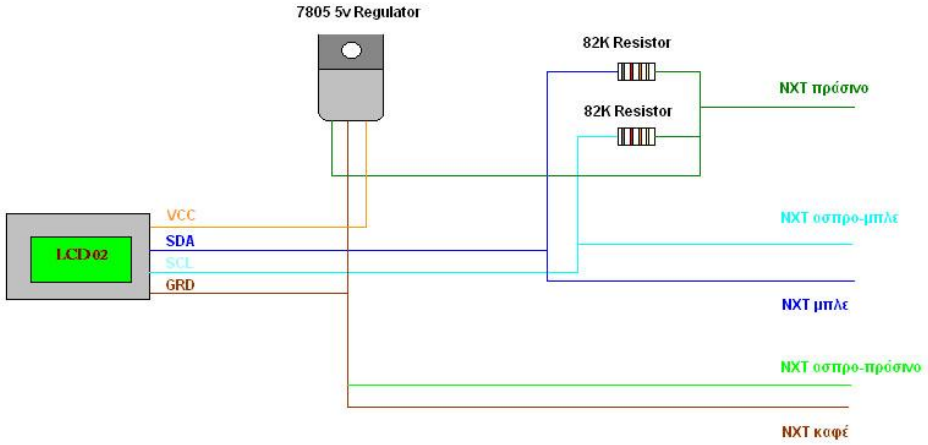


Σχήμα 6.15: Το custom- made καλώδιο που έφτιαξα

Καλώδιο NXT	Custom-made καλώδιο
μπλε	μπλε
κίτρινο	άσπρο-μπλε
πράσινο	πράσινο
κόκκινο	άσπρο-πράσινο
μαύρο	καφέ
άσπρο	άσπρο-καφέ

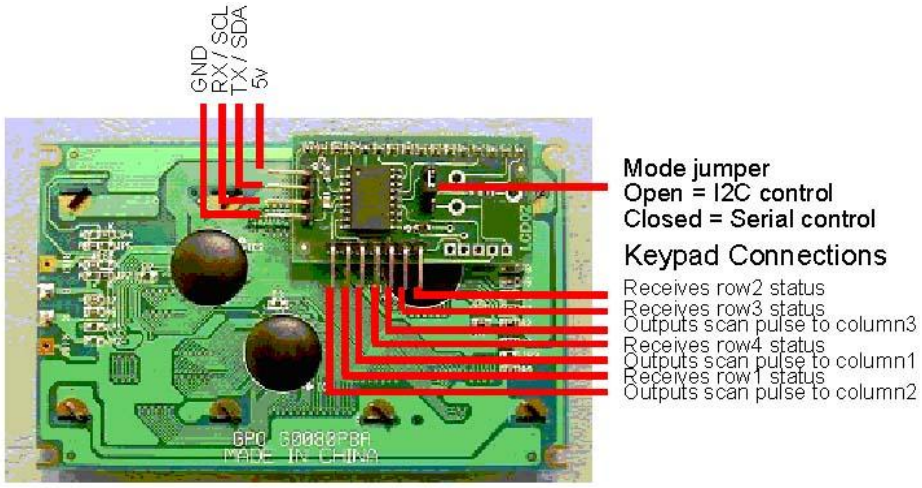
Πίνακας 6.2: Αντιστοιχία χρωματισμού custom made καλωδίου με το καλώδιο NXT

Επειδή η επικοινωνία με την οθόνη θα γίνει με το πρωτόκολλο I2C, αφαιρούμε το jumper για να μπορεί η οθόνη να καταλαβαίνει τις I2C εντολές που θα της στέλλουμε. Ακολούθως ενώνουμε την οθόνη με το Lego NXT όπως φαίνεται στο πιο κάτω σχήμα. Επιπρόσθετα χρειαζόμαστε δύο αντιστάτες 82k και ένα regulator (5v).



Σχήμα 6.16: Σχεδιάγραμμα ένωσης του NXT με την LDC03

Αφού υλοποιήσουμε τις πιο πάνω ενώσεις και βεβαιωθούμε ότι δεν υπάρχει κάποιο βραχυκύκλωμα με τη χρήση ενός αμπερομέτρου, το μόνο που μένει είναι να προγραμματίσουμε το robot με τη γλώσσα προγραμματισμού lejos για να στείλει μηνύματα στην οθόνη.

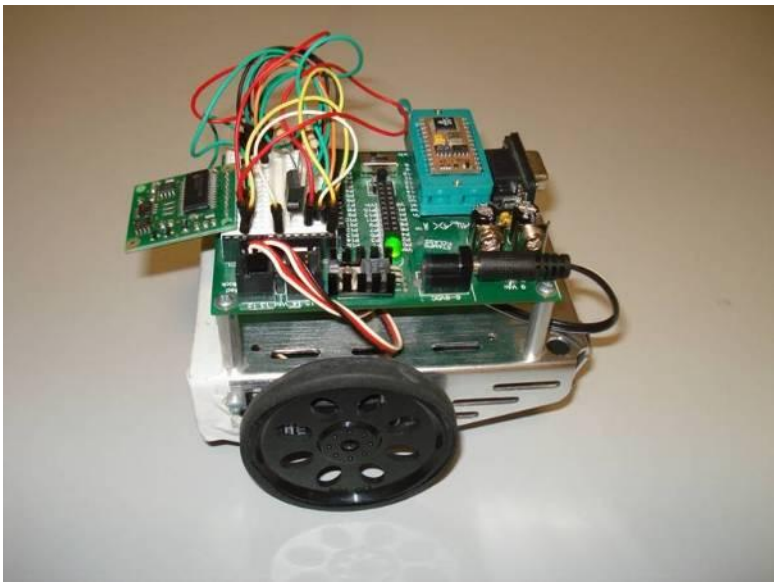


Σχήμα 6.17: LDC03 pin connectors [38].

Η διεύθυνση της οθόνης στο δίαυλο I2C είναι 0xC6 σύμφωνα με το εγχειρίδιο της [38]. Οι I2C διευθύνσεις όμως είναι μόνο μεγέθους 7 bits. Έτσι η διεύθυνση 0xC6 δεν είναι αυτή που πρέπει να χρησιμοποιηθεί, γιατί είναι μεγέθους 8 bits. Η διεύθυνση αυτή πρέπει να γίνει shift αριστερά κατά ένα bit και έτσι η διεύθυνση που θα χρησιμοποιηθεί θα είναι η 0x63. Το πρώτο bit της διεύθυνσης από τα αριστερά χρησιμοποιείται για τον καθορισμό της εντολής, δηλαδή αν η εντολή θα διαβάσει από τον καταχωριστή ή θα γράψει σε αυτόν.

Σχήμα 6.18: Το NXT στέλλει με επιτυχία τη φράση " Hello my master"

6.4 Πλοήγηση BOE-BOT με τη χρήση αισθητήρα micaz χρησιμοποιώντας RS232



Σχήμα 6.19: Ένα από τα δύο boebot που διαθέτει το εργαστήριο ενωμένο με πυξίδα

Για να ελέγξω αν είναι εφικτή η επικοινωνία μεταξύ BOE-BOT και αισθητήρα micaz έγραψα ένα πρόγραμμα στη γλώσσα προγραμματισμού nesC για το λειτουργικό συστήματος αισθητήρων TinyOs1.x, το οποίο στέλλει αριθμούς – χαρακτήρες στο BOE-BOT σειριακά. Συγκεκριμένα, ο κόμβος micaz, στέλλει ανά μικρά χρονικά διαστήματα 1,2,3,4 στο robot και αμέσως μετά από την αποστολή οποιουδήποτε από αυτούς τους αριθμούς στέλνεται και ο χαρακτήρας '\n' (New line). Το robot είναι προγραμματισμένο να ακούει τα μηνύματα που έρχονται σειριακά και να κινείται ανάλογα.

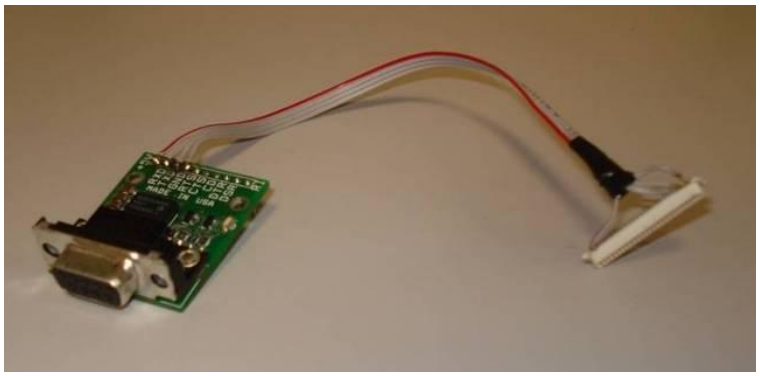
- Λαμβάνοντας τον αριθμό 1 ακολουθούμενο από το χαρακτήρα '\n', το robot κινείται μερικά εκατοστά προς τα εμπρός.
- Λαμβάνοντας τον αριθμό 2 ακολουθούμενο από το χαρακτήρα '\n', το robot κινείται μερικά εκατοστά προς τα πίσω.
- Λαμβάνοντας τον αριθμό 3 ακολουθούμενο από το χαρακτήρα '\n', το robot στρίβει δεξιά.

- Λαμβάνοντας τον αριθμό 3 ακολουθούμενο από το χαρακτήρα '\n', το robot στρίβει αριστερά.

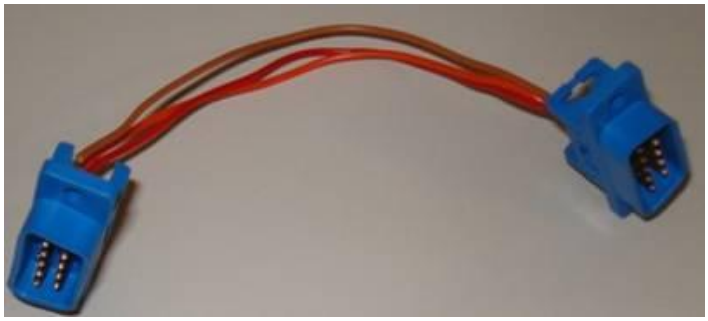
Για την εγκατάσταση της εφαρμογής αυτής πάνω στους ασύρματους κόμβους αισθητήρων, καθώς και όλων των άλλων που θα αναφερθούν παρακάτω έχουν χρησιμοποιηθεί δυο programming boards, το MIB510 και το MIB520.

Σχήμα 6.20: Τα programming boards τα οποία είχα διαθέσιμα για τον προγραμματισμό των κόμβων αισθητήρων

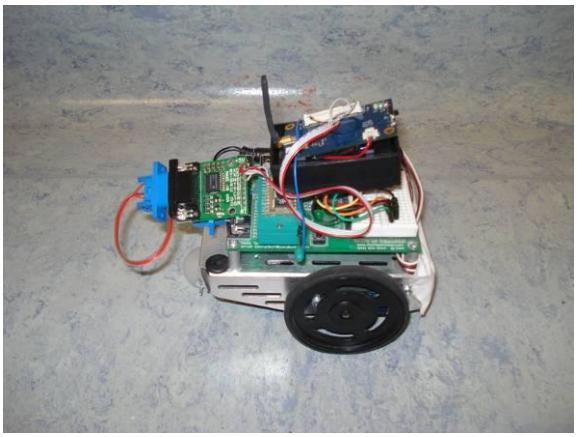
Για την επικοινωνία κόμβων αισθητήρων micaz με το boebot, δημιούργησα δύο custom – made καλώδια για τη σειριακή επικοινωνία μεταξύ τους



Σχήμα 6.22: custom – made καλώδιο για την επικοινωνία του Boebot με τον κόμβο micaz



Σχήμα 6.23: custom – made καλώδιο RS232 male to RS232



Σχήμα 6.24: Το boebot ενωμένο με έναν κόμβο αισθητήρα micaz σειριακά χρησιμοποιώντας τα custom – made καλώδια που έφτιαξα

6.5 Ασύρματη πλοήγηση BOE-BOT με τη χρήση δύο αισθητήρων micaz χρησιμοποιώντας RS232

Επόμενο βήμα είναι επίτευξη της ασύρματης πλοήγησης του boebot. Δηλαδή ο ένας κόμβος micaz στέλνει τις εντολές πλοήγησης στον άλλο κόμβο micaz που είναι ενωμένος σειριακά με το robot. Αυτός με την σειρά του τις μεταβιβάζει στο robot. Συγκεκριμένα, ο πρώτος κόμβος στέλνει τους αριθμούς 1,2,3,4 στο δεύτερο κόμβο και αυτός με τη σειρά του τους μεταβιβάζει στο robot, αφού μετά από κάθε αριθμό στείλει επιπλέον και το χαρακτήρα '\n'. Το robot είναι προγραμματισμένο με τον ίδιο κώδικα, που περιγράφηκε στο προηγούμενο πείραμα και έτσι λαμβάνοντας τις εντολές θα κινείται με τον ίδιο τρόπο όπως αναφέρθηκε πιο πάνω.

6.6 Επικοινωνία Multi-hop μεταξύ αισθητήρων Telosb.

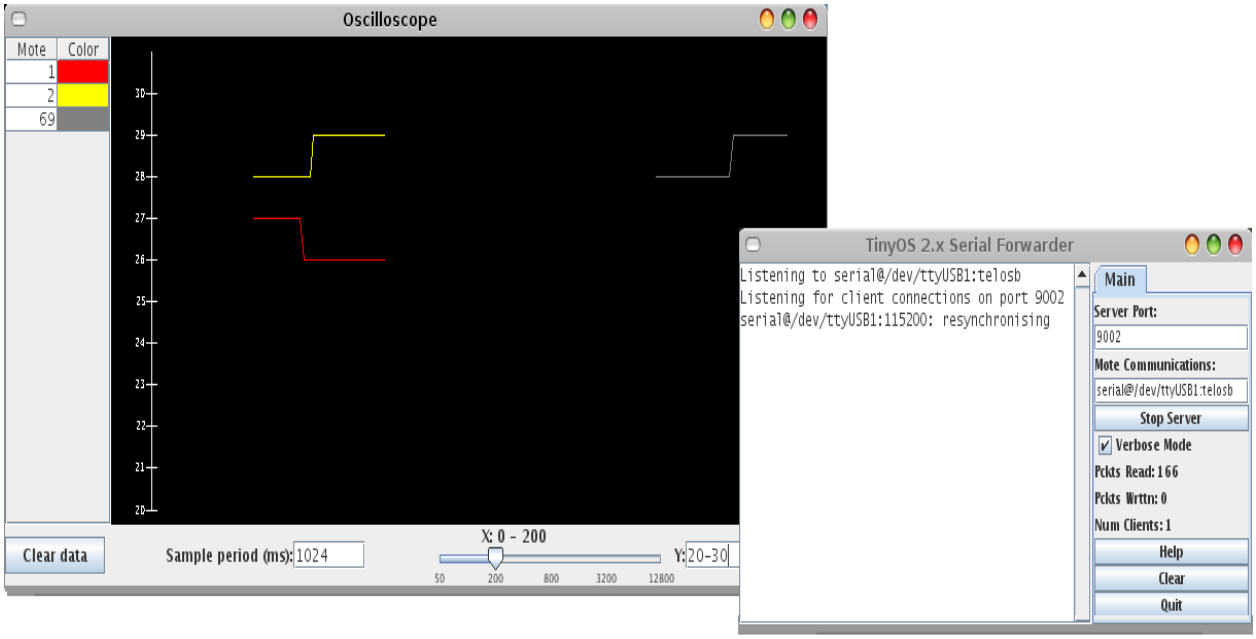
Σε περίπτωση που η εμβέλεια επικοινωνίας ενός κόμβου δεν είναι αρκετή ώστε να υπάρχει άμεση επικοινωνία με το sink, ο κόμβος πρέπει να στείλει τα μηνύματα του στο sink μέσω άλλων κόμβων.

Για την υλοποίηση multi-hop επικοινωνίας χρησιμοποιήθηκαν 3 κόμβοι αισθητήρων Telosb. Οι κόμβοι αισθητήρων ρυθμιστήκαν ώστε να έχουν μειωμένη εμβέλεια μετάδοσης (TX power -25 dBm) και προγραμματίστηκαν να παίρνουν μετρήσεις της θερμοκρασίας της περιοχής ανά τακτά χρονικά διαστήματα. Στους τρείς κόμβους αισθητήρων ανατέθηκαν node id 69,1 και 2. Ο κόμβος αισθητήρα με node id 69 θεωρείται ως το sink, είναι ενωμένος με τον ηλεκτρονικό υπολογιστή σειριακά και του στέλνει τις δίκες του μετρήσεις καθώς και τις μετρήσεις που καταγράφουν οι άλλοι δύο κόμβοι.

Για τη multi-hop επικοινωνία χρησιμοποιήθηκε το πρωτόκολλο CTP [18] το οποίο ήταν υπεύθυνο να δρομολογεί τα πακέτα των κόμβων με node id 1 και 2 στο sink (node id 69). Για να ελέγξω, αν πράγματι οι κόμβοι επικοινωνούν multihop, τοποθέτησα τον κόμβο με node id 2 σε ένα σημείο αρκετά μέτρα μακριά από το sink ούτως ώστε να μην υπάρχει επικοινωνία μεταξύ τους. Στη συνέχεια τοποθέτησα στο κέντρο της μεταξύ τους απόστασης τον κόμβο με node id 1 και παρατήρησα ότι το sink παραλάμβανε πακέτα και από τον κόμβο με node id 2. Αυτό δείχνει ότι οι μετρήσεις του κόμβου με node id 2 δρομολογούνταν στο sink μέσω του κόμβου με node id 1.

Το σενάριο αυτό το μπόρεσα να το παρακολουθήσω και οπτικά με τη βοήθεια ενός εργαλείου το οποίο ονομάζεται serial forwarder. Το εργαλείο αυτό διάβαζε τα πακέτα που έστελνε το sink στον ηλεκτρονικό υπολογιστή και τα έδινε ως είσοδο σε ένα πρόγραμμα γραμμένο σε γλώσσα

προγραμματισμού java που ήταν υπεύθυνο για τη γραφική απεικόνιση των μετρήσεων.



Σχήμα 6.25: Γραφική απεικόνιση των μετρήσεων με τη χρήση του serial forwarder και του προγράμματος γραμμένου σε java

Οι κόμβοι Telosb προγραμματίστηκαν με τη γλώσσα προγραμματισμού nesC για TinyOs 2.x.

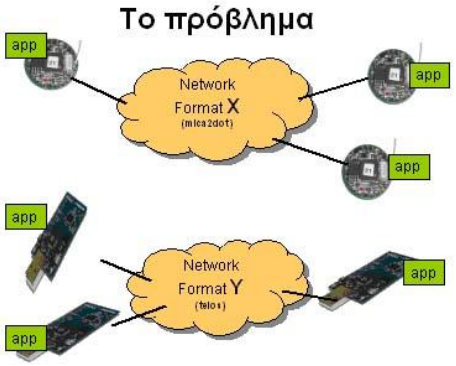
Το σενάριο δεν μπόρεσε να υλοποιηθεί με κόμβους αισθητήρων micaz, γιατί δεν έχουν ακόμα γραφεί drivers για τη χρήση του sensor board mst400 σε TinyOs 2.x. Έτσι οι κόμβοι αισθητήρων micaz που διαθέτει το εργαστήριο δεν μπορούν να χρησιμοποιηθούν για ανίχνευση σε περιβάλλον TinyOs 2.x προς το παρόν.

6.7 Ανάπτυξη ετερογενούς δικτύου Ασύρματων Κόμβων Αισθητήρων

Αφού δεν μπόρεσα να χρησιμοποιήσω τους κόμβους αισθητήρων micaz για ανίχνευση γεγονότων, δοκίμασα να χρησιμοποιήσω τους κόμβους

αυτούς μόνο για τη multi-hop μεταβίβαση των πληροφοριών που ανιχνεύουν οι κόμβοι αισθητήρων Telosb προς το sink. Δηλαδή προσπάθησα να αναπτύξω ένα ετερογενές δίκτυο για να χρησιμοποιήσω όλους τους κόμβους αισθητήρων που έχω στη διάθεσή μου.

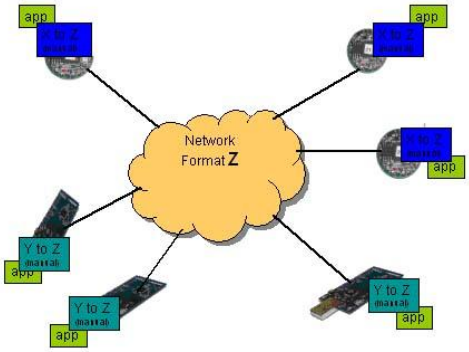
Η ανάπτυξη ενός ομογενούς δικτύου, δηλαδή δικτύου με μόνο ένα είδος κόμβων αισθητήρων είναι σχετικά απλή, αφού οι κόμβοι αισθητήρων μπορούν να ανταλλάζουν μηνύματα μεταξύ τους, αφήνοντας το μεταγλωττιστή να σχεδιάσει τα μηνύματα σε μια αυθαίρετη δομή και να χρησιμοποιήσει τον ίδιο εκτελέσιμο κώδικα σε όλους τους κόμβους. Δυστυχώς, σε ένα ετερογενές περιβάλλον δικτύου, αυτή η προσέγγιση δεν λειτουργεί αφού οι διαφορετικοί επεξεργαστές μπορεί να χρησιμοποιούν διαφορετικές αναπαραστάσεις για τη δομή των μηνυμάτων.



Σχήμα 6.26: Διαφορετικά είδη κόμβων αισθητήρων δεν μπορούν να επικοινωνήσουν μεταξύ τους [34].

Μια προοπτική επίλυσης αυτού του προβλήματος είναι με την παρεμβολή ρουτινών έτσι ώστε να μετατρέπονται τα μηνύματα στις ανάλογες δομές πριν την αποστολή των μηνυμάτων ή αμέσως μετά την παραλαβή των μηνυμάτων [34]. Αυτή η προοπτική όμως έχει μερικά μειονεκτήματα. Πρώτο μειονέκτημα είναι ότι με την ενσωμάτωση τέτοιων ρουτινών, ο κώδικας δεν θα είναι μεταφέρσιμος. Αυτό το μειονέκτημα μπορεί να ξεπεραστεί, αν χρησιμοποιηθούν μόνο πίνακες χαρακτήρων οι οποίοι θα καλούνται μέσω σταθερών offsets, αλλά με αυτό τον τρόπο ο κώδικας θα

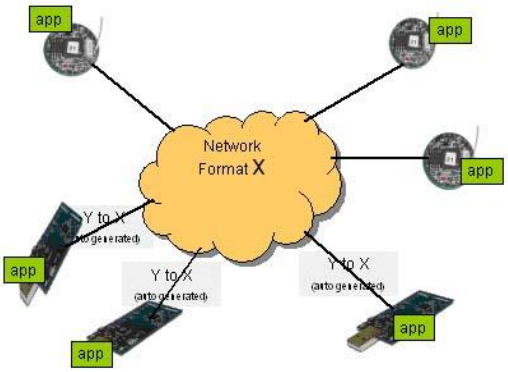
είναι δυσκολοδιάβαστος και μη συντηρήσιμος. Δεύτερο μειονέκτημα είναι ότι η ενσωμάτωση ρουτινών για μετατροπή των πακέτων εκτός από δύσκολη διαδικασία, είναι και μια διαδικασία πολύ επιρρεπής σε λάθη.



Σχήμα 6.27: Τα μηνύματα μετατρέπονται στις ανάλογες δομές πριν την αποστολή ή μετά την παραλαβή [34].

Μια άλλη προοπτική επίλυσης του προβλήματος αυτού, είναι με την χρήση των “Network types” [34]. Τα “Network types ” είναι μια επέκταση που έχει γραφτεί για τη γλώσσα προγραμματισμού nesC για ανάπτυξη δικτύου ανεξαρτήτως πλατφόρμας και δεν αντιμετωπίζει προβλήματα μεταφερσιμότητας του κώδικα, ετερογένεια ή δυσκολίες συντήρησης. Χρησιμοποιώντας τα “Network types”, ο προγραμματιστής ορίζει μια απλή δομή χρησιμοποιώντας μια απλή σύνταξη δήλωσης παρόμοια με αυτή της σύνταξης στη γλώσσα προγραμματισμού C η οποία θα αντιστοιχεί στη δομή του πακέτου του μηνύματος. Ο προγραμματιστής θα μπορεί να έχει πρόσβαση σε αυτήν, όπως σε ένα απλό τύπο γραμμένο σε γλώσσα προγραμματισμού C. Ο μεταγλωττιστής διασφαλίζει ότι αυτός ο τύπος θα έχει την ίδια αναπαράσταση σε όλες τις πλατφόρμες και τα παράγει αυτόματα οποιοδήποτε απαραίτητο κώδικα μετατροπής. Τα “Network types” δεν έχουν κανένα περιορισμό ευθυγράμμισης, οι δομές δεν περιέχουν padding και οι network base types χρησιμοποιούν αναπαράσταση 2's complement και little-endian. Γι’ αυτό το λόγο η αναπαράσταση τους είναι ανεξάρτητη πλατφόρμας και οποιοδήποτε

αυθαίρετο τμήμα της μνήμης μπορεί να προσεγγιστεί μέσω ενός Network type.



Σχήμα 6.28: Χρήση των Network types για την επικοινωνία διαφορετικών ειδών κόμβων [34].

Στο παρόν στάδιο τα Network types υποστηρίζονται μόνο για TinyOs 1.x.

6.8 Επικοινωνία ασύρματων κόμβων αισθητήρων μέσω IEEE 802.15.4 (zigbee)

Για την ασύρματη επικοινωνία μεταξύ κόμβων micaz και κόμβων Telosb σε περιβάλλον TinyOs 2.x δοκίμασα και το πρωτόκολλο IEEE 802.15.4 (zigbee) [36]. Έτρεξα μια απλή εφαρμογή η οποία υλοποιεί τον τρόπο δρομολόγησης μηνυμάτων σε μια τοπολογία cluster tree.

Κάθε συσκευή είναι προγραμματισμένη να στέλνει τα μηνύματα της σε έναν άλλο κόμβο, ο οποίος ονομάζεται PAN coordinator. Τα δυο πρώτα bytes των δεδομένων αυτών των μηνυμάτων χρησιμοποιούνται για τη δρομολόγηση, αφού περιέχουν τη διεύθυνση προορισμού των μηνυμάτων. Ο PAN coordinator, αφού λάβει τα πακέτα αυτά, διαβάζει τα δεδομένα τους και δημιουργεί ένα άλλο πακέτο το οποίο αναλαμβάνει να το στείλει στη διεύθυνση που αναγράφεται στα πρώτα δυο bytes των δεδομένων του

μηνύματος. Επιπλέον, εισάγει στο πακέτο τον αριθμό των πακέτων που έχει δρομολογήσει μέχρι στιγμής.

Για την πειραματική υλοποίηση προγραμμάτισα έναν κόμβο αισθητήρα ως PAN coordinator με node id 1 και δύο άλλους κόμβους ως κανονικούς κόμβους του δικτύου με node id 2 και 3 αντίστοιχα. Ο κόμβος με node id 2 δοκιμάζει να στείλει μηνύματα δεδομένων στον PAN coordinator και ταυτόχρονα ανάβει την πράσινη LED για κάθε πακέτο που στέλνει με επιτυχία. Ο coordinator διαβάζει τα μηνύματα που του στέλνει ο κόμβος με node id 2 και στέλνει τα πακέτα στον κόμβο με node id 3. Ο κόμβος με node id 3 ανάβει την κόκκινη LED κάθε φορά που παραλαμβάνει πακέτο από τον coordinator.

Για να ελέγξω την ορθότητα και την αξιοπιστία της επικοινωνίας του δικτύου, ρύθμισα τον κόμβο με node id 2 να στέλνει αριθμημένα τα μηνύματα του ώστε να μπορώ να παρατηρήσω, αν όλα τα πακέτα φτάνουν με επιτυχία στον προορισμό τους. Στον PAN coordinator και στον κόμβο με node id 3 πρόσθεσα κώδικα έτσι ώστε για κάθε πακέτο που παραλαμβάνουν να στέλνουν στο UART (σειριακά) τον αριθμό του πακέτου που έχουν παραλάβει. Στη συνέχεια ένωσα τον PAN coordinator και τον κόμβο με mode id 3 με ένα ηλεκτρονικό υπολογιστή και χρησιμοποίησα το Hyper Terminal το οποίο εμφάνιζε στην οθόνη τους αριθμούς των πακέτων τα οποία παραλάμβαναν. Μελετώντας τα αποτελέσματα που κατέγραψα, παρατήρησα ότι όλα τα πακέτα μεταφέρονται με επιτυχία στο PAN coordinator. Δυστυχώς όμως ο κόμβος με node id 3 δεν παραλάμβανε με επιτυχία όλα τα πακέτα. Το πιο πάνω σενάριο το έτρεξα αρκετές φορές, χρησιμοποιώντας:

- Τρεις κόμβους micaz.
- Τρεις κόμβους Telosb.
- Δυο κόμβους Telosb και ένα κόμβο micaz ως PAN coordinator.
- Δυο κόμβους Micaz και κόμβο Telosb ως PAN coordinator.

Και στις τέσσερεις περιπτώσεις το ποσοστό απώλειας πακέτων σε σχέση με τον συνολικό αριθμό των πακέτων που στέλνονταν ήταν αρκετά μεγάλο. Έτσι διαπίστωσα ότι η επικοινωνία μέσω του πρωτοκόλλου IEEE 802.15.4 σε TinyOs2.x δεν μπορεί να χρησιμοποιηθεί λόγω της αναξιόπιστης επικοινωνίας που παρέχει ο διαθέσιμος κώδικας προς αξιοποίηση.

6.9 Επικοινωνία Multihop μεταξύ αισθητήρων micaz

Για την αξιοποίηση των αισθητήρων micaz που διαθέτουμε (sensorboard mts400) έγραψα κώδικα nesC για περιβάλλον tinyOs1.x (στο περιβάλλον tinyOs2.x δεν έχουν γραφτεί ακόμα drivers για την υποστήριξη αυτού του sensorboard [42]). Χρησιμοποίησα το πρωτόκολλο δρομολόγησης surge [43] στο οποίο ενσωμάτωσα τα drivers για επικοινωνία με το sensorboard του κόμβου αισθητήρα micaz.

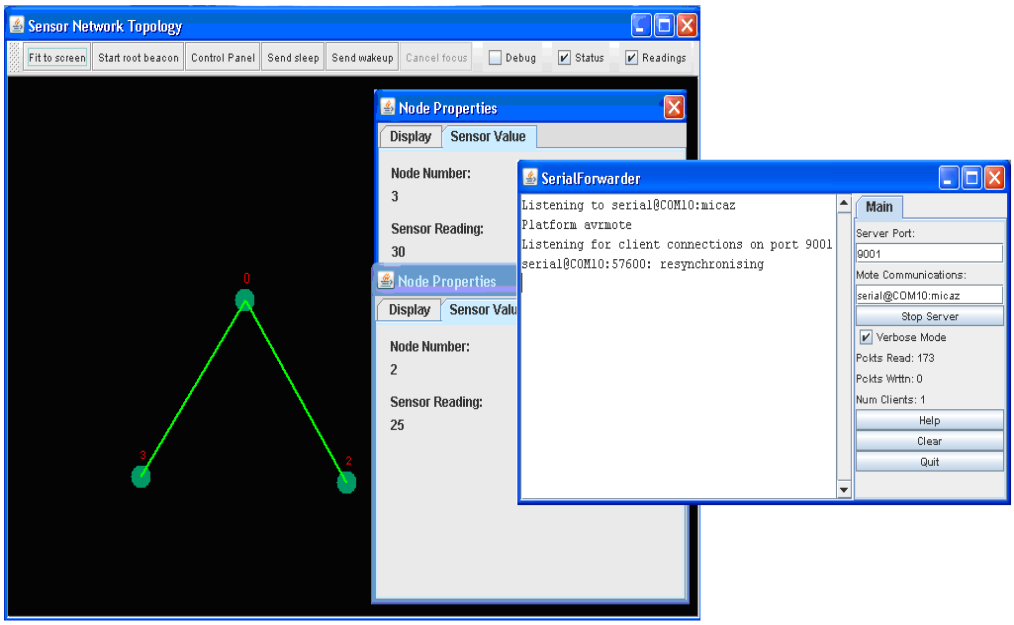
Σχήμα 6.29: sensorboard mts400

Για την πειραματική δοκιμή του κώδικα που έγραψα, χρησιμοποίησα τρεις κόμβους αισθητήρων micaz. Ο ένας κόμβος προγραμματίστηκε ως sink (node id 0) και ενώθηκε με τον ηλεκτρονικό υπολογιστή και οι άλλοι δύο κόμβοι (με node id 2 και 3 αντίστοιχα) προγραμματίστηκαν ώστε να παίρνουν μετρήσεις τις θερμοκρασίας και ανά τακτά χρονικά διαστήματα να δρομολογούν τις μετρήσεις τους στο sink χρησιμοποιώντας τον αλγόριθμο δρομολόγησης surge.

Οι κόμβοι αισθητήρων ρυθμιστήκαν ώστε να έχουν μειωμένη εμβέλεια μετάδοσης (TX power -25 dBm). Για να ελέγξω αν πράγματι οι κόμβοι επικοινωνούν multihop τοποθέτησα τον κόμβο με node id 3 σε ένα σημείο αρκετά μέτρα μακριά από το sink ούτως ώστε να μην υπάρχει επικοινωνία

μεταξύ τους. Στη συνέχεια τοποθέτησα στο κέντρο της μεταξύ τους απόστασης τον κόμβο με node id 2 και παρατήρησα ότι το sink παραλάμβανε πακέτα και από τον κόμβο με node id 3. Αυτό δείχνει ότι οι μετρήσεις του κόμβου με node id 3 δρομολογούνταν στο sink μέσω του κόμβου με node id 2. Το σενάριο αυτό το μπόρεσα να το παρακολουθήσω και οπτικά με τη βοήθεια του εργαλείου serial forwarder. Το εργαλείο αυτό διάβαζε τα πακέτα που έστελνε το sink στον ηλεκτρονικό υπολογιστή και τα έδινε ως είσοδο σε ένα πρόγραμμα γραμμένο σε γλώσσα προγραμματισμού java που ήταν υπεύθυνο για τη γραφική απεικόνιση των μετρήσεων.

Δυστυχώς ο κώδικας του προγράμματος σε java δεν ήταν σωστός και αναγκάστηκα να τον τροποποιήσω ανάλογα για να μπορώ να παρακολουθώ τα πραγματικά αποτελέσματα των μετρήσεων.



Σχήμα 6.30: Γραφική απεικόνιση των μετρήσεων με τη χρήση του serial forwarder και του προγράμματος γραμμένου σε java

Με τον ίδιο τρόπο προγραμμάτισα τους κόμβους αισθητήρων να παίρνουν μετρήσεις του φωτός αντί για θερμοκρασία.

6.10 Υλοποίηση αλγορίθμου ανίχνευσης και ελαχιστοποίησης τρυπών κάλυψης με την προσθήκη επιπρόσθετων κινητών κόμβων αισθητήρων σε πιλοτικό δίκτυο

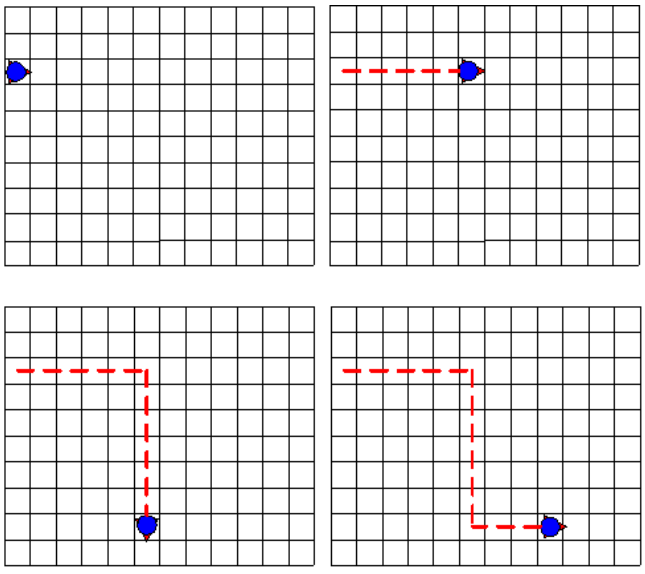
Για την υλοποίηση του αλγορίθμου σε πιλοτικό δίκτυο χρησιμοποιήθηκαν 7 κόμβοι αισθητήρων micaz, από τους οποίους ένας αναπαριστούσε το sink και ήταν ενωμένος με ηλεκτρονικό υπολογιστή. Από τους υπόλοιπους 6 κόμβους, 5 αντιπροσώπευαν τους κόμβους αισθητήρων που είναι διασκορπισμένοι στην περιοχή παρακολούθησης και ο έκτος ενσωματώθηκε πάνω στο robot boebot και αντιπροσώπευε τον επιπρόσθετο κινητό κόμβο. Ο κινητός κόμβος καθοδηγούμενος από το sink ενσωματώνεται στο δίκτυο για να ελαχιστοποιήσει τις τρύπες κάλυψης του δικτύου. Για την αναπαράσταση της περιοχής του δικτύου χρησιμοποιήθηκε μια μεγάλη κόλλα που πάνω στην οποία ζωγραφίστηκε ένα μεγάλο grid 120cm x 180cm. Κάθε τετράγωνο στο grid είχε διαστάσεις 10cm x 10cm.

Για την ελαχιστοποίηση του σφάλματος της κίνησης του boebot, ρύθμισα το boebot ώστε να παρέχει ακρίβεια μέχρι και 2cm (η μέγιστη που μπορεί να προσφέρει). Ο μικροεπεξεργαστής του boebot ήταν ο BS2 (Basic Stamp 2) και χρησιμοποίησα τον BASIC Stamp Editor για τη συγγραφή του προγράμματος μετακίνησής του. Το πρόγραμμα μετακίνησης του boebot δουλεύει με τον ακόλουθο τρόπο:

Το boebot λαμβάνει σειριακά την κατεύθυνση και αμέσως μετά την ποσότητα της μετακίνησης προς αυτή την κατεύθυνση. Συγκεκριμένα αν λάβει ως εντολή κατεύθυνσης τον αριθμό 1 μετακινείται μπροστά, αν λάβει τον αριθμό 2 κινείται προς τα πίσω, αν λάβει τον αριθμό 3 περιστρέφεται επιτόπου δεξιόστροφα και αν λάβει τον αριθμό 4 περιστρέφεται αριστερόστροφα. Το boebot αποθηκεύει τις εντολές μετακίνησης και περιμένει την εντολή εκκίνησης για να αρχίσει να κινείται εκτελώντας με

τη σειρά τις εντολές που του δόθηκαν. Η εντολή εκκίνησης είναι ο αριθμός 5 ακολουθούμενος από τον αριθμό 0.

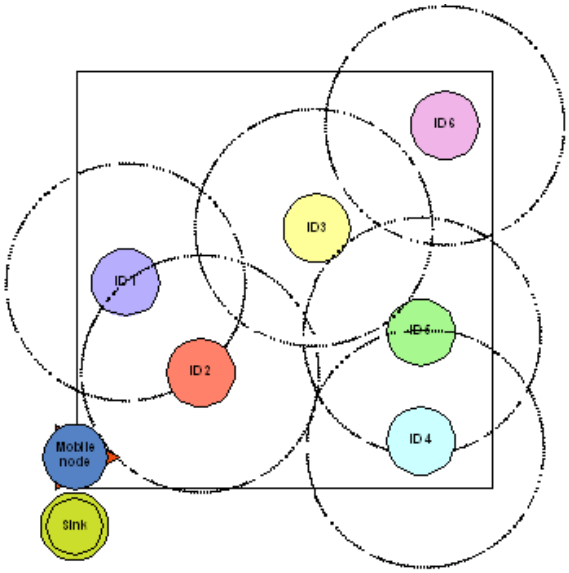
Το boebot δυστυχώς έχει τη δυνατότητα να αποθηκεύσει μέχρι και 10 εντολές, έτσι πάντα μετά από κάθε 9 εντολές πρέπει να του δοθεί η εντολή εκκίνησης, να προχωρήσει το boebot στο χώρο εκτελώντας τις μέχρι στιγμής εντολές που του δόθηκαν και στη συνέχεια αν πρέπει να προχωρήσει περαιτέρω, να του δοθούν οι επόμενες εννιά εντολές μετακίνησης (η δέκατη εντολή πρέπει πάντα να είναι η εντολή εκκίνησης). Για τις κατευθύνσεις μπροστά και πίσω, κάθε μονάδα μετακίνησης μετακινεί το boebot 2cm, επομένως για τη διακριτή μετακίνηση του από τετράγωνο σε τετράγωνο πάνω στο σχεδιασμένο grid της κόλλας πρέπει να παίρνει ως ποσότητα μετακίνησης τον αριθμό 5. Για την δεξιόστροφη και αριστερόστροφη μετακίνηση του robot κατά 90 μοίρες (αριθμός εντολής κατεύθυνσης 3 και 4) πρέπει να παίρνει ως ποσότητα μετακίνησης τον αριθμό 90.



Σχήμα 6.31 : Παράδειγμα πλοήγησης του Boebot

Αν για παράδειγμα το boebot πάρει την πιο κάτω ακολουθία εντολών : 1, 25, 3, 90, 1, 30, 4, 90,1,15, 5, 0 , τότε θα μετακινηθεί πάνω στο grid με τον εξής τρόπο: Το boebot θα κινηθεί 5 τετράγωνα μπροστά, θα περιστραφεί δεξιόστροφα 90 μοίρες, θα κινηθεί 6 τετράγωνα μπροστά θα

περιστραφεί αριστερόστροφα 90 μοίρες, θα κινηθεί 3 τετράγωνα μπροστά και θα σταματήσει.

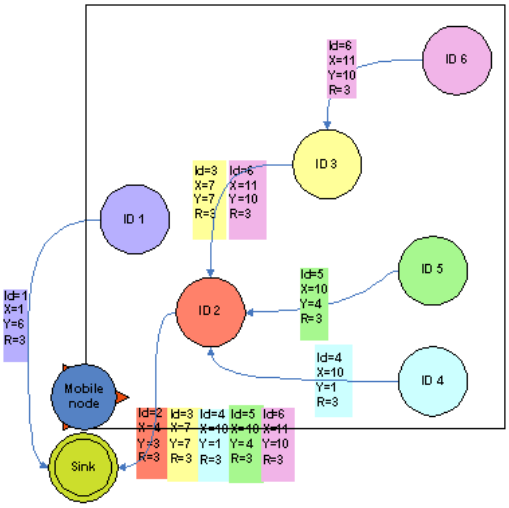


Σχήμα 6.32 : Το δίκτυο πριν εφαρμοστεί ο αλγόριθμος κάλυψης

Οι 5 κόμβοι micaz οι οποίοι θα αποτελούν τους κόμβους αισθητήρων που είναι διασκορπισμένοι στην περιοχή πρέπει με κάποιο τρόπο να γνωρίζουν τις συντεταγμένες τους. Αν η ανάπτυξη του δικτύου είναι σε εξωτερικό χώρο, ο καθορισμός θέσης των κόμβων αισθητήρων μπορεί να επιτευχθεί είτε με κάποια range - based μέθοδο όπως Received Signal Strength Indication (RSSI), Angle of Arrival (AoA), Time of Arrival (ToA) and Time Difference of Arrival (TDoA), είτε έχοντας Global Positioning System receivers σε μερικούς από τους κόμβους ή σε ένα κινητό κόμβο beacon. Αν η ανάπτυξη του δικτύου είναι σε κλειστό χώρο, μπορεί να χρησιμοποιηθεί σύστημα RADAR, το μοντέλο Floor Attenuation Factor propagation ή το location estimation system. Μια άλλη λύση θα ήταν να γίνει ο καθορισμός της τοποθεσίας των κόμβων με τη χρήση κινητών Sink, όπως περιγράφηκε στο κεφάλαιο 3.

Για τους σκοπούς του πειράματος θεωρώ ότι έχει εφαρμοστεί ήδη μια από τις πιο πάνω τεχνικές και ο κάθε κόμβος γνωρίζει ήδη τις συντεταγμένες του σε σχέση με την περιοχή του δικτύου (Grid). Οι κόμβοι που είναι αναπτυγμένοι στο δίκτυο προγραμματίστηκαν να στέλνουν περιοδικά την εμβέλεια αίσθησης

τους και τις συντεταγμένες τους στο sink, χρησιμοποιώντας το multihop πρωτόκολλο δρομολόγησης CTP [18].



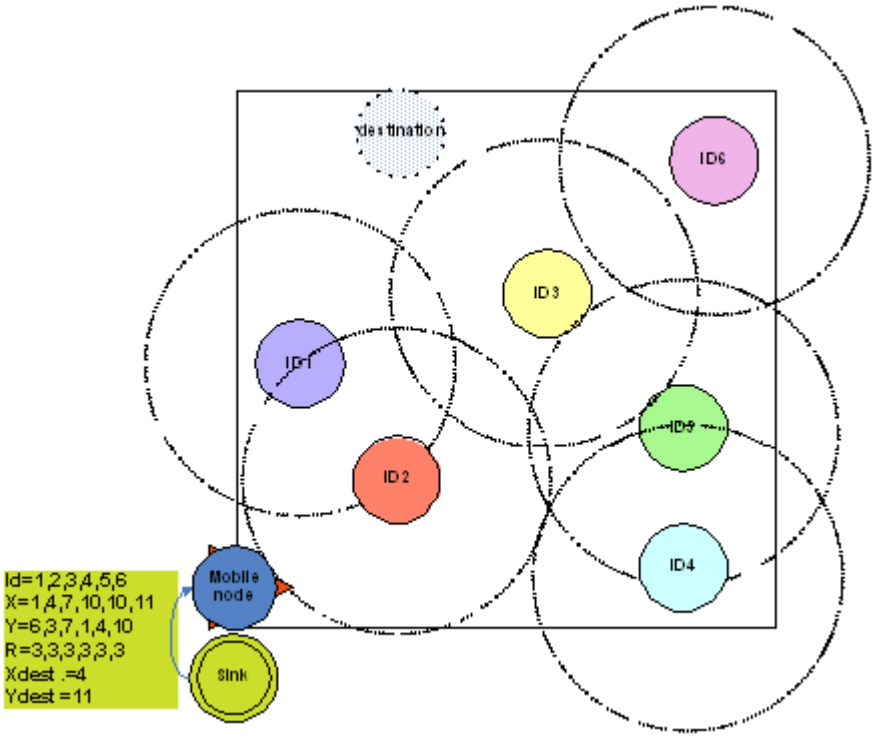
Σχήμα 6.33 : Οι κόμβοι δρομολογούν στο Sink τις προσωπικές τους πληροφορίες με τον αλγόριθμο δρομολόγησης CTP.

Το CTP [18] είναι ένα tree-based collection πρωτόκολλο κατά το οποίο ένας αριθμός κόμβων στο δίκτυο διαφημίζει τον εαυτό του ως ρίζα του δέντρου (στην περίπτωση μας το sink) και οι υπόλοιποι κόμβοι δημιουργούν το δέντρο με ρίζα τους κόμβους αυτούς. Το CTP είναι address-free, δηλαδή ένας κόμβος του δέντρου δεν στέλνει τα δεδομένα του σε μια συγκεκριμένη ρίζα του δέντρου, αλλά διαλέγει ένα από τους κόμβους ρίζα ανάλογα με το επόμενο hop. Οι κόμβοι παράγουν δρομολόγια προς τις ρίζες χρησιμοποιώντας routing gradient. Το πρωτόκολλο CTP υποθέτει ότι το data link layer παρέχει τα πιο κάτω:

- Μια αποδοτική τοπική διεύθυνση για broadcast μετάδοση.
- Συγχρονισμένα acknowledgments για πακέτα unicast.
- Ένα protocol dispatch field για την υποστήριξη πολλαπλών πρωτοκόλλων υψηλού επιπέδου.
- Πεδία για single – hop source και destination.

Ακόμη το CTP υποθέτει ότι έχει αποθηκευμένους υπολογισμούς της ποιότητας σύνδεσης με έναν αριθμό γειτονικών κόμβων για κάθε κόμβο αισθητήρα. Δηλαδή τον αριθμό των μεταδόσεων που χρειάστηκαν για να στείλει στον καθένα από αυτούς ένα unicast πακέτο του οποίου το acknowledgment λήφθηκε με επιτυχία. Το CTP έχει πολλούς μηχανισμούς ώστε να μπορεί να βελτιώσει την αξιοπιστία της μετάδοσης των μηνυμάτων αλλά δεν υπόσχεται 100% αξιόπιστη μεταφορά των δεδομένων. Είναι ένα best effort πρωτόκολλο το οποίο προσπαθεί πολύ σκληρά.

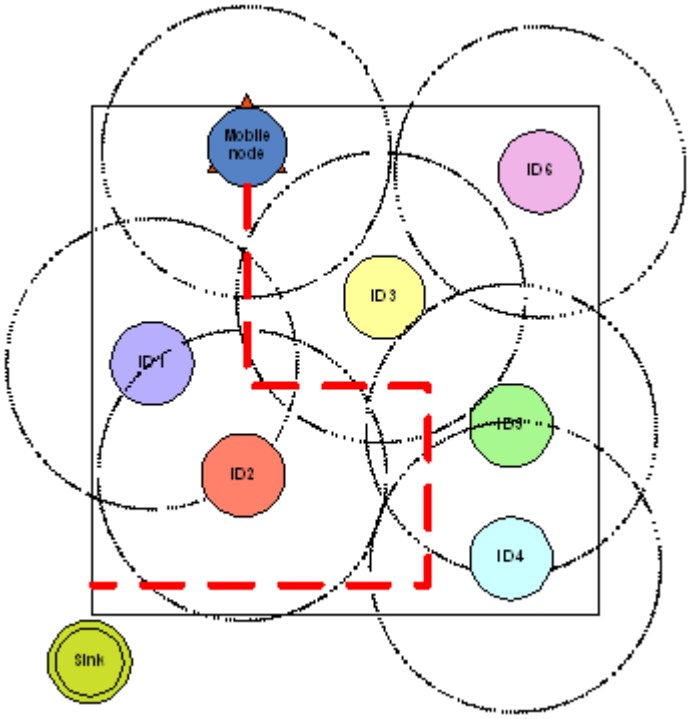
Το sink έχει προγραμματιστεί να σχεδιάσει το χάρτη κάλυψης του δικτύου, μόλις παραλάβει από όλους τους κόμβους τις συντεταγμένες και την εμβέλεια τους. Ο χάρτης στο sink θα αναπαριστάται από ένα δυσδιάστατο πίνακα 18 x 12, ο οποίος θα είναι αρχικοποιημένος με 0 στη κάθε θέση. Κάθε κόμβο του δικτύου, θα τον τοποθετεί στον πίνακα ανάλογα με τις συντεταγμένες του (μετατροπή της θέσης του πίνακα σε τιμή 1) και θα μετατρέπει τις θέσεις του πίνακα γύρω από αυτό σε 1 αν η απόσταση της θέσης αυτής σύμφωνα με τον τύπο της Ευκλείδειας Απόστασης είναι μικρότερη ή ίση με την ακτίνα αίσθησης του κόμβου. Όταν σχεδιαστούν όλοι οι κόμβοι στο δυσδιάστατο πίνακα αρχίζει η διαδικασία για τον εντοπισμό της βέλτιστης θέσης ανάπτυξης επιπρόσθετου. Δηλαδή, υπολογίζονται πόσες θέσεις του grid μπορούν να καλυφθούν από την ακτίνα κάλυψης του κινητού κόμβου που θα προστεθεί και ακολούθως γίνεται έλεγχος σε ολόκληρο τον πίνακα για να βρεθεί η θέση του πίνακα, στην οποία αν τοποθετηθεί ο επιπρόσθετος κόμβος θα μετατραπούν οι περισσότερες γύρω θέσεις από 0 σε 1. Όταν υπολογιστεί η βέλτιστη θέση στέλλεται στον κινητό κόμβο μαζί με τις συντεταγμένες όλων των κόμβων του δικτύου.



Σχήμα 6.34 : Το Sink ενημερώνει τον κινητό κόμβο με τις πληροφορίες του δικτύου και τις συντεταγμένες ανάπτυξής του.

Ο κινητός κόμβος, αφού λάβει το πακέτο από το sink, θα σχεδιάσει το χάρτη δρομολόγησης, αφού τοποθετήσει σε αυτόν τις θέσεις των κόμβων αισθητήρων του δικτύου και προσθέσει γύρω από κάθε κόμβο τη δική του περιοχή κίνδυνου. Ο χάρτης δρομολόγησης αναπαριστάται από ένα δυσδιάστατο πίνακα 18 x 12 ο οποίος είναι αρχικοποιημένος με 1. Κάθε κόμβο του δικτύου, θα τον τοποθετεί στον πίνακα ανάλογα με τις συντεταγμένες του (μετατροπή της θέσης του πίνακα σε τιμή 0) και θα μετατρέπει τις γειτονικές θέσεις του πίνακα γύρω από αυτό σε 0 (τα τετράγωνα- θέσεις που εφάπτονται στη θέση που είναι τοποθετημένος ο κόμβος αισθητήρα). Αφού σχεδιαστεί ο χάρτης δρομολόγησης, υπολογίζεται η διαδρομή που θα ακολουθήσει ο κινητός κόμβος σύμφωνα με τον αλγόριθμο που έχω περιγράψει στο κεφάλαιο 5 (5.5). Ο κινητός κόμβος στέλλει σειριακά μέσω UART, τις εντολές για τη μετακίνηση του boebot. Για κάθε 9 εντολές στέλνεται στο boebot η εντολή εκκίνησης, αφού το boebot δεν μπορεί να αποθηκεύσει περισσότερες από 10 εντολές. Έτσι το boebot θα κινηθεί σύμφωνα με τις 9 εντολές και όταν

φτάσει στο σημείο που του προσδιόρισαν οι 9 αυτές εντολές θα περιμένει για να παραλάβει περαιτέρω οδηγίες.



Σχήμα 6.35 : Ο κινητός κόμβος αναπτύσσεται στις συντεταγμένες που του υπέδειξε το Sink αποφεύγοντας τη σύγκρουση με άλλους κόμβους του δικτύου.

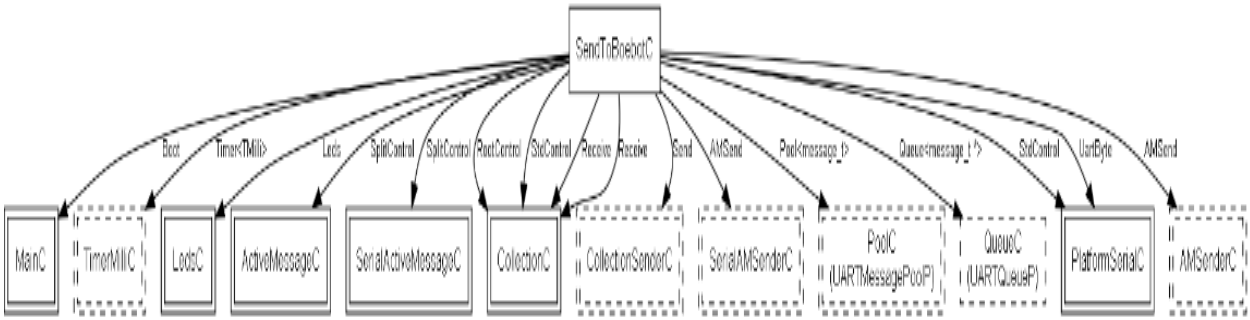
Με αυτό τον τρόπο, ο κινητός κόμβος, θα μετακινηθεί και θα φτάσει στη βέλτιστη θέση που του υπέδειξε το sink. Αφού φτάσει στον προορισμό του, ο κινητός κόμβος ενεργοποιεί τους αισθητήρες του και ενεργεί και επεξεργάζεται δεδομένα, όπως οι υπόλοιποι κόμβοι του δικτύου, δρομολογώντας ανά περιόδους τις συντεταγμένες και το εύρος αίσθησης του στο sink

Components που χρησιμοποιήθηκαν για τη συγγραφή των προγραμμάτων:

Τα προγράμματα γράφτηκαν σε γλώσσα προγραμματισμού nesC για TinyOs 2.x.

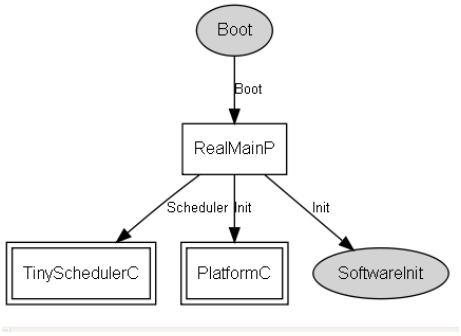
Το πρόγραμμα SendToBoebotC πρέπει να εγκατασταθεί πάνω στους 5 κόμβους του δικτύου (με node id 1-5) καθώς και στο Sink (με node id 0).

Το πρόγραμμα αυτό αποτελείται από τα πιο κάτω components: MainC, TimerMilliC, LedsC, ActiveMessageC, SerialActiveMessageC, CollectionC, CollectionSenderC, SerialAMSenderC, PoolC, (UARTMessagePoolP), QueueC (UARTQueueP), PlatformSerialC, AMSenderC



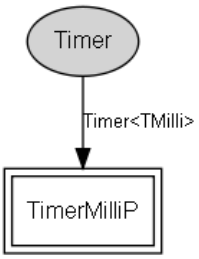
Σχήμα 6.36 : Η καλωδίωση (wiring) του SendToBoebotC.

MainC: Χρησιμοποιείται για την εκκίνηση του προγράμματος.



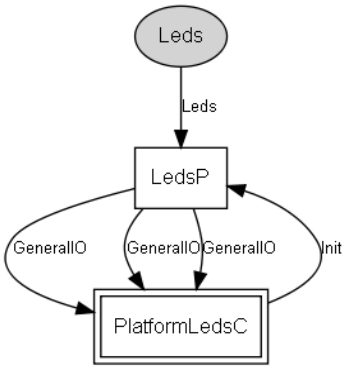
Σχήμα 6.37 : Η καλωδίωση (wiring) του component MainC.

TimerMilliC: Παρέχει timers για την εκτέλεση εντολών σε διάφορες χρονικές περιόδους



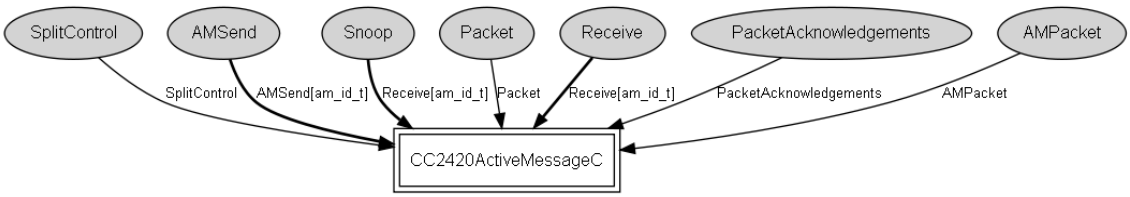
Σχήμα 6.38 : Η καλωδίωση (wiring) του component TimerMilliC.

LedsC: παρέχει interface για τη διαχείριση των LEDS του κόμβου.



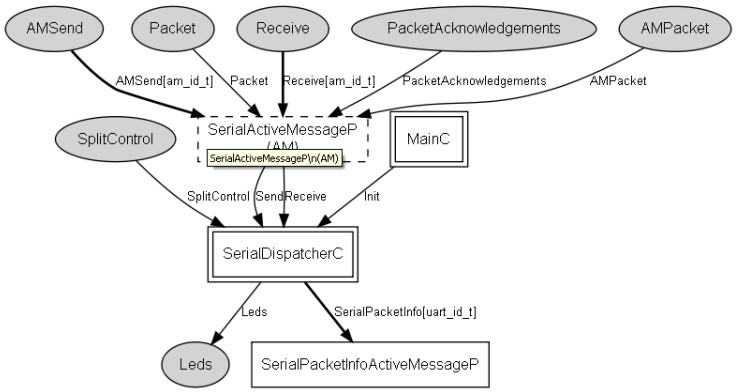
Σχήμα 6.39: Η καλωδίωση (wiring) του component LedsC.

ActiveMessageC: Είναι υπεύθυνο για τη δημιουργία πακέτων τα οποία θα σταλούν RF επικοινωνία.



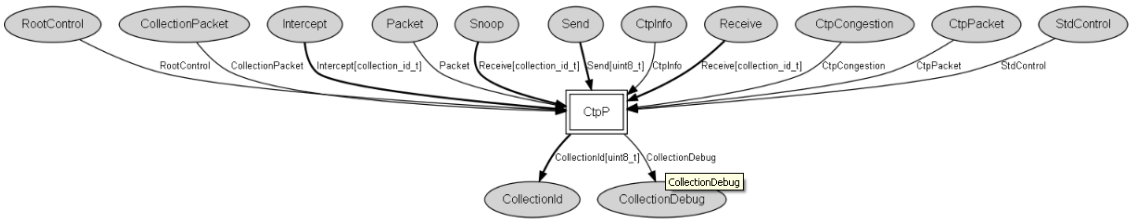
Σχήμα 6.40 : Η καλωδίωση (wiring) του component ActiveMessageC.

SerialActiveMessageC: Είναι υπεύθυνο για τη δημιουργία πακέτων τα οποία θα σταλούν μέσω UART (σειριακά).



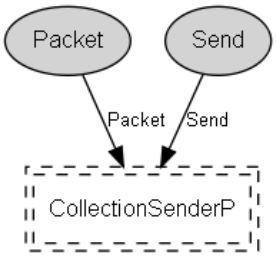
Σχήμα 6.41 : Η καλωδίωση (wiring) του component SerialActiveMessageC.

CollectionC: Είναι υπεύθυνο για τη δρομολόγηση των πακέτων με τη χρήση του CTP πρωτοκόλλου.



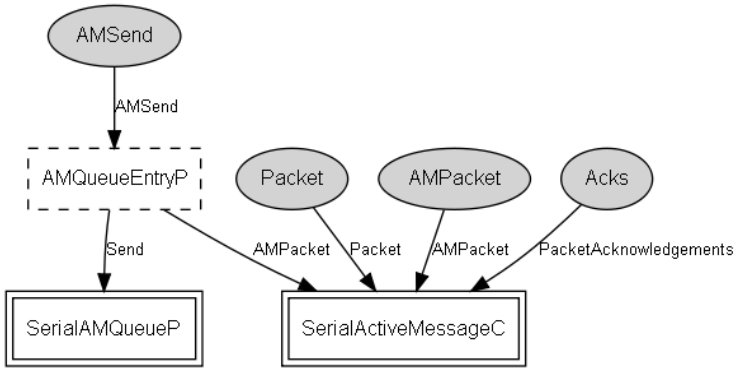
Σχήμα 6.42 : Η καλωδίωση (wiring) του component CollectionC.

CollectionSenderC: Με αυτό το πακέτο μπορούν να δημιουργηθούν πακέτα για την δρομολόγηση με το πρωτόκολλο CTP.



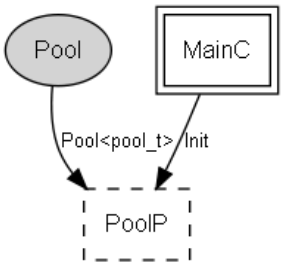
Σχήμα 6.43 : Η καλωδίωση (wiring) του component CollectionSenderC.

SerialAMSenderC: Είναι υπεύθυνο για την αποστολή πακέτων μέσω UART (σειριακά).



Σχήμα 6.44 : Η καλωδίωση (wiring) του component SerialAMSenderC.

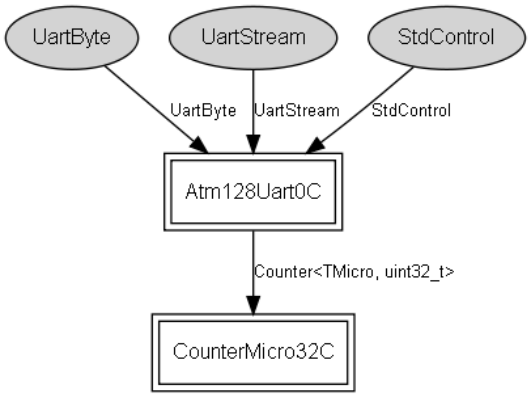
PoolC: Υπεύθυνο για τη διαχείριση των μηνυμάτων τα οποία αποθηκεύονται από το root για επεξεργασία, όταν ο ρυθμός επεξεργασίας των μηνυμάτων είναι πιο αργός από την παραλαβή πακέτων.



Σχήμα 6.45 : Η καλωδίωση (wiring) του component PoolC.

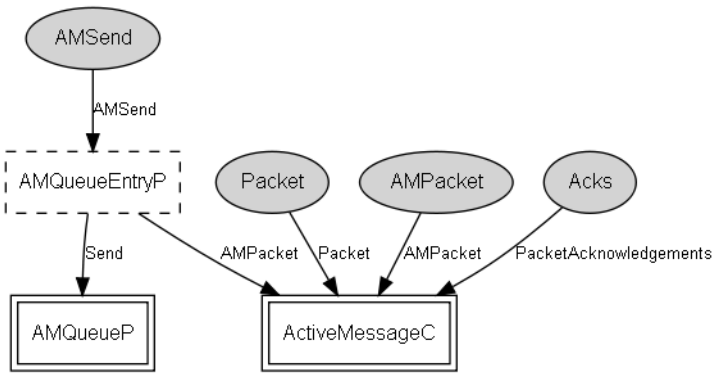
QueueC: Είναι ένα component - ουρά (FIFO) με συγκεκριμένο μέγεθος, για την αποθήκευση των πακέτων που λαμβάνονται από τον κόμβο root.

PlatformSerialC: Είναι υπεύθυνο για τη μεταφορά bytes μέσω UART (σειριακά).



Σχήμα 6.46 : Η καλωδίωση (wiring) του component PlatformSerialC.

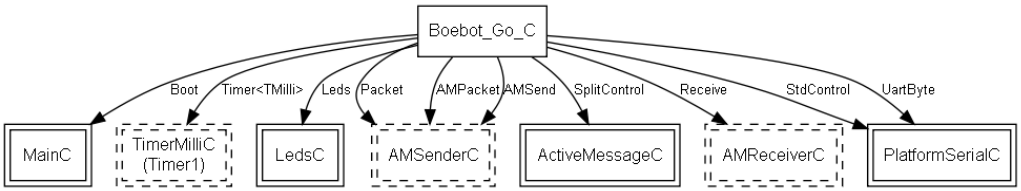
AMSenderC: Είναι υπεύθυνο για την αποστολή πακέτων μέσω RF.



Σχήμα 6.47: Η καλωδίωση (wiring) του component AMSenderC.

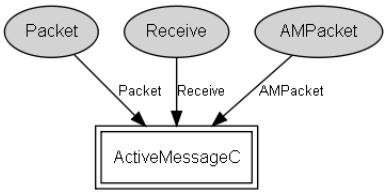
Το πρόγραμμα Boebot_Go_C πρέπει να εγκατασταθεί πάνω στον κόμβο που θα είναι ενσωματωμένος πάνω στο boebot (node id 10) και θα επικοινωνεί σειριακά μαζί του.

Το πρόγραμμα αυτό αποτελείται από τα πιο κάτω components:
MainC, TimerMilliC(Timer1), LedsC, AmSenderC, ActiveMessageC, AMReceiverC, PlatformSerialC.



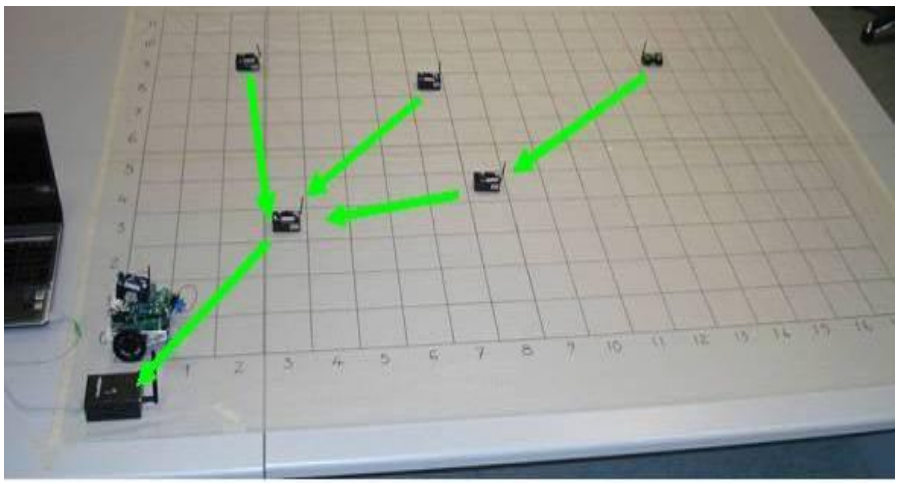
Σχήμα 6.48: Η καλωδίωση (wiring) του Boebot_Go_C.

AMReceiverC: Είναι υπεύθυνο για την παραλαβή πακέτων μέσω RF.

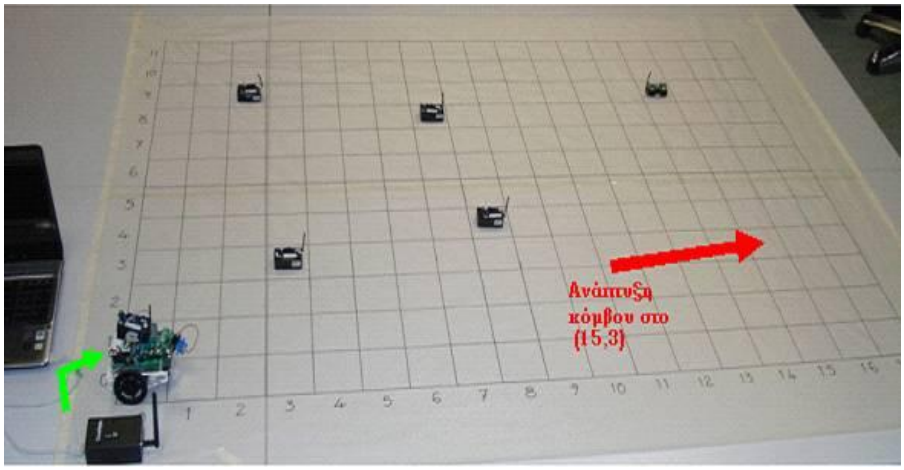


Σχήμα 6.49: Η καλωδίωση (wiring) του component AMReceiverC

6.11 Αξιολόγηση – Συμπεράσματα

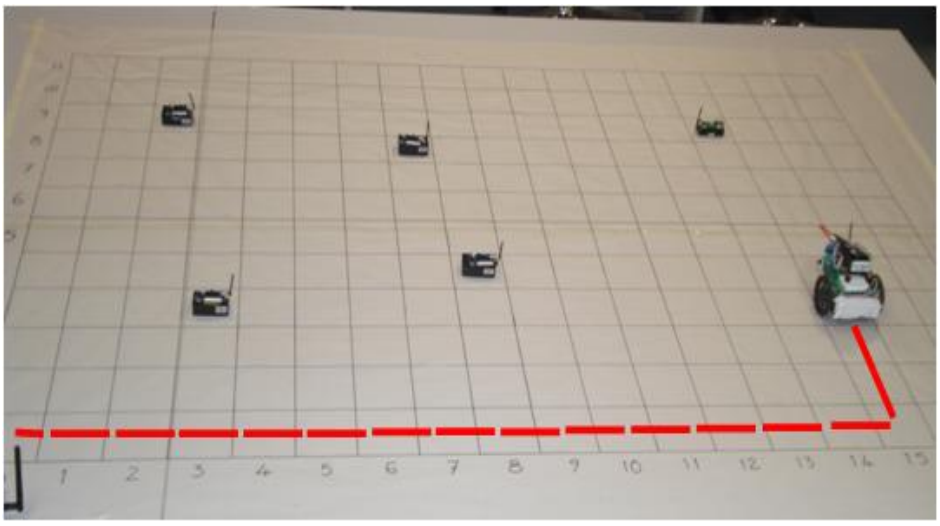


Σχήμα 6.50: Πιλοτικό δίκτυο. Οι ασύρματοι κόμβοι αισθητήρων ενημερώνουν το sink με πληροφορίες που αφορούν την τοποθεσία τους και την ακτίνα αίσθησης τους με τη χρήση του πρωτοκόλλου CTP.



Σχήμα 6.51 : Πιλοτικό δίκτυο. Το Sink υπολογίζει τις συντεταγμένες ανάπτυξης επιπρόσθετου κόμβου και ενημερώνει τον κινητό κόμβο με τις απαραίτητες πληροφορίες για την πλοήγησή του.

Τα αποτελέσματα από το πιλοτικό δίκτυο έδειξαν ότι ο αλγόριθμος δουλεύει με επιτυχία, αφού το Sink κατάφερε και εντόπισε την τρύπα κάλυψης και ενημέρωσε με επιτυχία τον κινητό κόμβο για να μεταβεί στη θέση ανάπτυξης σε όλες τις περιπτώσεις. Ο κινητός κόμβος μετέβηκε με επιτυχία στην περιοχή που του υπέδειξε το Sink μόνο σε περιπτώσεις που η πορεία του ήταν σχετικά απλή.



Σχήμα 6.52 : Πιλοτικό δίκτυο. Ο κινητός κόμβος μεταβαίνει με επιτυχία στον προορισμό του και ελαχιστοποιεί την τρύπα κάλυψης του δικτύου.

Σε περιπτώσεις που ήταν πολύπλοκη η πορεία που έπρεπε να ακολουθήσει, ο κινητός κόμβος δεν κατάφερνε να μεταβεί στη σωστή θέση. Αυτό όμως δεν οφειλόταν σε λάθος του αλγόριθμου, αλλά σε λόγους που αφορούσαν την κατασκευή και το υλικό του Boebot. Εκτός από το γεγονός ότι τα λάστιχα των τροχών του Boebot ήταν φθαρμένα, τα servos που κινούσαν τους τροχούς του δεν γύριζαν με τον ίδιο ρυθμό. Αυτό είχε ως αποτέλεσμα, σε "μεγάλες" αποστάσεις να βγαίνει εκτός πορείας. Ακόμη, όταν το boebot έστριβε, μερικές φορές δεν ολοκλήρωνε με ακρίβεια τη στροφή 90 μοιρών. Αυτό οφειλόταν στην έλλειψη τριβής κατά την κίνηση του, λόγω των φθαρμένων του λάστιχων και στο ότι η απόδοση των servos μειωνόταν κατακόρυφα κατά την μείωση της ενέργειας των μπαταριών του BoeBot.

Τα προβλήματα αυτά μπορούν να αντιμετωπιστούν με την ενσωμάτωση πάνω στο boebot διαφόρων αισθητήρων, οι οποίοι να μπορούν να ελέγχουν την πορεία του και να το καθοδηγούν με τέτοιο τρόπο ώστε να διορθώνει την πορεία του. Για παράδειγμα, με την ενσωμάτωση πυξίδας μπορεί να ελεγχτεί και να διορθωθεί η κατεύθυνση του boebot και με την ενσωμάτωση acceleration sensor μπορεί να υπολογίσει και να διατηρήσει σταθερή την ταχύτητα του. Απαραίτητη είναι και η αντικατάσταση των φθαρμένων του λάστιχων με καινούργια για την ελάττωση των σφαλμάτων της κίνησής του.

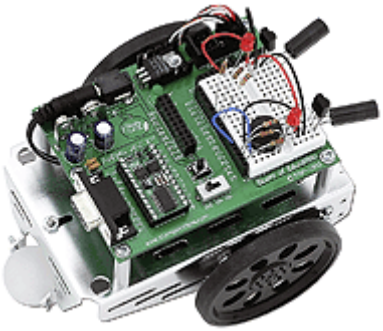
Κεφαλαιο 7

Κινητοί Κομβόι στα Ασύρματα Δίκτυα Αισθητήρων

7.1 Γενικά

Για να ερευνηθεί η κινητικότητα στα Ασύρματα Δίκτυα Αισθητήρων χρησιμοποιούνται συνήθως κάποιες πλατφόρμες – robots. Πάνω στις πλατφόρμες αυτές ενσωματώνονται τα sinks ή και οι κόμβοι αισθητήρων που πρέπει να είναι κινητοί. Αυτό το κεφάλαιο επικεντρώνεται στην περιγραφή των χαρακτηριστικών για τις πιο γνωστές από αυτές τις πλατφόρμες.

7.2 BOE-BOT



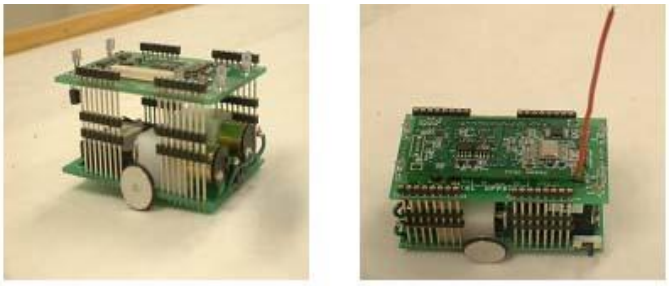
Σχήμα 7.1: Η πλατφόρμα Robomote [37].

Το Boe-bot της parallax [37] αποτελείται από ένα ψηλής ποιότητας αλουμινένιο chassis, το οποίο παρέχει μια εύρωστη πλατφόρμα για τις σέρβο μηχανές και τον τυπωμένο πίνακα κυκλωμάτων. Οι τρύπες και οι αυλακώσεις του, μπορούν να χρησιμοποιηθούν για να προσθέσουν διαφόρων ειδών ρομποτικό εξοπλισμό. Η οπίσθια ρόδα του Boe-bot είναι μια σφαίρα πολυαιθυλενίου τρυπημένη με τρυπάνι που κρατιέται σταθερή με μια σφηνωμένη καρφίτσα. Οι τροχοί του είναι επεξεργασμένοι με τέτοιο τρόπο, ώστε να εφαρμόζουν με ακρίβεια στα σερβο και κρατιούνται με μια μικρή βίδα.

Το πράσινο κύριο κύκλωμα, που είναι τοποθετημένο στο πάνω μέρος του robot ονομάζεται Board of Education. Ο μικροεπεξεργαστής που μπορεί να συνδεθεί σε μια υποδοχή στον πράσινο πίνακα κυκλωμάτων ονομάζεται BASIC Stamp και μπορεί να προγραμματιστεί στη γλώσσα προγραμματισμού PBASIC. Ακόμη, όλες οι εισόδοι – εξόδοι για την επικοινωνία με άλλα εξαρτήματα μπορούν να υλοποιηθούν πάνω στο breadboard που παρέχει.

Το ρομπότ μπορεί να προγραμματιστεί για να ακολουθήσει μια γραμμή, να ακολουθήσει το φως, ή να περιπλανηθεί αυτόνομα αποφεύγοντας τα αντικείμενα τα οποία μπορεί να συναντήσει στο δρόμο του.

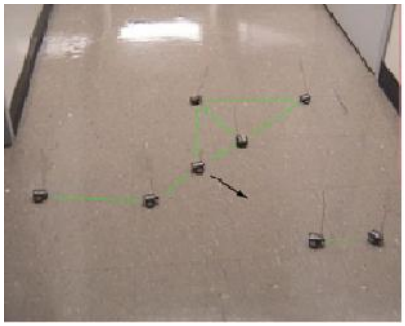
7.3 ROBOMOTE



Σχήμα 7.2: Η πλατφόρμα Robomote [17].

Το robomote έχει σχεδιαστεί ως μια πλατφόρμα για πειραματισμό στα ασύρματα δίκτυα αισθητήρων. Οι σχεδιαστικοί στόχοι αυτής της πλατφόρμας ήταν η ευκολία ανάπτυξής της και το χαμηλό κόστος της. Έχοντας αυτά υπόψη, το robomote έχει σχεδιαστεί ώστε να είναι συμβατό με τα motes του Berkeley. Αποτελείται από ένα μικροεπεξεργαστή Atmel8535 ο οποίος είναι ένας 8-bit AVR RISC MCU με 8k bytes προγραμματιζόμενη Flash , 512 bytes EEPROM και 512 bytes εσωτερική SRAM. Επίσης το robomote έχει δυο motors, μια πυξίδα για να μπορεί να υπολογίζει την πορεία του και αισθητήρες IR. Ο κόμβος αισθητήρα χρησιμοποιείται ως master και επικοινωνεί με το robomote για να το κατευθύνει.

Ένα από τα γνωστά πειράματα που έχουν γίνει με αυτή την πλατφόρμα και παρουσιάζεται στο [17] είναι καταγραφή γεγονότων από το περιβάλλον με κίνηση εμπνευσμένη από τον τρόπο που κινούνται τα βακτηρίδια (biased random walk).

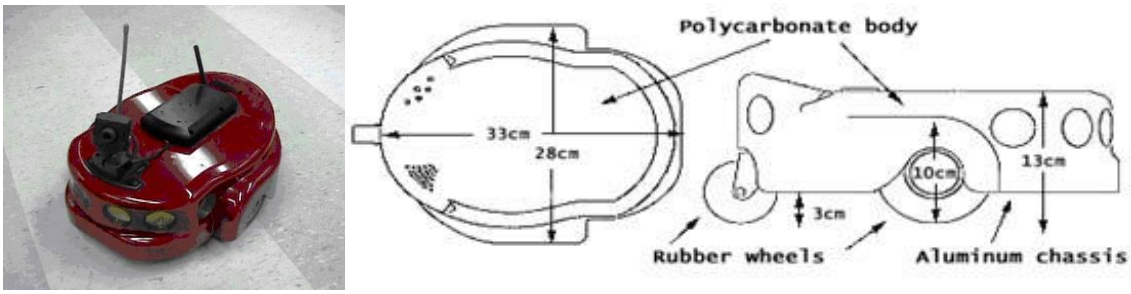


Σχήμα 7.3: Πλήθος από Robomote εν δράσει [17]

7.4 MobileRobots: Pioneer P3-DX and AmigoBot

Σχήμα 7.4: Η πλατφόρμα Pioneer P3-DX [6]

Το Pioneer 3DX (P3DX) από τη Mobile Robots είναι ένα robot το οποίο μπορεί να διαθέτει ένα on-board pc με ένα μεγάλο εύρος από αισθητήρες και μπορεί να επικοινωνεί μέσω WiFi (Wireless Ethernet).

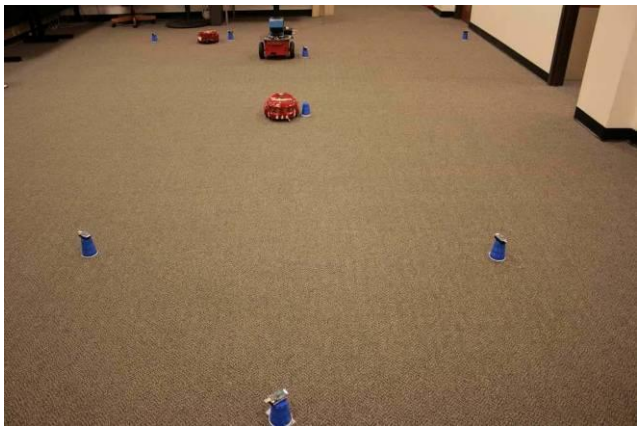


Σχήμα 7.5: Η πλατφόρμα AmigoBot [6]

Το AmigoBot είναι ένα έξυπνο robot με μια παντοδύναμη μηχανή κίνησης. Έχει ενσωματωμένο ένα μικροεπεξεργαστή με το λειτουργικό σύστημα AmigOS και αισθητήρες οι οποίοι του επιτρέπουν να αισθάνεται τι βρίσκεται γύρω από αυτό ώστε να κινείται με ασφάλεια στο περιβάλλον γύρω του αποφεύγοντας εμπόδια. Σε συνδυασμό με ένα state-of-the-art λογισμική ρομποτικής, όπως το *Activ*Media Robotics' Interface for Applications (ARIA) και το λογισμικό Saphira να τρέχει στον υπολογιστή,

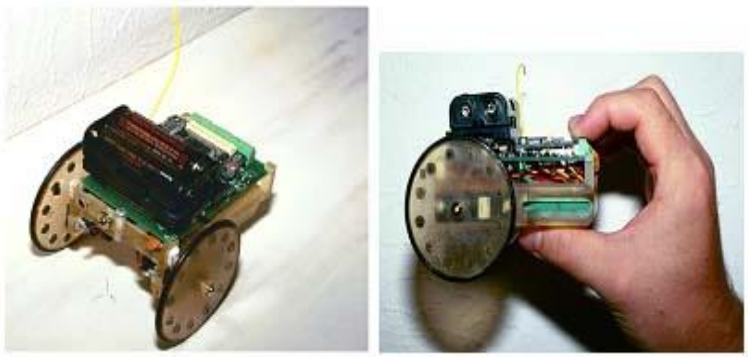
το AmigoBot μπορεί να καθορίσει την τοποθεσία του και να βρεί δρομολόγιο από μια θέση σε μια άλλη.

Τα δύο αυτά robot στο [6] χρησιμοποιήθηκαν σε πειράματα που αφορούν επιδιόρθωση συνδεσιμότητας σε ασύρματα δίκτυα αισθητήρων με k-redundancy. Τα robot αυτά κατευθύνονται από ασύρματους κόμβους αισθητήρων για την επιδιόρθωση της συνδεσιμότητας του δικτύου.



Σχήμα 7.6: Συνεργασία AmigoBot και Pioneer P3-DX για επιδιόρθωση συνδεσιμότητας [6]

7.5 MICAbots



Σχήμα 7.7: Η πλατφόρμα MICAbot [30]

Τα MICAbots έχουν σχεδιαστεί για εργαστηριακά πειράματα και αποτελούνται από δύο motors, ένα μικροεπεξεργαστή, μια βάση πάνω στην οποία τοποθετούνται οι μπαταρίες και ο κόμβος αισθητήρα. Τα robot

αυτά έχουν μικρό μέγεθος (*8.6* cm x 6.1 cm x 2.1 cm το μέγεθος της βάσης τους) και χαμηλό κόστος (λιγότερα από $350) γιατί κατασκευάζονται με υλικά που είναι διαθέσιμα στην αγορά. Χρησιμοποιούν τις πλατφόρμες MICA του Berkeley για την επεξεργασία των δεδομένων και την επικοινωνία και προγραμματίζονται σε περιβάλλον TinyOs. Τα MICAbots έχουν χρησιμοποιηθεί στο [30] σε πειράματα για την υλοποίηση scalable formation αλγορίθμων για μεγάλης κλίμακας συστημάτων με robot του ίδιου τύπου. Άλλες εφαρμογές περιλαμβάνουν ad-hoc δίκτυα μεγάλης κλίμακας και robotic swarms.

Σχήμα 7.8: MICAbots εν δράσει [30].

7.6 CotsBots

Σχήμα 7.9: Η πλατφόρμα CotsBot [8].

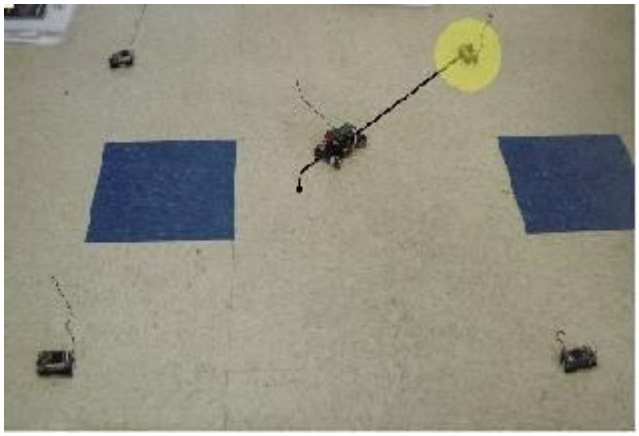
Τα CotsBots [8] είναι modular mobile robots τα οποία είναι κατασκευασμένα με εμπορικά υλικά (Kyosho mini-z και micaZ αισθητήρες). Τα robot αυτά είναι μια κατάλληλη πλατφόρμα πάνω στην οποία μπορούν να ερευνηθούν αλγόριθμοι συνεργασίας και

κατανεμημένης αίσθησης σε μεγάλα ασύρματα δίκτυα αισθητήρων. Έχουν μικρό μέγεθος (13cm x 6.5cm μέγεθος της βάσης) και κοστίζουν λιγότερο από $200. Τα robot αυτά είναι εξοπλισμένα με ενσωματωμένο επεξεργαστή, ασύρματη επικοινωνία και ένα σκελετό αυτοκινήτου (Kyosho mini-z) για να έχει τη δυνατότητα να κινείται. Τα προγράμματα του είναι γραμμένα σε περιβάλλον TinyOs.

Το motor board το οποίο ελέγχει τις κινήσεις του CotsBot αποτελείται από:

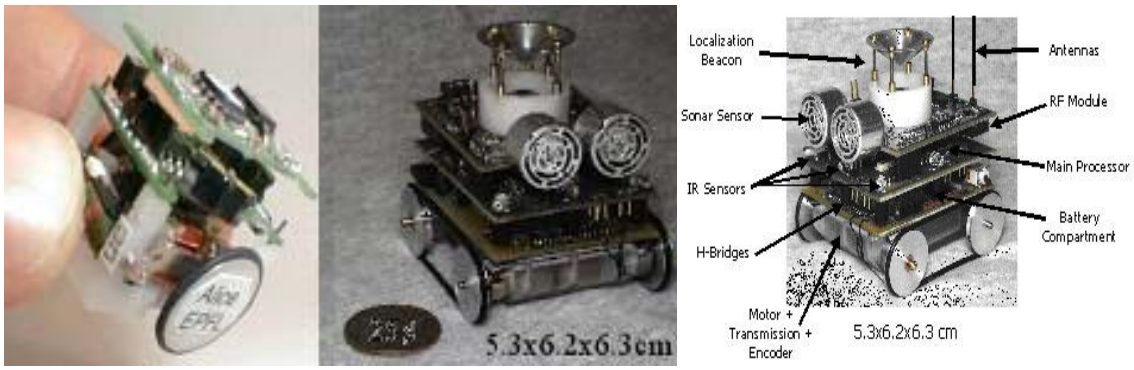
- Atmel ATmega8L
- 1-8MHz CPU
- 8KB Program Memory
- 1KB RAM
- 2 Discrete H-Bridge Circuits
    - Speed and Direction Control of Motors
    - up to 4A, 30V load
- Power Monitoring
- Accelerometer ADXL202e
- 51-pin bus

Ένας αλγόριθμος που μελετήθηκε στο [9] είναι ο simple diffusion algorithm (BeepDiffusion) ο οποίος δεν είναι βέλτιστος αλλά μέσω αυτού έγινε επίδειξη για το τι μπορεί να επιτευχθεί με τη χρήση απλών αισθητήρων και τη βασική λειτουργία των CotsBots. Επίσης έχει αρχίσει έρευνα για την ενσωμάτωση των robot σε ένα ασύρματο δίκτυο αισθητήρων. Σε αυτή την περίπτωση το δίκτυο αισθητήρων θα παρέχει πληροφορίες για τον καθορισμό της θέσης των robot και τις μετρήσεις που καταγράφηκαν από τους κόμβους αισθητήρων.



Σχήμα 7.10: Cotsbot εν δράσει [9].

7.7 Millibots (The Millibot Team)



Σχήμα 7.11: Η πλατφόρμα MilliBot [33].

Τα millibots [33] είναι μικρά ημι-αυτόνομα ή και αυτόνομα robots. Λόγω του μικρού τους μεγέθους (7 X 7 X 7cm), είναι αδύνατο να χωρέσουν σε ένα robot όλοι οι απαραίτητοι αισθητήρες που χρειάζονται για την ολοκλήρωση μιας αποστολής. Ακόμη λόγω του μικρού μεγέθους των αισθητήρων και της μικρής ισχύς του robot, οι αισθητήρες έχουν περιορισμένες δυνατότητες. Γι' αυτό το λόγο, τα robot αυτά δουλεύουν σε ομάδες, στις οποίες το κάθε robot έχει τις δικές του ιδιαιτερότητες και όλα μαζί συνεργάζονται για να φέρουν εις πέρας μια αποστολή. Επομένως η συνεργασία είναι απαραίτητη και πρέπει να υπάρχει συντονισμός στο μάζεμα των πληροφοριών από το περιβάλλον. Η επιλογή των αισθητήρων που θα διαθέτει το κάθε robot και η συγκρότηση ομάδας εξαρτάται από τις

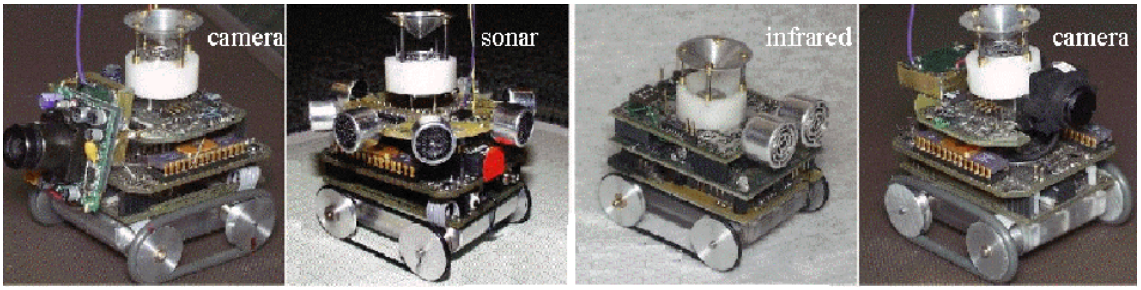
ανάγκες της αποστολής. Π.χ κάποια robots μπορεί να είναι εξοπλισμένα με αισθητήρες για την εξερεύνηση της ομάδας, άλλα να είναι εξοπλισμένα με μεγαλύτερη υπολογιστική ισχύ για την επεξεργασία των δεδομένων αίσθησης και της δημιουργίας του τοπικού χάρτη της περιοχής και άλλα robots μπορεί να είναι εξοπλισμένα με καλύτερα motors και με πιο ανεπτυγμένη κινητικότητα.

Για να επιτευχθεί η ειδίκευση του κάθε robot χωρίς να χρειαστεί μεγάλη ποσότητα από robots, τα millibots έχουν κατασκευαστεί ώστε να είναι διαμορφώσιμα. Ακόμα και η πλατφόρμα κινητικότητας είναι διαμορφώσιμη και μπορεί να επιλεχθεί ανάλογα με την επιφάνεια στην οποία θα αναπτυχθούν.

Ένα σημαντικό συστατικό των millibots είναι το σύστημα καθορισμού τοποθεσίας τους που βασίζεται σε υπέρηχους. Αυτό το σύστημα έχει ένα σημαντικό πλεονέκτημα σε σχέση με τα υπάρχοντα συστήματα γιατί δεν χρειάζεται σταθερά beacons. Χρησιμοποιώντας τα millibots είτε ως beacons, είτε ως localization receivers εναλλακτικά, η ομάδα ως σύνολο μπορεί να κινηθεί και να επανατοποθετηθεί διατηρώντας ακριβείς τους υπολογισμούς καθορισμού τοποθεσίας. Ο ακριβής εντοπισμός της τοποθεσίας των robot είναι πολύ σημαντικός για εργασίες οι οποίες περιλαμβάνουν εξερεύνηση και mapping.

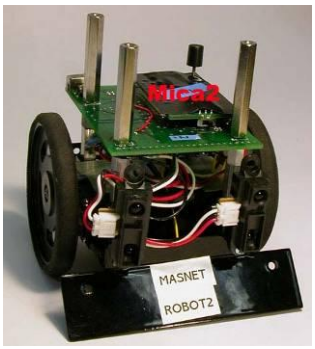
Μερικές εφαρμογές στις οποίες μπορούν να χρησιμοποιηθούν τα millibots περιλαμβάνουν:

- Collaborative localization.
- Collaborative mapping.
- Rescue Support.
- Urban Reconnaissance.



Σχήμα 7.12: MilliBots με διαφορετικές ειδικεύσεις [33].

7.8 MASMOTES



Σχήμα 7.13: Η πλατφόρμα MASMOTE [15].

Τα MAS-motes [15] είναι κατασκευασμένα βασισμένα στους αισθητήρες micaZ της Crossbow Technology Inc. Τα robot αυτά είναι φθηνά και δεν έχουν ψηλό κόστος συντήρησης. Είναι γερά και μικρά έτσι ώστε να αντέχουν σε αντίξοες συνθήκες. Οι δυο τροχοί που διαθέτουν έχουν ξεχωριστό motor. Επίσης είναι εξοπλισμένα με τρεις αισθητήρες IR (δυο μπροστά και ένας πίσω) για αποφυγή των συγκρούσεων καθώς και 2 φωτεινούς αντιστάτες οι οποίοι είναι γυρισμένοι προς τα κάτω για να ανιχνεύουν τη συγκέντρωση ομίχλης κάτω από τα MAS-mote. Τα robot αυτά χρησιμοποιήθηκαν σε πείραμα για vision – based καθορισμό τοποθεσίας.

Σχήμα 7.14: MASMOTES εν δράσει [15].

7.9 E-PUCK

Σχήμα 7.15: Η πλατφόρμα E-PUCK[32].

Το E-PUCK είναι ένα mini κινητό robot το οποίο δημιουργήθηκε από το Swiss Federal Institute of Technology στη Lausanne για εκπαιδευτικούς σκοπούς. Τώρα είναι εμπορικά διαθέσιμο από την K-Team.

Το E-PUCK έχει ένα επεξεργαστή dsPIC στα 60MHz (15MIPS) με πυρήνα DSP για επεξεργασία σημάτων, 8 proximity sensors και φωτός (ambient), ένα 3D accelerometer, 3 μικρόφωνα omni-directonal, μια VGA κάμερα, και ένα δέκτη IR για ασύρματο έλεγχο. Επίσης έχει 8 κόκκινα LEDS, ένα πράσινο body light, ένα κόκκινο LED μεγαλύτερης έντασης στο μπροστινό μέρος και ένα μεγάφωνο το οποίο μπορεί να αναπαράγει αρχεία WAV. Παρέχει επικοινωνία μέσω RS232 και Bluetooth. Ενέργεια του παρέχουν 5 WH LiIon μπαταρίες. Προγραμματίζεται σε γλώσσα C (μεταγλωττιστής GNU GCC).

Για να μπορέσει το E-PACK να επικοινωνεί ασύρματα έχει δημιουργηθεί ένα radio communication module έτσι ώστε να μπορούν να διεξαχθούν πειράματα τα οποία απαιτούν ασύρματη επικοινωνία.

Τα E-PACK έχουν χρησιμοποιηθεί σε πειράματα για Collective Decision και Isolated Collective Decision στο [32].

Σχήμα 7.16: Πλήθος από E-PUCK εν δράσει [32].

Κεφαλαιο 8

Συμπεράσματα

8.1 Γενική σύνοψη

Η διπλωματική ξεκίνησε αναφέροντας τις εφαρμογές και τα σχέδια αλληλεπίδρασης των ασύρματων δικτύων αισθητήρων, τους σημαντικούς τους μηχανισμούς και τα layers που τα αποτελούν. Αναλύθηκαν τα κύρια χαρακτηριστικά με τα οποία αποτελείται ένας κόμβος αισθητήρα, παρουσιάστηκαν δύο οικογένειες κόμβων αισθητήρων (mica και telosb), δόθηκαν τα βασικά χαρακτηριστικά Quality of service και οι διάφορες μετρικές για την κατανάλωση ενέργειας και αναφέρθηκαν οι προκλήσεις των ασύρματων δικτύων που πρέπει να αντιμετωπιστούν.

Ακολούθησε η ανάλυση των μορφών κινητικότητας που μπορούμε να συναντήσουμε στα ασύρματα δίκτυα αισθητήρων καθώς και των δυνατοτήτων που προσφέρει η ελεγχόμενη κινητικότητα. Αφού αναφέρθηκαν τα ζητήματα κινητικότητας που βρίσκονται υπό έρευνα, δόθηκε έμφαση σε πρωτόκολλα συλλογής δεδομένων με πολλαπλά κινητά sink και εξηγήθηκε ένας εύκολος τρόπος καθορισμού θέσης με τη χρήση κινητών sink.

Στη συνέχεια, αφού δόθηκε ο ορισμός της τρύπας κάλυψης, παρουσιάστηκαν τρόποι αντιμετώπισης των τρυπών κάλυψης με τη χρήση στατικών κόμβων, κινητών κόμβων και με τη χρήση στατικών και κινητών κόμβων ταυτόχρονα.

Το υπόλοιπο μέρος της διπλωματικής ασχολήθηκε με την ανάπτυξη ενός αλγορίθμου ανίχνευσης και ελαχιστοποίησης τρυπών κάλυψης με την προσθήκη επιπρόσθετων κινητών κόμβων αισθητήρων και τα στάδια που ακολούθησα για την υλοποίηση του σε πιλοτικό δίκτυο.

Η διπλωματική κλείνει με την παρουσίαση διαφόρων πλατφόρμων για την δημιουργία κινητών κόμβων στα ασύρματα δίκτυα αισθητήρων.

8.2 Συμπεράσματα

Το μεγάλο εύρος των εφαρμογών, που μπορούν να αναπτυχθούν με τη χρήση των ασύρματων δικτύων, τα καθιστά πολυδύναμα και σύντομα μέρος της καθημερινής μας ζωής. Έτσι, παράγοντες όπως η ευελιξία, η μεγάλη ανοχή σε λάθη, η υψηλή αξιοπιστία αίσθησης, το χαμηλό κόστος και η γρήγορη ανάπτυξη ενός δικτύου αισθητήρων αποτελούν πρόκληση για έρευνα.

Πρόσφατα μια νέα κατηγορία για σημαντικές εφαρμογές δικτύων αισθητήρων εμφανίζεται όπου η κινητικότητα είναι θεμελιώδης χαρακτηριστικό για τα συστήματα που εξετάζονται. Τέτοια είδη δικτύων είναι πολύτιμα σε περιπτώσεις που η παραδοσιακή ανάπτυξη σταθερών κόμβων αποτυγχάνει ή δεν είναι επιτρεπτή. Σε τέτοιες εφαρμογές οι αισθητήρες συνάπτονται πάνω σε αυτοκίνητα, robots, ζώα ή ανθρώπους που κινούνται γύρω σε μεγάλες γεωγραφικές περιοχές. Οι τρεις κύριες μορφές κινητικότητας στα ασύρματα δίκτυα αισθητήρων είναι η περίπτωση που έχουμε κινητούς κόμβους, η περίπτωση που έχουμε κινητά sinks, και η περίπτωση κινητικότητας βάση γεγονότων.

Η ελεγχόμενη κινητικότητα προσφέρει πολλά πλεονεκτήματα, όπως τη μεγιστοποίηση της κάλυψης της περιοχής του δικτύου, τον περιορισμό των προβλημάτων που παρουσιάζονται λόγω τοπικής και χρονικής ανωμαλίας στα στατικά ασύρματα δίκτυα, τον καλύτερο έλεγχο της τοπολογίας του δικτύου και την επιδιόρθωση τρυπών κάλυψης των δικτύων που μπορούν να δημιουργηθούν με το θάνατο συγκεκριμένων κόμβων. Ακόμη μπορεί να αυξήσει την ενεργειακή χωρητικότητα του δικτύου, να μειώσει τα λάθη κατά τη μεταφορά δεδομένων, να βοηθήσει στη συγκομιδή ενέργειας, στον υπολογισμό για τον καθορισμό θέσης των κόμβων και στον εντοπισμό γεγονότων, καθώς και στην απομόνωση από τους κακόβουλους ή ελαττωματικούς κόμβους.

Ζητήματα που μελετούνται σήμερα και αφορούν την κινητικότητα στα ασύρματα δίκτυα αισθητήρων συμπεριλαμβάνουν το adaptive localization, την κατανεμημένη χαρτογράφηση, την κάλυψη, το μαζικό επαναπρογραμματισμό, την κατανεμημένη ρύθμιση μετρήσεων, την επιδιόρθωση του δικτύου, την κατανεμημένη αποθήκευση πληροφοριών, τη συνάθροιση πληροφοριών και τεχνικές querying.

Η διαδικασία της μεταφοράς των δεδομένων, που καταγράφουν οι κόμβοι στο sink σε ένα στατικό δίκτυο, καταναλώνει μεγάλα ποσά ενέργειας ειδικά στην περιοχή που βρίσκεται το sink, όπου οι κόμβοι πρέπει να μεταδίδουν τα δεδομένα τα οποία έρχονται από κόμβους που βρίσκονται πιο μακριά. Ο τερματισμός της λειτουργίας αυτών των κόμβων λόγω εξάντλησης της ενέργειάς τους οδηγεί σε αποσυνδεδεμένα δίκτυα και σε δίκτυα που δεν λειτουργούν σωστά. Μια καλή λύση για την αντιμετώπιση αυτού του προβλήματος είναι να κινείται το sink μέσα στην περιοχή του δικτύου και έτσι θα βρίσκεται σχεδόν πάντα κοντά σε ένα υποσύνολο από αισθητήρες και θα μπορεί να λαμβάνει τα δεδομένα από αυτούς με πολύ λίγη κατανάλωση ενέργειας. Αυτή η προσέγγιση έχει πολλά επιπρόσθετα πλεονεκτήματα, όπως:

- Επιτρέπει την παρακολούθηση μιας περιοχής με τη χρήση λιγότερων αισθητήρων, μειώνοντας το συνολικό κόστος λειτουργίας του δικτύου.
- Επιτρέπει στους κόμβους αισθητήρα να μειώσουν την εμβέλεια μετάδοσής τους σε χαμηλότερη τιμή εξοικονομώντας ενέργεια.
- Μπορούν να προσπεραστούν προβληματικές περιοχές στις οποίες οι αισθητήρες δεν μπορούν να λειτουργήσουν.
- Μειώνεται η πιθανότητα για τη μετάδοση λάθους και οι συγκρούσεις των πακέτων, αφού ελαττώνεται ο αριθμός των hops για την επικοινωνία.
- Η παρουσία του δικτύου δεν ανιχνεύεται εύκολα λόγω της χαμηλής δύναμης μετάδοσης που χρησιμοποιείται.

Όμως η διάσχιση του δικτύου πρέπει να γίνεται με αποτελεσματικό τρόπο, γιατί η αποτυχία της επίσκεψης μερικών περιοχών του δικτύου μπορεί να οδηγήσει σε απώλεια δεδομένων και η μη συχνή επίσκεψη σε μερικές περιοχές μπορεί να οδηγήσει σε μεγάλες καθυστερήσεις παράδοσης δεδομένων.

Η Στρατηγική της ολοκληρωτικής κινητικότητας ελαχιστοποιεί την κατανάλωση ενέργειας, αφού γίνεται μόνο μια μετάδοση ανά ανιχνευμένο γεγονός. Όμως χρειάζεται αρκετός χρόνος για την επίσκεψη όλων των κόμβων, ειδικά σε περιπτώσεις που τα sinks δεν είναι πολλά. Η Μικτή Στρατηγική από στατικά και κινητά Sinks και το πρωτόκολλο Equidistribution αποτελούν μια πιο δίκαιη λύση για τη συλλογή των δεδομένων από ότι η Στρατηγική της ολοκληρωτικής κινητικότητας. Η απόδοση που έχει επιτευχθεί σε καθένα από τα πιο πάνω πρωτόκολλα, είναι ανάλογη με τον αριθμό των κινητών sink.

Το πρωτόκολλο Deterministic Walk with Multi-hop Data Propagation έχει λιγότερες καθυστερήσεις από τους αλγόριθμους Random Walk and passive data collection, Partial Random Walk with Limited Multi-Hop

Data propagation και Biased Random Walk with Passive Data Collection. Επιπρόσθετα, η επιλογή της τροχιάς μεγέθους L δημιουργεί ένα trade-off μεταξύ του κόστους στο sink και του κόστους στους κόμβους αισθητήρων.

Η μέθοδος καθορισμού θέσης με τη χρήση κινητού sink κατά την οποία το κινητό sink κινείται μέσα στο δίκτυο και υλοποιεί μια γεωμετρική αναπαράσταση της τοπολογίας των ασύρματων αισθητήρων βασιζόμενο στο επικοινωνιακό εύρος του κάθε κόμβου αισθητήρα δεν είναι μόνο οικονομική, αλλά είναι και αρκετά απλή για να υλοποιηθεί και ταυτόχρονα αρκετά αποδοτική.

Οι τρύπες κάλυψης είναι περιοχές του δικτύου οι οποίες δεν καλύπτονται από κανένα κόμβο και δημιουργούνται από την τυχαία εναέρια ανάπτυξη του δικτύου, από την ύπαρξη εμποδίου στην περιοχή και πιο συχνά με το θάνατο ενός κόμβου. Αρκετοί αλγόριθμοι έχουν αναπτυχθεί για την αντιμετώπιση των τρυπών κάλυψης στα ασύρματα δίκτυα αισθητήρων. Οι αλγόριθμοι αυτοί ταξινομούνται σε τρεις κατηγορίες, στην κατηγορία των κινητών κόμβων αισθητήρων, στην κατηγορία στατικών κόμβων αισθητήρων και στην κατηγορία των υβριδικών δικτύων.

Ο αλγόριθμος που ανάπτυξα για την ανίχνευση και ελαχιστοποίηση τρυπών κάλυψης με την προσθήκη επιπρόσθετων κινητών κόμβων αισθητήρων, προσπαθεί να μειώσει τις τρύπες κάλυψης όσο το δυνατόν γίνεται, με την προσθήκη μειωμένου αριθμού κινητών κόμβων αισθητήρων. Η μέθοδος για τη δρομολόγηση του κινητού κόμβου στη βέλτιστη θέση ανάπτυξης, εγγυάται ότι ο κινητός κόμβος θα φτάσει με επιτυχία στον προορισμό του στις περισσότερες περιπτώσεις. Όμως, δεν εγγυάται ότι ο κινητός κόμβος θα ακολουθήσει το πιο σύντομο δρομολόγιο προς τον προορισμό του. Η επέκταση του αλγορίθμου που εισηγήθηκα τον κάνει ικανό να τρέξει κατανεμημένα σε μεγαλύτερου μεγέθους δίκτυα και να είναι πιο επεκτάσιμος.

Τέλος, ο αλγόριθμος μπορεί να ενταχτεί στην κατηγορία των κινητών ασύρματων αισθητήρων, αφού η ελαχιστοποίηση των τρυπών κάλυψης πετυχαίνεται με τη μετακίνηση κινητού κόμβου στην ακάλυπτη περιοχή. Χρησιμοποιεί την προσέγγιση του grid για τον υπολογισμό των τρυπών κάλυψης και προϋποθέτει ότι ο κάθε κόμβος αισθητήρα γνωρίζει τη θέση του μέσα στο δίκτυο. Η πρώτη προσέγγιση του αλγορίθμου είναι centralized και δεν επιβαρύνονται οι κόμβοι αισθητήρων με επιπρόσθετο υπολογισμό, αφού το μόνο που έχουν να κάνουν είναι να στείλουν τις συντεταγμένες και την εμβέλεια τους στο sink. Μόνο το sink και ο κινητός κόμβος αισθητήρα επιβαρύνονται με υπολογισμούς. Η επέκταση του αλγορίθμου είναι κατανεμημένη, αλλά οι κόμβοι αισθητήρων επιβαρύνονται με μέρος του υπολογισμού.

Μέσα από την διαδικασία υλοποίησης του αλγορίθμου που ανάπτυξα σε πιλοτικό δίκτυο, διαπίστωσα ότι χρειάζεται να καταβληθεί πάρα πολύ μεγάλη προσπάθεια ώστε να μπορεί να τρέξει με επιτυχία οποιοσδήποτε αλγόριθμος real time σε πραγματικό σενάριο. Τα προβλήματα που μπορούν να παρουσιαστούν μπορούν να οφείλονται σε πολλούς και διάφορους παράγοντες όπως για παράδειγμα λάθος στον κώδικα, μη λειτουργικό υλικό, ασύμβατα μέρη υλικού μεταξύ τους, καταλυμένων μπαταριών, βραχυκυκλωμάτων, ακόμα και λόγω ελαττωματικών εργαλείων υποστήριξης. Για την επίλυση τέτοιων προβλημάτων χρειάζεται μεθοδικός έλεγχος, έρευνα και σταδιακή επιμέρους ανάπτυξη του πιλοτικού δικτύου. Κάθε κομμάτι πρέπει να ελέγχεται ξεχωριστά και μόνο όταν είναι σίγουρο ότι τρέχει σωστά να ενσωματώνεται στο πιλοτικό δίκτυο.

8.3 Μελλοντική εργασία

Ως μελλοντική εργασία μπορεί να θεωρηθεί η ενσωμάτωση ενός αλγορίθμου για τον καθορισμό της θέσης των ασύρματων κόμβων αισθητήρων στο πιλοτικό δίκτυο που έχω δημιουργήσει, όπως για παράδειγμα η πιλοτική υλοποίηση του αλγόριθμου καθορισμού θέσης με τη χρήση κινητού sink που

αναλύω στο κεφάλαιο 3 ή να γίνει ο καθορισμός τοποθεσίας με τη χρήση του Cricket location estimation system. Επίσης, μπορεί να ενσωματωθεί πυξίδα στο boebot και να προγραμματιστεί ώστε να ελέγχει και να διορθώνει την κατεύθυνση του boebot, καθώς και ενσωμάτωση και προγραμματισμός κάποιου acceleration sensor ώστε να υπολογίσει και να διατηρεί σταθερή την ταχύτητά του.

Ακόμη μπορεί να γραφτεί ένας προσομοιωτής ο οποίος να προσομοιώνει και να συγκρίνει διάφορους αλγόριθμους για την ελαχιστοποίηση των τρυπών κάλυψης ώστε να μπορεί να αξιολογηθεί σωστά ο αλγόριθμος που ανάπτυξα.

Τέλος θα μπορούσε να υλοποιηθεί σε πιλοτικό δίκτυο η επέκταση του αλγορίθμου που έφτιαξα ή και να συγκριθεί με διάφορους αλγόριθμους για την ελαχιστοποίηση των τρυπών κάλυψης.

Αυτή η διπλωματική εργασία μπορεί να χρησιμοποιηθεί από άλλους φοιτητές που ενδιαφέρονται να ασχοληθούν με την κινητικότητα στα ασύρματα δίκτυα αισθητήρων και να αναπτύξουν τους δικούς τους αλγόριθμους και να τους υλοποιήσουν σε πιλοτικό δίκτυο.

Βιβλιογραφία

[1] Saad Ahmed, Munir Yu Wen, Bin Ma Jian. “Efficient Minimum Cost Area Localization for Wireless Sensor Network with a Mobile Sink”, 2007.

[2] Nadeem Ahmed, Salil S. Kanhere, Sanjay Jha. “Sink The Hole Problem in wireless Sensor Networks: A Survey”, 2005.

[3] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci “A Survey on Sensor Networks.”, August 2002.

[4] Ghosh Amitabha. “Estimating Coverage Holes and Enhancing Coverage in Mixed Sensor Networks”, 2004.

[5] A. Ananda, Mun Choon Chan Wei Tsang Ooi , Edited by Rajeev Shorey “Mobile, Wireless, And Sensor Networks Technology, Applications, And Future Directions.”, pp.176, April 2006.

[6] Nuzhet Atay, Burchan Bayazit. “Mobile Wireless Sensor Network Connectivity Repair with K-Redundancy”, September 2007.

[7] Maxim A. Batalin and Gaurav S. Sukhtame. “Coverage, exploration and deployment by a mobile robot and communication network.” Telecommunication Systems Journal, Special Issue on Wireless Sensor Networks, 26(2):181–196, 2004.

[8]Sarah Bergbreiter, Jameson Lee, Dr. Kris Pister, “Cotsbots” [Online]. Available: http://www-bsac.eecs.berkeley.edu/projects/cotsbots/.

[9] Sarah Bergbreiter and K.S.J. Pister "CotsBots: An off-the-shelf Platform for Distributed Robotics", October 2003.

[10] Juan Antonio Brena Moral , Develop Lejos Programs Step by Step July 2008 [E-book] Available: http://www.juanantonio.info/lejos-ebook/.

[11] Juan Antonio Brena Moral, Using I2C with java leJos Augoust 2008, [E-book] Available: http://www.juanantonio.info/lejos-ebook/.

[12]Phil Buonadonna, Arch Rock Corporation, "Low-Level I/O". [Online]. Available: http://www.tinyos.net/dist-2.0.0/tinyos-2.0.2/doc/txt/tep117.txt.

[13] Ioannis Chatzigiannakis, Athanasios Kinalis, Sotiris Nikoletseas "Sink Mobility Protocols for Data Collection in Wireless Sensor Networks", 2006.

[14] B.Chen, K. Jamieson, H. Balakrshnan, R. Morris. SPAN: An energy-efficient coordination algorithm for topology maintenance in adhoc wireless networks. In *Proceedings of the ACM/IEEE MobiCom '01*, July 2001.

[15] Pengyu Chen's "Mobile Actuator Sensor Networks (MAS-net)", February 2005 [Online]. Available: http://cc.usu.edu/~pchen/masnet.htm.

[16] Peter Corke, Steven Hrabar, Ronald Peterson, Daniela Rus, Srikanth Saripalli, and Gaurav Sukhatme." Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle." In Proceedings of the IEEE 2004 International Conference on Robotics and Automation, Volume 4, pp. 3602–3608, May 2004.

[17] Karthik Dantu, Mohammad Rahimi, Hardik Shah, Sandeep Babel, Amit Dhariwal and Gaurav Sukhatme "Robomote: Enabling Mobility In Sensor Networks.", 2004.

[18] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo, The Collection Tree Protocol (CTP), February 2007[Online]. Available: http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html

[19] Saurabh Ganeriwal, Aman Kansal, and Mani B.Srivastava." Self aware actuation for fault repair in sensor networks." In Proceedings of the IEEE International Conference on Robotics and Automation(ICRA), May 2004.

[20] Michael Gasperi, Philippe Hurbain, Isabelle Hurbain, Extreme NXT: Extending the Lego Mindstorms NXT to the Next Level, Ed. Jim Sumser 2007.

[21] Nojeong Heo and Pramod K. Varshney." An intelligent deployment and clustering algorithm for a distributed mobile sensor network." In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, volume 5, pp. 4576–4581, Oct 2003.

[22] Andrew Howard, Maja J Mataric, and Gaurav S. Sukhatme. "Mobile sensor network deployment using potential fields:A distributed, scalable solution to the area coverage problem." In 6th International Symposium on Distributed Autonomous Robotics Systems (DARS02), June 2002.

[23] Andrew Howard, Maja J. Mataric, and Gaurav S. Sukhatme. "An incremental self-deployment algorithm for mobile sensor networks."

Autonomous Robots, Special Issue on Intelligent Embedded Systems, 13(2):113–126, September 2002.

[24] Chi-Fu Huang and Yu-Chee Tseng. “The Coverage Problem in a Wireless Sensor Network”, September 2003.

[25] Jie Jiang and Wenhua Dou. “A coverage preserving density control algorithm for wireless sensor networks.” In ADHOC-NOW’04, pp. 42–55. Springer-Verlag, July 2004.

[26] Zhen Jiangm, Jie Wu, Afrand Agah, Bin Lu “Topology Control for Secured Coverage in Wireless Sensor Networks”, October 2007.

[27] Holger Karl, Andreas Willig. “Protocols and Architectures For Wireless Sensor Networks” Ed. John Wiley & Sons Inc., pp. 1-88, 2005

[28] Lames Floyd Kelly, Lego Mindstorms NXT: The Mayan Adventure Ed. Jim Sumser, 2006.

[29] Athanasios Kinalis and Sotiris Nikoletseas. “Scalable Data Collection Protocols for Wireless Sensor Networks with Multiple Mobile Sinks”, 2007.

[30] M. Brett McMickell, Bill Goodwine, Luis Antonio Montestruque “MICAbot: A Robotic Platform for Large-scale Distributed Robotics”, September 2003.

[31] Philip Levis and Gilman Tolle, “Dissemination of Small Values” [Online].Available:http://www.tinyos.net/tinyos-2.x/doc/html/tep118.html.

[32] Christopher M. Cianci, Xavier Raemy, Jim Pugh, and Alcherio MartinoliCommunication in a Swarm of Miniature. “Robots: The e-Puck as an Educational Tool for Swarm Robotics”, 2007.

[33] Luis E. Navvarro - Serment, Robert Grabowski, Christiaan J.J. Paredis, and Pradeep K. Khosla. “Millibots:The Development of a Frameworkand Algorithms for a Distributed Heterogeneous Robot Team”, December 2002.

[34] nescc.sourceforge.net, “Network Types: Language Support for Messaging in the Heterogeneous Networks” 16 December 2004 [Online]. Available: http://nescc.sourceforge.net/networktypes/index.html.

[35] Tanja Neumann, “Lego NXT sensors”, [Online]. Available: http://www.stud.uni-karlsruhe.de/~uqbdx/lego/nxt.html.

[36] Open-zb.net, “OpenSource Toolset for IEEE 802.15.4 and zigBee” [Online]. Available: http://www.open-zb.net/.

[37] Parallax inc, “Boe-Bot Robot”, [Online]. Available: http://www.parallax.com/tabid/411/Default.aspx.

[38] Robot Electronics,” LCD03 - I2C/Serial LCD Technical Documentation”, Online]. Available: http://www.robot-electronics.co.uk/htm/Lcd03tech.htm.

[39] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Philip Levis, “TEP 119: Collection”, February 2006, [Online]. Available: http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep119.html.

[40] Di Tian, Nicolas D. Georganas. “A coverage preserving node scheduling scheme for large wireless sensor networks.” , In Proceedings of the first ACM WSNA’02, Sep 2002.

[41] The Software for Minimal I2C, [Online]. Available: http://boldinventions.com/i2c_simple_implement.html.

[42] TinyOS Documentation Wiki, “mts400”, November 2008. [Online]. Available: http://docs.tinyos.net/index.php/MTS400.

[43] TinyOS Documentation, ”Multihop Routing”, September 2003, [Online]. Available: http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html.

[44] TinyOS Documentation Wiki, “TinyOS Tutorials”, 30 March 2009, [Online]. Available: http://docs.tinyos.net/index.php/TinyOS_Tutorials.

[45]TinyOS Documentation Wiki, “Tymo”, [Online]. Available: http://docs.tinyos.net/index.php/Tymo.

[46] Sivan Toledo, “I2C Interfacing Part1: Adding Digital I/O Ports”, October 2006, [Online]. Available: http://www.tau.ac.il/~stoledo/lego/i2c-8574-l272/.

[47] Total Phase Inc, “I2C Background”, [Online]. Available: http://www.totalphase.com/support/kb/10037/.

[48] Guiling Wang, Guohong Cao, and Tom La Porta. “Movement-assisted sensor deployment”, In IEEE INFOCOM 2004, June 2004.

[49] Guiling Wang, Guohong Cao, and Tom La Porta.” A bidding protocol for deploying mobile sensors.” In 11th IEEE International Conference on Network Protocol ICNP ’03, pp. 315–324, Nov 2003.

[50] Xiaorui Wang, Guoliang Xing, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. “Integrated coverage and connectivity configuration in wireless sensor networks.” In Proceedings of the ACM SenSys ’03, pp 28–39, Nov 2003.

[51] Axel Wolf, ESAcademy, “I2C (Inter-Integrated Circuit) Bus Technical Overview and Frequently Asked Questions (FAQ)”, [Online]. Available: http://www.esacademy.com/faq/i2c/index.htm.

[52] T. Yan, T. He, and J. Stankovic. “Differentiated surveillance for sensor networks.”, In Proceedings of the ACM SenSys ’03, Nov 2003.

[53] Honghai Zhang and Jennifer C. Hou. “Maintaining sensing coverage and connectivity in large sensor networks.” Technical Report UIUDCS-R-2003-2351, Department of Computer Science, University of Illinois at Urbana Champaign, 2003.

[54] Χαραλάμπους Γεώργιος. “Προγραμματισμός Lego Mindstorm NXT για την υποστήριξη κινητικότητας στα ασύρματα Δίκτυα Αισθητήρων ”, pp.59-63, 2009.

Παράρτημα Α

Πιο κάτω ακολουθουν οι οδηγίες για την εγκατάσταση του TinyOs1.x και του TinyOs2.1 στα λειτουργικά συστήματα Windows XP και Ubuntu αντίστοιχα.

ΕΓΚΑΤΆΣΤΑΣΗ TinyOs 1.x

Windows XP με Cygwin:

Αφού εγκαταστήσετε το Cygwin:

1. Κατεβάστε από την σελίδα της java την τελευταία έκδοση του JDK από τη σελίδα *http://java.sun.com* και εγκαταστείστε το στον υπολογιστή σας.
2. Κατεβάστε το πακέτο javax.comm της Sun από τη σελίδα *http://java.sun.com/products/javacomm/*. Το πακέτο αυτό χρειάζεται ώστε να μπορεί να τρέξει ο Serial Forwarder. Εγκαταστείστε το με τον ακόλουθο τρόπο:
   a. *unzip javacomm20-win32.zip*
   b. *cd commapi*
   c. *cp win32com.dll "c:\Program Files\jdk\jre\bin"*
   d. *chmod 755 "c:\Program Files\jdk\jre\bin\win32com.dll"*
   e. *cp comm.jar "c:\Program Files\jdk\jre\lib\ext"*
   f. *cp javax.comm.properties "c:\Program Files\jdk\jre\lib"*
   g. `cp javax.comm.properties"c:\ProgramFiles\jdk\lib"`
3. Κατεβάστε το graphviz από τη σελίδα
   http://webs.cs.berkeley.edu/tos/dist-1.1.0/tools/windows/graphviz-1.10.exe
   και εγκαταστείστε το. Το πρόγραμμα αυτό μπορεί να χρησιμοποιηθεί για τη γραφική αναπαράσταση των component που αποτελούν τις εφαρμογές που θα γράψετε σε γλώσσα nesC.

4. Κατεβάστε από τη σελίδα *http://webs.cs.berkeley.edu/tos/dist-1.1.0/tools/windows* τα ακόλουθα rpms ώστε να μπορείτε να μεταγλωττίσετε με επιτυχία τις εφαρμογές σας:
   avarice-2.0.20030825cvs-1w.cygwin.i386.rpm
   avr-binutils-2.13.2.1-1w.cygwin.i386.rpm
   avr-gcc-3.3tinyos-1w.cygwin.i386.rpm
   avr-insight-pre6.0cvs.tinyos-1w.cygwin.i386.rpm
   avr-libc-20030512cvs-1w.cygwin.i386.rpm
   Κατεβάστε και από τη σελίδα
   http://webs.cs.berkeley.edu/tos/dist-1.1.0/tinyos/windows
   τα ακόλουθα rpms
   nesc-1.1-1w.cygwin.i386.rpm
   tinyos-tools-1.1.0-1.cygwin.i386.rpm
   tinyos-1.1.0-1.cygwin.noarch.rpm
   Για να τα εγκαταστήσετε τα πιο πάνω rpms, εκτελέστε την εντολή *"rpm --ignoreos -ivh *.rpm"* στο αρχείο που έχετε αποθηκεύσει τα αρχεία αυτά.

5. Τέλος καθορίζουμε τις μεταβλητές περιβάλλοντος:
   TOSROOT="/opt/tinyos-1.x"
   TOSDIR="$TOSROOT/tos"
   CLASSPATH=`$TOSROOT/tools/java/javapath`
   MAKERULES="$TOSROOT/tools/make/Makerules"
   export TOSROOT
   export TOSDIR
   export CLASSPATH
   export MAKERULES
   type java >/dev/null 2>/dev/null || PATH=`/usr/bin/tos-locate-jre --java`:$PATH
   type javac >/dev/null 2>/dev/null || PATH=`/usr/bin/tos-locate-jre --javac`:$PATH
   echo $PATH | grep -q /usr/local/bin || PATH=/usr/local/bin:$PATH

Το tinyOs θα το βρείτε εγκατεστημένο στο /opt/tinyos-1.x.

- Η μεταγλώττιση και εγκατάσταση σε κόμβους αισθητήρα micaz και telosb χρησιμοποιώντας το programming board mib510 γίνεται με τις εντολές:

MIB510=/dev/ttyS(PORT_NUMBER -1) make micaz install.(node_id) mib510
MIB510=/dev/ttyS(PORT_NUMBER -1) make telosb install.(node_id) mib510

ΕΓΚΑΤΆΣΤΑΣΗ TinyOs 2.x

Debian – Ubuntu installation:

1. Τοποθέτησε το πιο κάτω στο repository

deb http://tinyos.stanford.edu/tinyos/dists/ubuntu <distribution> main

2. Κάνε update το respository με την πιο κάτω εντολή:

sudo apt-get update

3. Τρέξετε την πιο κάτω εντολή για εγκατάσταση της τελευταίας έκδοσης του tinyOs και όλων των εργαλείων που χρειάζεται για να λειτουργήσει:

   sudo apt-get install tinyos

4. Πρόσθεσε το πιο κάτω στο αρχειο ~/.bashrc ή στο αρχείο ~/.profile το οποίο βρίσκεται στο home directory για τη ρύθμηση του περιβάλλοντος για το TinyOs:

#Sourcing the tinyos environment variable setup script
source /opt/tinyos-2.1.0/tinyos.sh

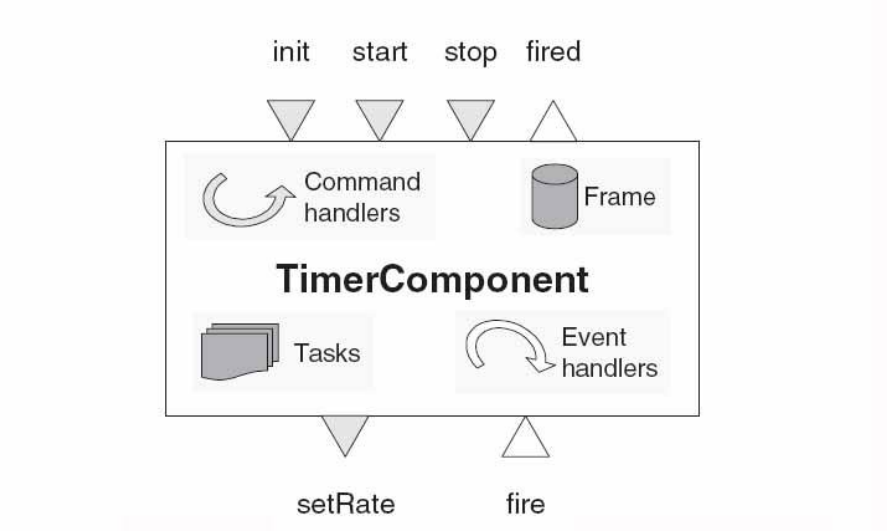
Το tinyOs θα το βρείτε εγκατεστημένο στο /opt/tinyos-2.1.0.

- Η μεταγλώττιση και εγκατάσταση σε κόμβους αισθητήρα micaz και telosb χρησιμοποιώντας το programming board mib510 γίνεται με τις εντολές:

MIB510=/dev/ttyUSB0 make micaz install.(node_id) mib510
MIB510=/dev/ttyUSB0 make telosb install.(node_id) mib510

Λίγα λόγια για το λειτουργικό TinyOs:

Το λειτουργικό σύστημα TinyOS μαζί με τη γλώσσα προγραμματισμού nesC υποστηρίζει modularity και "event-based" προγραμματισμό με τη χρήση των components. Ένα component περιέχει σημασιολογικά συσχετιζόμενη λειτουργία. Ένα τέτοιο component περιλαμβάνει την απαιτούμενη πληροφορία μιας κατάστασης σε ένα frame, τον κώδικα του προγράμματος για εκτέλεση κανονικών στόχων και χειριστήρια για τα events και τα commands. Τα components κατατάσσονται ιεραρχικά από τα low-level components κοντά στο υλικό μέχρι τα high level components τα οποία υλοποιούν την πραγματική εφαρμογή. Τα events δημιουργούνται στο υλικό και περνούν προς τα πάνω από low-level σε high level components. Από την άλλη τα commands περνούν από τα high-level στα low-level components.



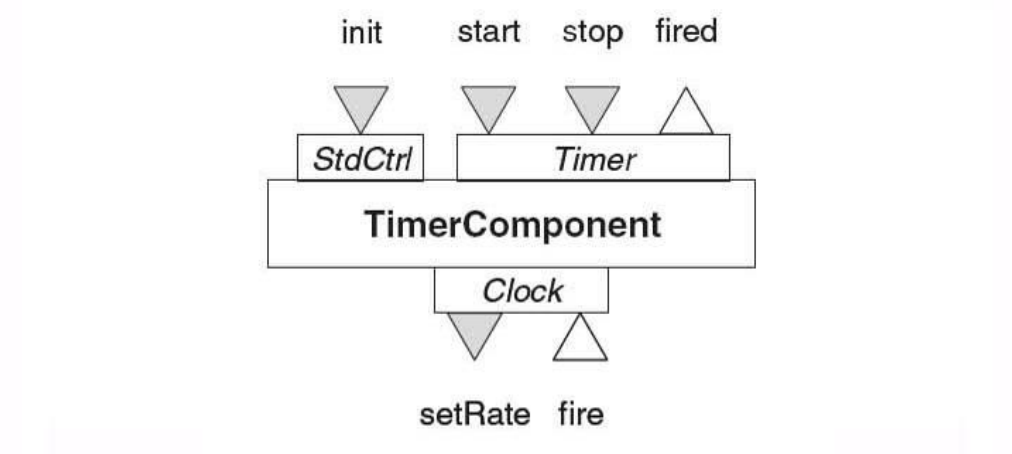
Στην πιο πάνω εικόνα βλέπουμε ένα timer component, όπου περιγράφεται ένα πιο αφαιρετικό version ενός απλού hardware time. Καταλαβαίνει 3 εντολές ("init", "start", and "stop") και μπορεί να χειριστεί ένα event ("fire") από ένα άλλο component. Μπορεί να εκδώσει την εντολή "setRate" στο άλλο component και μπορεί να στείλει το ίδιο ένα event "fired".

Είναι σημαντικό να αναφερθεί ότι και commands και τα event handlers πρέπει να τρέξουν μέχρι να ολοκληρωθούν και μπορούν μόνο για να εκτελέσουν απλά καθήκοντα triggering. Συγκεκριμένα τα commands δεν πρέπει να μπλοκάρονται ή να περιμένουν για ένα απροσδιόριστο χρονικό διάστημα και απλά είναι ένα αίτημα πάνω στο οποίο θα εκτελεστεί κάποια εργασία ενός component χαμηλότερης ιεραρχίας. Παρομοίως ένα event handler αφήνει πληροφορίες στο frame του component του και διευθετεί μια εργασία να εκτελεστεί αργότερα. Μπορεί επίσης να στείλει εντολές σε άλλα components ή να ενημερώσει άμεσα ένα event στην πιο πάνω ιεραρχία.

Η πραγματική υπολογιστική δουλεία γίνεται μέσα στα tasks τα οποία στο TinyOS πρέπει να τρέξουν μέχρι το τέλος, αλλά μπορούν να διακοπούν από διάφορα handlers. Το πλεονέκτημα είναι διπλό, διότι δεν υπάρχει καμία ανάγκη για τη διαχείριση στοίβας και τα tasks είναι ατομικά όσον αφορά τη σχέση μεταξύ τους. Επειδή τα tasks έχουν προκληθεί από τα handlers, γίνονται φαινομενικά ταυτόχρονα το ένα σε σχέση με το άλλο. Η διαιτησία μεταξύ των tasks γίνεται από ένα απλό First In First Out (FIFO) scheduler το οποίο απενεργοποιεί τον κόμβο, όταν δεν υπάρχουν στην ουρά tasks να περιμένουν και δεν εκτελείται κάποιο task.

Ένα component μπορεί να επικαλεστεί συγκεκριμένες εντολές από χαμηλότερης ιεραρχίας component και να πάρει συγκεκριμένα events από αυτό. Η γλώσσα προγραμματισμού nesC τυποποιεί αυτή την διαίσθηση με το να επιτρέπει στον προγραμματιστή να μπορεί να ορίσει interface types που καθορίζουν τα events και τα commands τα οποία ανήκουν από κοινού. Αυτό επιτρέπει στη δημιουργία split-phase προγραμματιστικού στυλ (χωρισμός του request και της πληροφορίας για την απάντηση του request σε δύο φάσεις) βάζοντας τα commands και τα αντίστοιχα events για τον τερματισμό του command μέσα στο ίδιο interface. Έτσι τα components παρέχουν συγκεκριμένα interfaces στους

χρήστες τους (components) και εκείνα με τη σειρά τους χρησιμοποιούν τα interfaces άλλων components.



Η πιο πάνω εικόνα δείχνει το Timer component στο οποίο χρησιμοποιείται ένα clock interface και παρέχει δυο άλλα interfaces το StdCtrl και το Timer. Πιο κάτω φαίνεται ο κώδικας του σε nesC .

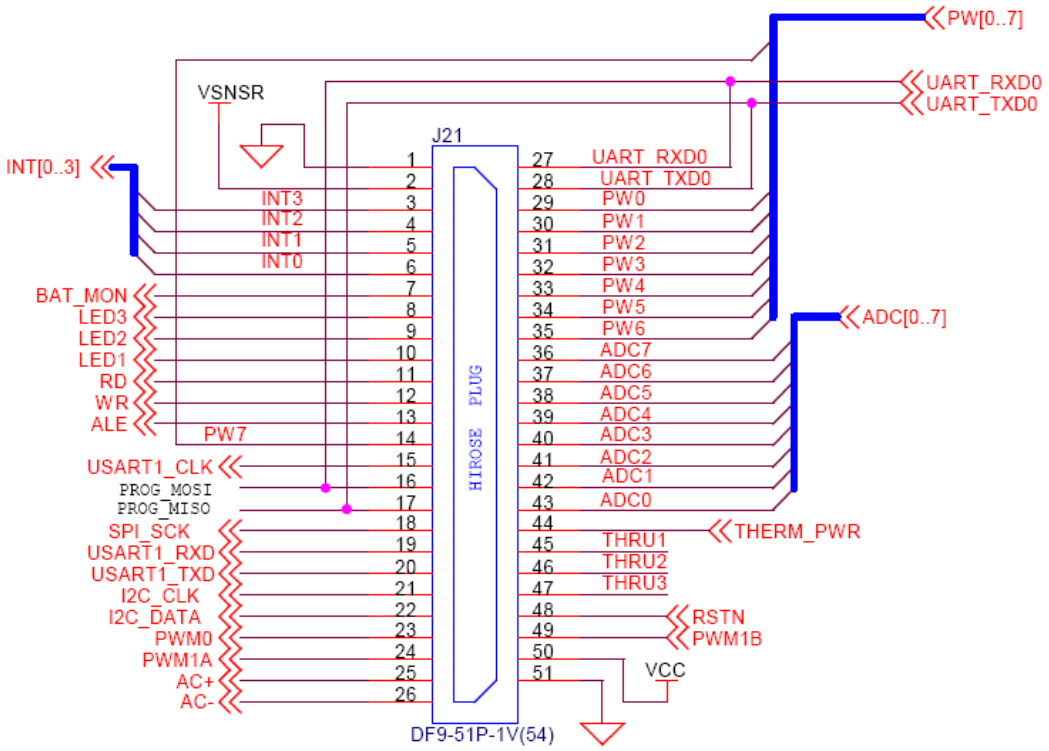
```
interface StdCtrl {
  command result_t init();
}

interface Timer {
  command result_t start (char type, uint32_t interval);
  command result_t stop ();
  event result_t fired();
}

interface Clock {
  command result_t setRate (char interval, char scale);
  event result_t  fire ();
}

module TimerComponent {
  provides {
    interface StdCtrl;
    interface Timer;
  }
  uses interface Clock as Clk;
}
```

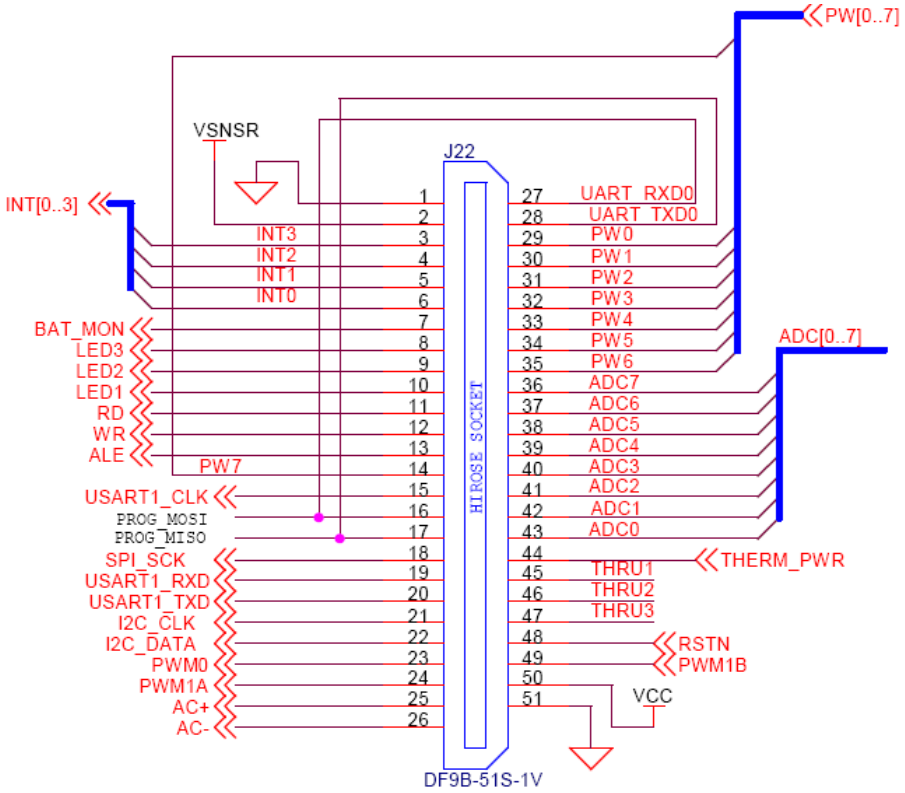
Τέτοιου είδους components ή modules μπορούν να συνδυαστούν σε μεγάλα configurations ενώνοντας τα κατάλληλα interfaces μαζί. Για να γίνει σωστά η ένωση πρέπει να ενώνονται μεταξύ τους τα components που έχουν το σωστό τύπο interface.

Πιο κάτω βλέπουμε ένα παράδειγμα configuration σε κώδικα nesC

```
configuration CompleteTimer {
  provides {
    interface StdCtrl;
    interface Timer;
  }
  implementation {
    components TimerComponent, HWClock;
    StdCtrl = TimerComponent.HWClock;
    Timer = TimerComponent.Timer;
    TimerComponent.Clk = HWClock.Clock;
  }
}
```

Ακολουθεί το διάγραμμα των εξόδων του 51-pin expansion slot και socket του Micaz το οποίο μελετήθηκε για τη δημιουργία των custom-made καλωδίων που έφτιαξα.





PIN	NAME	DESCRIPTION
1	GND	GROUND
2	VSNSR	SENSOR SUPPLY
3	INT3	GPIO
4	INT2	GPIO
5	INT1	GPIO
6	INT0	GPIO
7	BAT_MON	BATTERY VOLTAGE MONITOR ENABLE
8	LED3	LED3
9	LED2	LED2
10	LED1	LED1
11	RD	GPIO
12	WR	GPIO
13	ALE	GPIO
14	PW7	POWER CONTROL 7
15	USART1_CLK	USART1 CLOCK
16	PROG_MOSI	SERIAL PROGRAM MOSI
17	PROG_MISO	SERIAL PROGRAM MISO
18	SPI_SCK	SPI SERIAL CLOCK
19	USART1_RXD	USART1 RX DATA
20	USART1_TXD	USART1 TX DATA
21	I2C_CLK	I2C BUS CLOCK
22	I2C_DATA	I2C BUS DATA
23	PWM0	GPIO/PWM0
24	PWM1A	GPIO/PWM1A
25	AC+	GPIO/AC+
26	AC-	GPIO/AC-

PIN	NAME	DESCRIPTION
27	UART_RXD0	UART_0 RECEIVE
28	UART_TXD0	UART_0 TRANSMIT
29	PW0	POWER CONTROL 0
30	PW1	POWER CONTROL 1
31	PW2	POWER CONTROL 2
32	PW3	POWER CONTROL 3
33	PW4	POWER CONTROL 4
34	PW5	POWER CONTROL 5
35	PW6	POWER CONTROL 6
36	ADC7	ADC INPUT 7 - BATTERY MONITOR/JTAG TDI
37	ADC6	ADC INPUT 6 / JTAG TDO
38	ADC5	ADC INPUT 5 / JTAG TMS
39	ADC4	ADC INPUT 4 / JTAG TCK
40	ADC3	ADC INPUT 3
41	ADC2	ADC INPUT 2
42	ADC1	ADC INPUT 1
43	ADC0	ADC INPUT 0 / RSSI MONITOR
44	THERM_PWR	TEMP SENSOR ENABLE
45	THRU1	THRU CONNECT 1
46	THRU2	THRU CONNECT 2
47	THRU3	THRU CONNECT3
48	RSTN	RESET (NEG)
49	PWM1B	GPIO/PWM1B
50	VCC	DIGITAL SUPPLY
51	GND	GROUND

Παράρτημα Β

Σε αυτό το παράρτημα θα βρείτε τον κώδικα που γράφτηκε για την υλοποίηση όλων των πειραματικών δοκιμών που έχουν γίνει.

Κώδικας γραμμένος σε nesC που προγραμματίστηκε το sink και οι κόμβοι του πιλοτικού δικτύου για ελαχιστοποίηση των τρυπών κάλυψης

SendToBoebotAppC.nc

```
//////////////////////////////////////////////////////////////
//   This application is for the micaz motes of the         //
//   network and the sink. The micaz motes programmed       //
//   with this application send periodically their          //
//   coordinates to the sink using the TCP multihop         //
//   rooting protocol. The sink that is programmed with     //
//   this application, when it receives the coordinates     //
//   of all the motes in the network, estimates the         //
//   coverage holes of the network and send to the          //
//   mobile mote the coordinates of the optimum position    //
//   to minimize the uncovered area with a minimum          //
 //   number of extra mobile motes.                         //
//                                                          //
// Author: Constantinos Constantinides                      //
// Date last modified:  11/04/09                            //
//                                                          //
//////////////////////////////////////////////////////////////

configuration SendToBoebotAppC {}
implementation {
  /* comonents needed*/
  components MainC, SendToBoebotC, LedsC,
  new TimerMilliC(),/* for serial byte*/PlatformSerialC;


  SendToBoebotC.Boot -> MainC;
  SendToBoebotC.Timer -> TimerMilliC;
  SendToBoebotC.Leds -> LedsC;


  // Communication components.  These are documented in TEP 113:
  // Serial Communication, and TEP 119: Collection.
   components CollectionC as Collector,      // Collection layer
   ActiveMessageC,                    // AM layer
   new CollectionSenderC(AM_OSCILLOSCOPE), // Sends multihop RF
```

```
    SerialActiveMessageC,                  // Serial messaging
    new SerialAMSenderC(AM_OSCILLOSCOPE);    /* Sends to the serial
port*/


  SendToBoebotC.RadioControl -> ActiveMessageC;
  SendToBoebotC.SerialControl -> SerialActiveMessageC;
  SendToBoebotC.RoutingControl -> Collector;


  SendToBoebotC.Send -> CollectionSenderC;
  SendToBoebotC.SerialSend -> SerialAMSenderC.AMSend;
  SendToBoebotC.Snoop -> Collector.Snoop[AM_OSCILLOSCOPE];
  SendToBoebotC.Receive ->   Collector.Receive[AM_OSCILLOSCOPE];
  SendToBoebotC.RootControl -> Collector;


  /*componets for the pool*/
  components new PoolC(message_t, 10) as UARTMessagePoolP,
  new QueueC(message_t*, 10) as UARTQueueP;


  SendToBoebotC.UARTMessagePool -> UARTMessagePoolP;
  SendToBoebotC.UARTQueue -> UARTQueueP;



  /* for serial byte communication  --debuging */
  SendToBoebotC.UartControl -> PlatformSerialC;
  SendToBoebotC.UartByte    -> PlatformSerialC;


  /* components for communication with boebot*/
  components new AMSenderC(AM_BOE_BOT_MSG);
  components new AMReceiverC(AM_BOE_BOT_MSG);


  SendToBoebotC.BoebotSend -> AMSenderC;
}
```

SendToBoebotAppC.nc

```
//////////////////////////////////////////////////////////
// This application is for the micaz motes of the network and the      //
//  sink. The micaz motes programmed with this application send     //
```

```
//  periodically their coordinates to the sink using the TCP            //
//  multihop rooting protocol. The sink that is programmed with this //
//  application, when it receives the coordinates of all the motes      //
//  in the network, estimates the coverage holes of the network and   //
//  send to the mobile mote the coordinates of the optimum position  //
//  to minimize the uncovered area with  a minimum number of extra//
//  mobile motes                                                          //
//                                                                        //
// Author: Constantinos Constantinides                                   //
// Date last modified:  11/04/09                                          //
//                                                                        //
//////////////////////////////////////////////////////////////////////////

#include "Timer.h"
#include "MultihopOscilloscope.h"

module SendToBoebotC {
  uses {
    // Interfaces for initialization:
    interface Boot;
    interface SplitControl as RadioControl;
    interface SplitControl as SerialControl;
    interface StdControl as RoutingControl;

    // Interfaces for communication, multihop and serial:
    interface Send;
    interface Receive as Snoop;
    interface Receive;
    interface AMSend as SerialSend;
    interface CollectionPacket;
    interface RootControl;

        //for communication with the boebot sensor
    interface AMSend as BoebotSend;

        //for the pool
    interface Queue<message_t *> as UARTQueue;
    interface Pool<message_t> as UARTMessagePool;

    // Miscalleny:
    interface Timer<TMilli>;
    interface Leds;

        // for uart communication debuging
        interface StdControl as UartControl;
         interface UartByte;

         //for RF communication
```

```
        interface Packet;
        interface AMPacket;

  }
}

implementation {
  /*****************************************************/
  /************** application prototypes  ************/
  /*****************************************************/
  task void uartSendTask();
  void initializeAreaNodes( uint16_t );
  void initializeMaps();
  uint16_t drawNodeInMap(uint16_t , uint16_t, uint16_t );
  uint16_t findCoverageHoleCoordinates( uint16_t *, uint16_t *,
uint16_t);
  static void startTimer();
  static void fatal_problem();
  static void report_problem();
  static void report_sent();
  static void report_received();
  /*****************************************************/
  /**************    Global variables  *****************/
  /*****************************************************/
  uint8_t uartlen;
  bool found = FALSE;
  message_t sendboebotbuf;
  message_t sendbuf;
  message_t uartbuf;
  bool sendbusy=FALSE, uartbusy=FALSE;

  //array that would keep the id if every note in the network
  //it will be used to check the sink if all the motes of the network
  // have sent their coordinates at list one time successfully
  bool areaNodes[NODES_IN_THE_AREA];
  uint8_t areaNodesCounter=0;

  // the coverage map
  bool Map[AREA_X][AREA_Y];
  oscilloscope_t* messageReceived;


  /*Current local state - interval, version and accumulated readings*/
  oscilloscope_t local;
  boebot_t boebotSensorMessage;

  uint8_t reading; /* 0 to NREADINGS */
```

```
  /* When we head an Oscilloscope message, we check it's sample
count. If it's ahead of ours, we "jump" forwards (set our count to the
received count). However, we must then suppress our next count
increment. This is a very simple form of "time" synchronization (for an
abstract notion of time). */
  bool suppress_count_change;

  /*****************************************************************/
  /*On boot up, initialize radio and serial communications, and our own
state variables.                                                    */
/*****************************************************************/
  event void Boot.booted() {
    local.interval = DEFAULT_INTERVAL;
    local.id = TOS_NODE_ID;
    local.version = 0;
    local.courdinateX =COORDINATE_X;
    local.courdinateY =COORDINATE_Y;
    initializeAreaNodes(NODES_IN_THE_AREA);
    initializeMaps();


    // Beginning our initialization phases:
    if (call RadioControl.start() != SUCCESS)
      fatal_problem();

    if (call RoutingControl.start() != SUCCESS)
      fatal_problem();
         call UartControl.start();
     }

  event void RadioControl.startDone(error_t error) {
    if (error != SUCCESS)
      fatal_problem();

    if (sizeof(local) > call Send.maxPayloadLength())
      fatal_problem();

   if (call SerialControl.start() != SUCCESS)
      fatal_problem();
  }

  event void SerialControl.startDone(error_t error) {
    if (error != SUCCESS)
      fatal_problem();

  /*****************************************************************/
  /* This is how to set yourself as a root to the collection layer:*/
  /*****************************************************************/
```

```
    if (local.id == 0)
        call RootControl.setRoot();
      startTimer();
     }

  /*****************************************************************/
  /*   Start of the first timer that starts the periodical timer  */
  /*****************************************************************/
  static void startTimer() {
    if (call Timer.isRunning()) call Timer.stop();
    call Timer.startPeriodic(local.interval);
    reading = 0;
  }

  event void RadioControl.stopDone(error_t error) {}
  event void SerialControl.stopDone(error_t error) {}


/*
 *Only the root -SINK  will receive messages from this interface; its job
is to get all the coordinates of the nodes in the network, estimate the
coverage hole and find the optimal position for the deployment of a new
node to minimize the coverage holes
*
*/

  event message_t*
  Receive.receive(message_t* msg, void *payload, uint8_t len) {
    oscilloscope_t* in = (oscilloscope_t*)payload;
    oscilloscope_t* out;

     if (uartbusy == FALSE) {
      out = (oscilloscope_t*)call SerialSend.getPayload(&uartbuf,
sizeof(oscilloscope_t));
      if (len != sizeof(oscilloscope_t) || out == NULL) {
        return msg;
      }
      else {
          memcpy(out, in, sizeof(oscilloscope_t));
      }
      uartlen = sizeof(oscilloscope_t);
      post uartSendTask();
    } else {
      // The UART is busy; queue up messages and service them when
the
      // UART becomes free.
      message_t *newmsg = call UARTMessagePool.get();
```

```
      if (newmsg == NULL) {
        // drop the message on the floor if we run out of queue space.
        report_problem();
        return msg;
      }

      //Serial port busy, so enqueue.
      out = (oscilloscope_t*)call SerialSend.getPayload(newmsg,
sizeof(oscilloscope_t));
      if (out == NULL) {
            return msg;
      }
      memcpy(out, in, sizeof(oscilloscope_t));

      if (call UARTQueue.enqueue(newmsg) != SUCCESS) {
        // drop the message on the floor and hang if we run out of
        // queue space without running out of queue space first (this
        // should not occur).
        call UARTMessagePool.put(newmsg);
        fatal_problem();
        return msg;
      }
    }
    return msg;
  }


  task void uartSendTask() {
   uint16_t  holeCoordinateX=0;
   uint16_t   holeCoordinateY=0;
   uint16_t result;
   uartbusy = TRUE;
   messageReceived = (oscilloscope_t*)call
SerialSend.getPayload(&uartbuf, sizeof(oscilloscope_t));
            if ( messageReceived == NULL) {
             fatal_problem();
            return ;
      }

           if (areaNodes[(messageReceived->id) -1]!=TRUE){


boebotSensorMessage.coordinatesX[areaNodesCounter]=messageReceiv
ed->courdinateX;

boebotSensorMessage.coordinatesY[areaNodesCounter]=messageReceive
d->courdinateY;
           areaNodes[(messageReceived->id) -1]=TRUE;
```

```
          result=drawNodeInMap(messageReceived->courdinateX,
messageReceived->courdinateY, 3 );
          areaNodesCounter++;
          }
          if (areaNodesCounter==NODES_IN_THE_AREA){
           if (found==FALSE){
            boebotSensorMessage.numberOfnodes=areaNodesCounter;
            call UartByte.send('f');
            findCoverageHoleCoordinates( &holeCoordinateX,
&holeCoordinateY, 3);
                boebotSensorMessage.destinationX=holeCoordinateX;
                boebotSensorMessage.destinationY=holeCoordinateY;

                  if (!sendbusy) {
             boebot_t *o = (boebot_t *)call
BoebotSend.getPayload(&sendboebotbuf, sizeof(boebot_t));
             if (o == NULL) {
               fatal_problem();
               return;
             }
              memcpy(o, &boebotSensorMessage,
sizeof(boebotSensorMessage));
              call UartByte.send('y');
                  /* send to mobile mote the coordinates for its
deployment*/
                  if (call BoebotSend.send(0xA,&sendboebotbuf,
sizeof(boebot_t)) == SUCCESS) {
                    sendbusy = TRUE;
        }
          }
          /*send to uard the  coordinates for debuging */
            call UartByte.send('0'+holeCoordinateX);
                call UartByte.send(',');
                call UartByte.send('0'+holeCoordinateY);
            call UartByte.send('\n');
                found=TRUE;
                }
          }
         uartbusy = FALSE;
    if (call UARTQueue.empty() == FALSE) {
      // We just finished a UART send, and the uart queue is
      // non-empty.  Let's start a new one.
      message_t *queuemsg = call UARTQueue.dequeue();
      if (queuemsg == NULL) {
        fatal_problem();
        return;
      }
      memcpy(&uartbuf, queuemsg, sizeof(message_t));
```

```
      if (call UARTMessagePool.put(queuemsg) != SUCCESS) {
        fatal_problem();
        return;
      }
      post uartSendTask();
    }

  }

  event void SerialSend.sendDone(message_t *msg, error_t error) {
  }

  //
  // Overhearing other traffic in the network.
  //
  event message_t*
  Snoop.receive(message_t* msg, void* payload, uint8_t len) {
    oscilloscope_t *omsg = payload;
    report_received();
    // If we receive a newer version, update our interval.
    if (omsg->version > local.version) {
      local.version = omsg->version;
      local.interval = omsg->interval;
      startTimer();
    }
    // If we hear from a future count, jump ahead but suppress our own
    // change.
    if (omsg->count > local.count) {
      local.count = omsg->count;
      suppress_count_change = TRUE;
    }
    return msg;
  }

  /*  Code for the motes of the network to send periotically their
coordinates to the sink */
  event void Timer.fired() {
      if (local.id != 0){
      if (!sendbusy) {
            oscilloscope_t *o = (oscilloscope_t *)call
Send.getPayload(&sendbuf, sizeof(oscilloscope_t));
            if (o == NULL) {
              fatal_problem();
              return;
            }
            memcpy(o, &local, sizeof(local));
            if (call Send.send(&sendbuf, sizeof(local)) == SUCCESS)
              sendbusy = TRUE;
```

```
        else
          report_problem();
        }
        reading = 0;
       /* Part 2 of cheap "time sync": increment our count if we didn't
jump ahead. */
      if (!suppress_count_change)
        local.count++;
      suppress_count_change = FALSE;
    }
  }

  /* event  to check if the message is send succesfully*/
  event void Send.sendDone(message_t* msg, error_t error) {
    if (error == SUCCESS){
      report_sent();
   } else
      report_problem();
    sendbusy = FALSE;
  }



  // Use LEDs to report various status issues.
  static void fatal_problem() {
    call Leds.led0On();
    call Leds.led1On();
    call Leds.led2On();
    call Timer.stop();
  }

  // Use LEDs to report various status issues.
  static void report_problem() { call Leds.led0Toggle(); }
  static void report_sent() { call Leds.led1Toggle(); }
  static void report_received() { call Leds.led2Toggle(); }


  /*****************************************************************/
   /*initialize the AreaNodes array with FALSE*/
    void initializeAreaNodes( uint16_t size){
      uint8_t index;
      for (index =0; index< size; index++)
          {
       areaNodes[index]=FALSE;
      }
    }
/*****************************************************************/
    //initialize the Map array with FALSE
```

```
        void initializeMaps(){
          uint8_t index_X;
          uint8_t index_Y;
      for (index_X =0; index_X< AREA_X; index_X++)
          {
            for (index_Y =0; index_Y< AREA_Y; index_Y++)
            {
             Map[index_X][index_Y]=FALSE;
       }
     }
        }
   /*****************************************************************/
  uint16_t drawNodeInMap(uint16_t x, uint16_t y, uint16_t radius ){
         int16_t i ,j,end1,end2,count = 0 ;
         int16_t temporary1,temporary2,temporary3;
         double distance, temp;

          i=(x+radius);
          end1=(x-radius);
               while (i >= end1){
            j=(y+radius);
                 end2=(y-radius);
                 while (j >= end2){
            if ((i < 0)||(j < 0)||(i >= AREA_X)||(j >= AREA_Y)){
                       j--;
                              continue;
                     }
                         temporary1=(x-i);
                         temporary1=(temporary1*temporary1);
                         temporary2=(y-j);
                         temporary2=(temporary2*temporary2);
                         temporary3=temporary1 + temporary2;
                         distance= sqrt(temporary3);
                         temp=radius-distance;
                  if  (temp>=0){
                        count++;
                            Map[i][j]=TRUE;
                     }
                             j--;
                      }

                i--;
         }
          return count;
        }
  /*****************************************************************/
  /* Function that finds the coverage hole and estimates the optimal
position for the deployment of a new sensor mote                  */
```

```
   uint16_t findCoverageHoleCoordinates( uint16_t * holeCoordinateX,
uint16_t *holeCoordinateY, uint16_t radius){

        uint16_t index_X,index_Y,maxX=0,maxY=0;
        int16_t i ,j,end1,end2,count = 0, maxCount=0 ;
        int16_t temporary1,temporary2,temporary3;
        double distance, temp;

      for(index_X=0; index_X<AREA_X;index_X++ ) {
        for(index_Y=0; index_Y<AREA_Y;index_Y++ ) {
          if (Map[index_X][index_Y]==FALSE){
            count = 0 ;
            i=(index_X+radius);
                end1=(index_X-radius);
                while (i >= end1){
                        j=(index_Y+radius);
                   end2=(index_Y-radius);
                  while (j >= end2){
             if ((i < 0)||(j < 0)||(i >= AREA_X)||(j >= AREA_Y)){
                        j--;
                            continue;
                    }
                        temporary1=(index_X-i);
                        temporary1=(temporary1*temporary1);
                        temporary2=(index_Y-j);
                        temporary2=(temporary2*temporary2);
                        temporary3=temporary1 + temporary2;
                        distance= sqrt(temporary3);
                        temp=radius-distance;
                if  (temp>=0){
                              if (Map[i][j]==FALSE){
                      count++;
                    }
                        }
                        j--;
                    }
                    i--;
                }
          }

          //find the bigger hole and keep its coordinates
         if (maxCount < count){
           maxX = index_X;
           maxY = index_Y;
       maxCount=count;
         }

        }
```

```
        }
        *holeCoordinateX =maxX;
        *holeCoordinateY =maxY;
        return 1;
        }
/*****************************************************************/
/* event  to check if the message is send succesfully to the mobile
mote*/
        event void BoebotSend.sendDone(message_t* msg, error_t error) {
    if (error == SUCCESS){
      report_sent();
    }else{
      report_problem();
}
    sendbusy = FALSE;
  }

}
```

MultihopOscilloscope.h

```
//////////////////////////////////////////////////////////////////
//                                                              //
// The library with all the constants needed                    //
//                                                              //
// Author: Constantinos Constantinides                          //
// Date last modified:  11/04/09                                //
//                                                              //
//////////////////////////////////////////////////////////////////

#ifndef MULTIHOP_OSCILLOSCOPE_H
#define MULTIHOP_OSCILLOSCOPE_H

enum {
  /* Number of readings per message. If you increase this, you may have
to increase the message_t size. */
  NREADINGS = 5,
  /* Default sampling period. */
  DEFAULT_INTERVAL = 5024,
  AM_OSCILLOSCOPE = 0x93,
  AM_BOE_BOT_MSG=17,
  COORDINATE_X= 8,
  COORDINATE_Y= 1,
  NODES_IN_THE_AREA=5,
  AREA_X=18,
  AREA_Y=12
};
```

```
typedef nx_struct oscilloscope {
  nx_uint16_t version; /* Version of the interval. */
  nx_uint16_t interval; /* Samping period. */
  nx_uint16_t id; /* Mote id of sending mote. */
  nx_uint16_t count; /* The readings are samples count * NREADINGS
onwards */
  //sensor node coordinates
  nx_uint16_t courdinateX;
  nx_uint16_t courdinateY;
} oscilloscope_t;

typedef nx_struct boebot {
  nx_uint16_t numberOfnodes;
  nx_uint16_t  coordinatesX[NODES_IN_THE_AREA];
  nx_uint16_t  coordinatesY[NODES_IN_THE_AREA];
  nx_uint16_t destinationX;
  nx_uint16_t destinationY;
} boebot_t;




#endif
```

Makefile

```
COMPONENT=SendToBoebotAppC
CFLAGS += -I$(TOSDIR)/lib/net/ -I$(TOSDIR)/lib/net/ctp
-I$(TOSDIR)/lib/net/4bitle

#CFLAGS += "-DCC2420_DEF_RFPOWER=1"
LIBS = -lm

include $(MAKERULES)
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκε ο κινητός κόμβος ο οποίος είναι ενσωματωμένος πάνω στο boebot.

Boebot_Go_AppC.nc

```
//////////////////////////////////////////////////////////
//  This  application is for the micaz mode that would be attached    //
//  on the boebot. The mote would receive the destination             //
//  coordinates and the position of the other motes in the network  //
//  from the sink and then it would estimate the path avoiding        //
//  collision to any of the motes. The mote would then drive the      //
//  boebot to the destination using uart communication.               //
//                                                                    //
// Author: Constantinos Constantinides                                //
```

```
// Date last modified:  11/04/09                                        //
//                                                                      //
//////////////////////////////////////////////////////////////////////////

#include <Timer.h>
#include "Boebot_Go_C.h"
#include "MultihopOscilloscope.h"


configuration Boebot_Go_AppC {
}
implementation {
   /* The components used*/
  components MainC;
  components LedsC;
  components PlatformSerialC;
  components Boebot_Go_C as App;
  components new TimerMilliC() as Timer1;
  components ActiveMessageC;
  components new AMSenderC(AM_BOE_BOT_MSG);
  components new AMReceiverC(AM_BOE_BOT_MSG);

  /*Basic components*/
  App.Boot -> MainC;
  App.Timer1 -> Timer1;

  /* LEDS*/
   App.Leds -> LedsC;

  /*RF communication*/
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;

  /*Uart communication*/
  App.UartControl -> PlatformSerialC;
  App.UartByte    -> PlatformSerialC;

}
```

Boebot_Go_C.nc

```
//////////////////////////////////////////////////////////////////////
// This  application is for the micaz mode that would be attached   //
// on the boebot. The mote would receive the destination coordinates//
// and the position of the other motes in the network from the sink  //
```

```
// and then it would estimate the path avoiding collision to any of    //
// the motes. The mote would then drive the boebot to the              //
// destination using uart communication.                               //
//                                                                     //
// Author: Constantinos Constantinides                                 //
// Date last modified:  11/04/09                                       //
//                                                                     //
/////////////////////////////////////////////////////////////////////////

#include <Timer.h>
#include "Boebot_Go_C.h"
#include "MultihopOscilloscope.h"


module Boebot_Go_C {
/*interfaces used*/

  /*Basic interfaces*/
  uses interface Boot;
  uses interface Timer<TMilli> as Timer1;

  /*LEDS interface*/
  uses interface Leds;

  /*RF communication interface*/
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;

  /* UART communication interfaces*/
  uses interface StdControl as UartControl;
  uses interface UartByte;
}

implementation {


/*****************************************************************/
/* prototypes of the program */
  void initializeMaps();
  void addNodeToPathMap(uint16_t , uint16_t);
  void SendToBoeBot(uint16_t , uint16_t );
  bool moveX_Y( uint16_t ,uint16_t ,uint16_t ,uint16_t );
  bool moveY_X( uint16_t ,uint16_t ,uint16_t ,uint16_t );

  void boebotStart();
```

```
  void boebotMove(uint16_t, uint16_t );
  void turnToXmax( uint16_t*  );
  void turnToXmin( uint16_t* );
  void turnToYmax( uint16_t* );
  void turnToYmin( uint16_t* );
  void pathfinder(uint16_t,uint16_t );
  void waitfor();



/*****************************************************************/
   /* Global varioables*/
  uint16_t  Boebotcommands[100];
  uint16_t index=0;
  uint16_t indexCommand=0;
  uint16_t counter;
  message_t pkt;
  bool busy = FALSE;
  bool PathMap[AREA_X][AREA_Y];


  uint16_t instructionCounter=0;

  /*Function that shows the output of the boebot command on the
leds*/
  void setLeds(uint16_t val) {
    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04){
      call Leds.led2On();
    }else
      call Leds.led2Off();
  }
/*****************************************************/
/*************  program initializations ****************/
/*****************************************************/
  event void Boot.booted() {
    call UartControl.start();
    call AMControl.start();
    initializeMaps();
  }
```

```
  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {

    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }


/*****************************************************************/
/*The commands to the boebot are send one by one each time the timer
fires                                                            */
/*****************************************************************/
   event void Timer1.fired() {
        /* local variables*/
        uint16_t i=0;
        uint16_t counterIn=0;
        uint16_t number=0;
        uint16_t times=0;
        uint16_t flag =0;
   if (indexCommand<index){

/*if the number is greater than 9, the digits of the numbers  are send
separately one by one*/
   while ( Boebotcommands[indexCommand] >= 10 ){
        number = (Boebotcommands[indexCommand] );
        counterIn=0;
        while (number >= 10){
          counterIn++;
              if ((number % 10 == 0) &&(counterIn !=1)){
                flag =1;
              }
              number= number / 10;
        }
        times =1;
        for (i=0; i < counterIn; i++ ){ times=times *10;}
/* the digits of the numbers greater than ten are send separately*/
        call UartByte.send('0'+number);
        Boebotcommands[indexCommand]
=Boebotcommands[indexCommand]  - (number * times);
      }
      if (flag ==1) {
            call UartByte.send('0');
          }
```

```
            /* The commands are send to the uart */
      call UartByte.send('0'+Boebotcommands[indexCommand] );
      call UartByte.send('\n');
    }
    /*go to the next command*/
    indexCommand++;
    }

  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }


/************************************************************/
/*Function that receives the packets sent by the sink  and makes the
estimation of the path                                      */
/************************************************************/
  event message_t* Receive.receive(message_t* msg, void* payload,
uint8_t len){
      uint16_t i=0;
           boebot_t* btrpkt = (boebot_t*)payload;
/*if packet from the sink is received all the leds of the mote are on*/
           call Leds.led1On();
      call Leds.led2On();
      call Leds.led0On();


           /*Creation of area map with objects*/
           for (i=0;i<btrpkt->numberOfnodes;i++){
                addNodeToPathMap(btrpkt->coordinatesX[i], btrpkt-
>coordinatesY[i]);
           }

/* call the function that finds the path and creates the commands to be
send to the boebot*/
           pathfinder( btrpkt->destinationX, btrpkt->destinationY);
/* Start the timer so that the commands would be send to the boebot*/
           call Timer1.startPeriodic(TIMER_PERIOD_MILLI);

          return msg;

    }


/************************************************************/
//Function that initialize the PathMap array with TRUE
```

```
        void initializeMaps(){
          uint8_t index_X;
          uint8_t index_Y;
      for (index_X =0; index_X< AREA_X; index_X++)
          {
            for (index_Y =0; index_Y< AREA_Y; index_Y++)
            {
                  PathMap[index_X][index_Y]=TRUE;
      }
    }
      }

/*************************************************************/
  // Function that adds a node in the Path Map
  void addNodeToPathMap(uint16_t X , uint16_t Y)
       {int16_t i, Xend;
         uint16_t j, Yend;
         i=X-1;
         Xend=X+1;
         Yend=Y+1;

       for (;i<=Xend;i++){
         j=Y-1;
         for (;j<=Yend;j++){
          PathMap[i][j]=FALSE;
       }
       }
}
/*************************************************************/
// Function that fills the Boebotcommands array with the instructions
  void SendToBoeBot(uint16_t direction, uint16_t howMuch){
                instructionCounter++;
/*every 9 instructions, the next instruction orders the boebot to start
executing the commands        because the memory of the boebot can
save up to ten instructions */
        if (instructionCounter==10){
          Boebotcommands[index]=5;
          index++;
      Boebotcommands[index]=0;
          index++;

        instructionCounter=0;
        }else{
        Boebotcommands[index]=direction;
                index++;
         if ((direction==1)||(direction==2)){
   /* one step in the Path Map equals 5  in the  real map*/
          howMuch=howMuch * 5;
```

```
            }
            Boebotcommands[index]=howMuch;
            index++;
        }
       }
/*************************************************************/
  // Function that orders the boebot to turn towards Ymin
  void turnToYmin( uint16_t *moveForward ){
  if (*moveForward == 1){
  /* Right -90 */
   atomic{ SendToBoeBot(3,90);}
  }

  if (*moveForward == 2){
    /*Right -90 */
    atomic{SendToBoeBot(3,90);}
         /* Right -90 */
         atomic{SendToBoeBot(3,90);}
  }

  if (*moveForward == 3){
   /*Left -90 */
   atomic{ SendToBoeBot(4,90);}
  }

  if (*moveForward == 4){
    //Do nothing
  }
  // Now the boebot looks towards Ymin
  *moveForward =4;
}

// Function that orders the boebot to turn towards Ymax
void turnToYmax( uint16_t *moveForward ){
  if (*moveForward == 1){
  /* Left -90 */
    atomic{SendToBoeBot(4,90);}
  }

  if (*moveForward == 2){
  // Do nothing
  }

  if (*moveForward == 3){
  /* Right -90 */
    atomic{SendToBoeBot(3,90);}
  }
```

```
  if (*moveForward == 4){
  /* Right -90 */
   atomic{ SendToBoeBot(3,90);}
        /* Right -90 */
        atomic {SendToBoeBot(3,90);}
  }
  // Now the boebot looks towards Ymax
  *moveForward =2;
}

// Function that orders the boebot to turn towards Xmin
void turnToXmin( uint16_t *moveForward ){
  if (*moveForward == 1){
  /* Right -90 */
    atomic{SendToBoeBot(3,90);}
        /* Right -90 */
        atomic{SendToBoeBot(3,90);}
  }

  if (*moveForward == 2){
  /* Left -90 */
        atomic {SendToBoeBot(4,90);}
  }

  if (*moveForward == 3){
    //Do nothing
  }

  if (*moveForward == 4){
  /* Right -90 */
    atomic{SendToBoeBot(3,90);}
  }
    // Now the boebot looks towards Xmin
    *moveForward =3;
}

// Function that orders the boebot to turn towards Xmax
void turnToXmax( uint16_t *moveForward ){
  if (*moveForward == 1){
    //do nothing
  }

  if (*moveForward == 2){
  /* Right -90 */
        atomic{SendToBoeBot(3,90);}
  }
```

```
  if (*moveForward == 3){
  /* Right -90 */
    atomic {SendToBoeBot(3,90);}
         /* Right -90 */
         atomic{SendToBoeBot(3,90);}

  }

  if (*moveForward == 4){
  /* Right -90 */
    atomic{SendToBoeBot(4,90);}
  }
      // Now the boebot looks towards Xmax
      *moveForward =1;
}

// function that moves the boebot front and backwards
void boebotMove(uint16_t direction, uint16_t steps){
  atomic{SendToBoeBot(direction,steps);}
}

/* Function that send a start command to the BoeBot*/
void boebotStart(){
/*  start moving command  */
atomic {SendToBoeBot(5,0);}

}
/*************************************************************/
/* Function that checks if there is a path to the destination if the
boebot moves first towards axis Y and then towards axis X*/
 bool moveY_X( uint16_t Xstart,uint16_t Ystart,uint16_t Xend,uint16_t
Yend){
uint16_t i;
 if (Ystart <=Yend) {
  for (i=Ystart; i<=Yend; i++){
   if (PathMap[Xstart][i]==FALSE){
    return FALSE;
   }
  }
}else{
  for (i=Ystart; i>=Yend; i--){
   if (PathMap[Xstart][i]==FALSE){
    return FALSE;
   }
  }
}

if (Xstart <=Xend) {
```

```
  for (i=Xstart; i<=Xend; i++){
   if (PathMap[i][Yend]==FALSE){
    return FALSE;
   }
  }
}else{
 for (i=Xstart; i>=Xend; i--){
   if (PathMap[i][Yend]==FALSE){
    return FALSE;
   }
  }
}
  return TRUE;

}

 /* Function that checks if there is a path to the destination if the
boebot moves first towards axis X and then towards axis Y*/
bool moveX_Y( uint16_t Xstart,uint16_t Ystart,uint16_t Xend,uint16_t
Yend){
uint16_t i;
 if (Xstart <=Xend) {
  for (i=Xstart; i<=Xend; i++){
   if (PathMap[i][Ystart]==FALSE){
    return FALSE;
   }
  }
  }else{
   for (i=Xstart; i>=Xend; i--){
    if (PathMap[i][Ystart]==FALSE){
    return FALSE;
    }
   }
  }
if (Ystart <=Yend) {
  for (i=Ystart; i<=Yend; i++){
   if (PathMap[Xend][i]==FALSE){
    return FALSE;
   }
  }
}else{
for (i=Ystart; i>=Yend; i--){
   if (PathMap[Xend][i]==FALSE){
    return FALSE;
   }
  }
}
```

```
return TRUE;
}
/*************************************************************/
/*Function that finds the path towards the testination estimating the
commands to be send to the boebot*/
   void pathfinder(uint16_t Xfinal,uint16_t Yfinal){
  //initialization of the boebot possision in the path map
  int16_t Xcurrent=0;
  int16_t Ycurrent=0;
  uint16_t moveForward=1;
  uint16_t steps=0;
  uint16_t countSteps=0;
  bool moved=FALSE;
/* The code below is executed until the boebot move to the destination,
and this is when one of the functions  moveX_Y  or  moveY_X returns
TRUE*/
  while((Xcurrent!=Xfinal)||(Ycurrent!=Yfinal)){

  if (moveY_X(Xcurrent,Ycurrent,Xfinal,Yfinal) ==TRUE){
     /*  move instructions  of moveY_X*/
   if (countSteps>0){
     boebotMove(1, countSteps);
   }
          if (Ycurrent < Yfinal){
          //1. turn towards  Ymax
          //2. move  (Yfinal-Ycurrent) steps forward
           turnToYmax( &moveForward );
           steps=(Yfinal-Ycurrent);
           boebotMove(1, steps);


          }else{
          //1.  turn towards Ymin
          //2. move  (Ycurrent -Yfinal) steps forward
           turnToYmin( &moveForward );
           steps=(Ycurrent -Yfinal);
           boebotMove(1, steps);
          }
          if (Xcurrent < Xfinal){
          //3. turn towards  Xmax
          //4.move  (Xfinal-Xcurrent) steps  forward
           turnToXmax( &moveForward );
           steps=(Xfinal-Xcurrent) ;
           boebotMove(1, steps);
          }else{
          //3. sturn towards  Xmin
          //4. move  (Xcurrent -Xfinal) steps forward
           turnToXmin( &moveForward );
```

```
          steps=(Xcurrent -Xfinal)  ;
          boebotMove(1, steps);
         }

         //Xcurrent=Xfinal;
         //Ycurrent=Yfinal;

         //send "over " to boebot to start moving
         boebotStart();
    break;

}else{
  if (moveX_Y(Xcurrent,Ycurrent,Xfinal,Yfinal) ==TRUE){

        if (countSteps>0){
   boebotMove(1, countSteps);
 }

 /*  move instructions of moveX_Y*/
       if (Xcurrent < Xfinal){
         //1. turn towards  Xmax
         //2. move  (Xfinal-Xcurrent) steps forward
          turnToXmax( &moveForward );
          steps=(Xfinal-Xcurrent);
          boebotMove(1, steps);
         }else{
         //1. turn towards Xmin
         //2. move (Xcurrent -Xfinal) steps forward
          turnToXmin(&moveForward );
          steps=(Xcurrent -Xfinal);
          boebotMove(1, steps);
         }

         if (Ycurrent < Yfinal){
         //3. turn towards  Ymax
         //4.move  (Xfinal-Xcurrent) steps forward
          turnToYmax( &moveForward );
          steps=(Yfinal-Ycurrent);
          boebotMove(1, steps);
         }else{
         //3. turn towards Ymin
         //4. move  (Xcurrent -Xfinal) steps forward
          turnToYmin( &moveForward );
          steps=(Ycurrent -Yfinal);
          boebotMove(1, steps);
         }
         //Xcurrent=Xfinal;
         //Ycurrent=Yfinal;
```

```
    //send "over " to boebot to start moving
boebotStart();
break;

  }else{

    while (moved == FALSE)
   {

     //try and moveRight towards y-min
    if (moveForward==1){  /////////// turn right
     if (Ycurrent-1>=0){
      if (PathMap[Xcurrent][Ycurrent-1]==TRUE){
      // turn Right
      //move one step forward
      boebotMove(1, countSteps);
      countSteps=0;
      atomic{ SendToBoeBot(3,90);}
      countSteps++;

       Ycurrent--;
       moved=TRUE;
  moveForward=4;
  break;
      }
}
    }

     //try and moveRight towards x-min
    if (moveForward==4){  /////////// turn right
     if (Xcurrent-1>=0){
      if (PathMap[Xcurrent-1][Ycurrent]==TRUE){
      // turn Right
      //move one step forward
      boebotMove(1, countSteps);
      countSteps=0;
      atomic {SendToBoeBot(3,90);}
      countSteps++;

       Xcurrent--;
       moved=TRUE;
  moveForward=3;
  break;
      }
}
    }
```

```
        //try and moveRight towards Y-max
        if (moveForward==3){   /////////// turn right
        if (Ycurrent+1 < AREA_Y){
         if (PathMap[Xcurrent][Ycurrent+1]==TRUE){
         // turn Right
          //move one step forward
          boebotMove(1, countSteps);
          countSteps=0;
          atomic {SendToBoeBot(3,90);}
          countSteps++;

          Ycurrent++;
          moved=TRUE;
      moveForward=4;
      break;
         }
    }
       }
        //try and moveRight towards X-max
        if (moveForward==2){ ////////////////turn right
         if (Xcurrent+1 < AREA_X){
          if (PathMap[Xcurrent+1][Ycurrent]==TRUE){
          // turn Right
          //move one step forward
          call UartByte.send('x');
          boebotMove(1, countSteps);
          countSteps=0;
          atomic {SendToBoeBot(3,90);}
          countSteps++;
           Xcurrent++;
           moved=TRUE;
      moveForward=1;
      break;
         }
     }
       }

        // move forward towards x-max else turn left
        if (moveForward ==1){
              if (Xcurrent+1 < AREA_X){
           if (PathMap[Xcurrent+1][Ycurrent]==TRUE){
               //move one step forward
                countSteps++;
        Xcurrent++;
        moved=TRUE;
          } else{
               //turn left
               boebotMove(1, countSteps);
```

```
        countSteps=0;
        atomic{ SendToBoeBot(4,90);}
            moveForward = 2;
           }
      }else{
            //turn left
            boebotMove(1, countSteps);
        countSteps=0;
        atomic{ SendToBoeBot(4,90);}
            moveForward = 2;

      }
    }


    // move forward towards y-max else turn left
          if (moveForward ==2){
       if (Ycurrent+1 < AREA_Y){
        if (PathMap[Xcurrent][Ycurrent+1]==TRUE){
            //move one step forward
              countSteps++;
     Ycurrent++;
     moved=TRUE;
       }else{
            //turn left
            boebotMove(1, countSteps);
        countSteps=0;
        atomic{ SendToBoeBot(4,90); }
            moveForward = 3;
            }
           }else{
            //turn left
            boebotMove(1, countSteps);
        countSteps=0;
       atomic{ SendToBoeBot(4,90);}
            moveForward = 3; }
    }

     // move forward towards x-min else turn left
          if (moveForward ==3){
           if (Xcurrent-1 >= 0){
       if (PathMap[Xcurrent-1][Ycurrent]==TRUE){
            //move one step forward
              countSteps++;
     Xcurrent--;
     moved=TRUE;
       }else{
            //turn left
```

```
                boebotMove(1, countSteps);
            countSteps=0;
            atomic{ SendToBoeBot(4,90);}
            moveForward = 4;
                 }
                } else{
                //turn left
                boebotMove(1, countSteps);
            countSteps=0;
            atomic{ SendToBoeBot(4,90);}
                moveForward = 4; }
               }

           // move forward towards y-min else turn left
               if (moveForward ==4){
                 if (Ycurrent-1 >= 0){
              if (PathMap[Xcurrent][Ycurrent-1]==TRUE){
                    //move one step forward
                    countSteps++;
           Ycurrent--;
           moved=TRUE;
              }else{
                    //turn left
                    boebotMove(1, countSteps);
               countSteps=0;
               atomic{SendToBoeBot(4,90);}
                        moveForward = 1;
                    }
                  }else{
                    //turn left
                    boebotMove(1, countSteps);
               countSteps=0;
               atomic {SendToBoeBot(4,90);}
                    moveForward = 1; }
           }
        }
         moved=FALSE;
   }
  }
}
}
/*************************************************************/
}
```

Boebot_Go_C.h

```
////////////////////////////////////////////////////////////
// The library with all the constants needed              //
```

```
//                                                                      //
//                                                                      //
// Author: Constantinos Constantinides                                 //
// Date last modified:  11/04/09                                        //
//                                                                      //
//////////////////////////////////////////////////////////////////////////

#ifndef BOEBOT_GO_C_H
#define BOEBOT_GO_C_H
enum {
  AM_BLINKTORADIO = 6,
  TIMER_PERIOD_MILLI = 2000
};
typedef nx_struct BlinkToRadioMsg {
  nx_uint16_t nodeid;
  nx_uint16_t counter;
} BlinkToRadioMsg;
#endif
```

Makefile

```
COMPONENT=Boebot_Go_AppC
include $(MAKERULES)
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκαν οι τρείς κόμβοι αισθητήρων για την επικοινωνία τους με τη χρήση του πρωτοκόλλου IEEE 802.15.4 (zigbee)

SimpleRoutingExample.nc

```
//////////////////////////////////////////////////////////////////////
//    This simple application  demonstrates how to route data       //
//     messages in cluster tree topolog. ach device sends data      //
//     messages to the PAN coordinator with the first 2 bytes of the//
//     data payload as a routing field with the destination address of//
//     the desired device. when the PAN coordinator receives the data//
//     frame, it will read the payload and create a new data frame  //
//     with the destination source address located in the 2 bytes of//
//     the data payload. The coordinator will insert in the data    //
//     payload the number of the routed packets.                    //
//                                                                  //
//                                                                  //
//  Modified by: Constantinos Constantinides                        //
//                                                                  //
//////////////////////////////////////////////////////////////////////
#include <Timer.h>
```

```
#include "simpleroutingexample.h"
#include "phy_const.h"
#include "phy_enumerations.h"
#include "mac_const.h"
#include "mac_enumerations.h"
#include "mac_func.h"

 configuration SimpleRoutingExample {
}

implementation
{
  components MainC;
  components LedsC;
  components SimpleRoutingExampleM;

  SimpleRoutingExampleM.Boot -> MainC;

  components Mac;

  SimpleRoutingExampleM.Leds -> LedsC;

  components new TimerMilliC() as Timer0;
  SimpleRoutingExampleM.Timer0 -> Timer0;

  components new TimerMilliC() as Timer_Send;
  SimpleRoutingExampleM.Timer_Send ->Timer_Send;


  //MAC interfaces

  SimpleRoutingExampleM.MLME_START -> Mac.MLME_START;

  SimpleRoutingExampleM.MLME_GET ->Mac.MLME_GET;
  SimpleRoutingExampleM.MLME_SET ->Mac.MLME_SET;

  SimpleRoutingExampleM.MLME_BEACON_NOTIFY -
>Mac.MLME_BEACON_NOTIFY;
  SimpleRoutingExampleM.MLME_GTS -> Mac.MLME_GTS;

  SimpleRoutingExampleM.MLME_ASSOCIATE-
>Mac.MLME_ASSOCIATE;
  SimpleRoutingExampleM.MLME_DISASSOCIATE-
>Mac.MLME_DISASSOCIATE;

  SimpleRoutingExampleM.MLME_ORPHAN->Mac.MLME_ORPHAN;
  SimpleRoutingExampleM.MLME_SYNC->Mac.MLME_SYNC;
  SimpleRoutingExampleM.MLME_SYNC_LOSS-
```

```
>Mac.MLME_SYNC_LOSS;
  SimpleRoutingExampleM.MLME_RESET->Mac.MLME_RESET;

  SimpleRoutingExampleM.MLME_SCAN->Mac.MLME_SCAN;


  SimpleRoutingExampleM.MCPS_DATA->Mac.MCPS_DATA;


}
```

SimpleRoutingExampleM.nc

```
//////////////////////////////////////////////////////////////
//    This simple application  demonstrates how to route data  //
//     messages in cluster tree topolog. ach device sends data  //
//     messages to the PAN coordinator with the first 2 bytes of the  //
//     data payload as a routing field with the destination address of  //
//     the desired device. when the PAN coordinator receives the data //
//     frame, it will read the payload and create a new data frame  //
//     with the destination source address located in the 2 bytes of  //
//     the data payload. The coordinator will insert in the data  //
//     payload the number of the routed packets.                  //
//                                                                //
//                                                                //
//  Modified by: Constantinos Constantinides                      //
//                                                                //
//////////////////////////////////////////////////////////////////

#include <Timer.h>
#include "printfUART.h"

module SimpleRoutingExampleM {

        uses interface Boot;
        uses interface Leds;
        uses interface Timer<TMilli> as Timer0;
        uses interface Timer<TMilli> as Timer_Send;
        //MAC interfaces
        uses interface MLME_START;
        uses interface MLME_GET;
        uses interface MLME_SET;
        uses interface MLME_BEACON_NOTIFY;
        uses interface MLME_GTS;
        uses interface MLME_ASSOCIATE;
        uses interface MLME_DISASSOCIATE;
```

```
        uses interface MLME_ORPHAN;
        uses interface MLME_SYNC;
        uses interface MLME_SYNC_LOSS;
        uses interface MLME_RESET;
        uses interface MLME_SCAN;
        uses interface MCPS_DATA;
}
implementation {

        PANDescriptor pan_des;
        uint32_t my_short_address=0x00000000;
        uint32_t DestinationMote[2];
        uint32_t SourceMoteAddr[2];

        //number of routed packet (coordinator)
        uint8_t routed_packets = 0x00;


  event void Boot.booted() {

        printfUART_init();
        printfUART("i_am_pan: %i\n", TYPE_DEVICE);
        DestinationMote[0]=0x00000000;
        DestinationMote[1]=0x00000002;
        if (TYPE_DEVICE == COORDINATOR)
        {
                my_short_address = 0x0000;
                call Timer0.startOneShot(4000);
        }
        else
        {
                call Timer0.startOneShot(4000);
        }

  }


  event void Timer0.fired() {
        uint8_t v_temp[2];
        if (TYPE_DEVICE == END_DEVICE)
        {

                my_short_address = TOS_NODE_ID;

                //set the MAC short address variable
                v_temp[0] = (uint8_t)(my_short_address >> 8);
                v_temp[1] = (uint8_t)(my_short_address );
```

```
                call MLME_SET.request(MACSHORTADDRESS,v_temp);

                //set the MAC PANID variable
                v_temp[0] = (uint8_t)(MAC_PANID >> 8);
                v_temp[1] = (uint8_t)(MAC_PANID );

                call MLME_SET.request(MACPANID,v_temp);

                call Timer_Send.startPeriodic(5000);


        }
        else
        {
                //set the MAC short address variable
                v_temp[0] = (uint8_t)(my_short_address >> 8);
                v_temp[1] = (uint8_t)(my_short_address );

                call MLME_SET.request(MACSHORTADDRESS,v_temp);

                //set the MAC PANID variable
                v_temp[0] = (uint8_t)(MAC_PANID >> 8);
                v_temp[1] = (uint8_t)(MAC_PANID );

                call MLME_SET.request(MACPANID,v_temp);

                //start sending beacons
                call MLME_START.request(MAC_PANID,
LOGICAL_CHANNEL, BEACON_ORDER,
SUPERFRAME_ORDER,1,0,0,0,0);

                //call Timer_send.start(TIMER_REPEAT,8000);
        }

  }

event void Timer_Send.fired() {


        uint8_t msdu_payload[4];

        DestinationMote[0]=0x00000000;
        DestinationMote[1]=0x00000000;

        //NKL destination address, coordinator will route packet to this
address
        msdu_payload[0] = 0x00;
        msdu_payload[1] = 0x03;
```

```
        SourceMoteAddr[0]=0x00000000;
        SourceMoteAddr[1]=TOS_NODE_ID;

        if (TOS_NODE_ID == 0x02)
        {
                call Leds.led1On();

                                                        //set_txoptions(ack,
gts, indirect_transmission, security)
                call MCPS_DATA.request(SHORT_ADDRESS, MAC_PANID,
SourceMoteAddr, SHORT_ADDRESS, MAC_PANID, DestinationMote, 2,
msdu_payload,1,set_txoptions(1,0,0,0));
        }


        }

/****************************************************************************/
/*****************************    MLME-SCAN    ****************************/
/****************************************************************************/
event error_t MLME_SCAN.confirm(uint8_t status,uint8_t ScanType,
uint32_t UnscannedChannels, uint8_t ResultListSize, uint8_t
EnergyDetectList[], SCAN_PANDescriptor PANDescriptorList[])
{

        return SUCCESS;
}

/****************************************************************************/
/************************    MLME-ORPHAN     ****************************/
/****************************************************************************/
event error_t MLME_ORPHAN.indication(uint32_t OrphanAddress[1],
uint8_t SecurityUse, uint8_t ACLEntry)
{

        return SUCCESS;
}
/****************************************************************************/
/************************    MLME-RESET    ****************************/
/****************************************************************************/
event error_t MLME_RESET.confirm(uint8_t status)
{



        return SUCCESS;
}
/****************************************************************************/
```

```
/*****************    MLME-SYNC-LOSS    ***************************/
/*************************************************************************/
event error_t MLME_SYNC_LOSS.indication(uint8_t LossReason)
{

        return SUCCESS;
}

/*************************************************************************/
/**************************      MLME-GTS     ************************/
/*************************************************************************/
event error_t MLME_GTS.confirm(uint8_t GTSCharacteristics, uint8_t
status)
{

        return SUCCESS;
}

event error_t MLME_GTS.indication(uint16_t DevAddress, uint8_t
GTSCharacteristics, bool SecurityUse, uint8_t ACLEntry)
{
        return SUCCESS;
}
/*************************************************************************/
/*********************    MLME-BEACON NOTIFY     ********************/
/*************************************************************************/
event error_t MLME_BEACON_NOTIFY.indication(uint8_t
BSN,PANDescriptor pan_descriptor, uint8_t PenAddrSpec, uint8_t
AddrList, uint8_t sduLength, uint8_t sdu[])
{

        return SUCCESS;
}
/*************************************************************************/
/************************     MLME-START     ************************/
/*************************************************************************/
event error_t MLME_START.confirm(uint8_t status)
{
   //set_txoptions(ack, gts, indirect_transmission, security)
return SUCCESS;
}
 /*************************************************************************/
/**************************  MLME-SET      ************************/
/*************************************************************************/

event error_t MLME_SET.confirm(uint8_t status,uint8_t PIBAttribute)
{
return SUCCESS;
```

```
}
/*****************************************************************************/
/************************    MLME-GET    *********************************/
/*****************************************************************************/
event error_t MLME_GET.confirm(uint8_t status,uint8_t PIBAttribute,
uint8_t PIBAttributeValue[])
{


return SUCCESS;
}


/*****************************************************************************/
/************************    MLME-ASSOCIATE    ***********************/
/*****************************************************************************/
event error_t MLME_ASSOCIATE.indication(uint32_t DeviceAddress[],
uint8_t CapabilityInformation, bool SecurityUse, uint8_t ACLEntry)
{

        return SUCCESS;
}

event error_t MLME_ASSOCIATE.confirm(uint16_t AssocShortAddress,
uint8_t status)
{

        return SUCCESS;
}
/*****************************************************************************/
/********************    MLME-DISASSOCIATE    ***********************/
/*****************************************************************************/
event error_t MLME_DISASSOCIATE.indication(uint32_t
DeviceAddress[], uint8_t DisassociateReason, bool SecurityUse, uint8_t
ACLEntry)
{
        return SUCCESS;
}

event error_t MLME_DISASSOCIATE.confirm(uint8_t status)
{
        return SUCCESS;
}
/*****************************************************************************/
/*************************    MCPS EVENTS    *************************/
/*****************************************************************************/

/*****************************************************************************/
```

```
/*************************     MCPS-DATA      ************************/
/*****************************************************************************/
event error_t MCPS_DATA.confirm(uint8_t msduHandle, uint8_t status)
{
call Leds.led1Off();

return SUCCESS;
}
event error_t MCPS_DATA.indication(uint16_t SrcAddrMode, uint16_t
SrcPANId, uint32_t SrcAddr[2], uint16_t DstAddrMode, uint16_t
DestPANId, uint32_t DstAddr[2], uint16_t msduLength,uint8_t
msdu[100],uint16_t mpduLinkQuality, uint16_t SecurityUse, uint16_t
ACLEntry)
{

//routing procedure, the short address of the destination device is the
first 2 bytes of the data payload
                if (TYPE_DEVICE == COORDINATOR)
                {
                        //route to the desired address
                        //call Leds.led0Toggle();
                        call Leds.led0Toggle();
                        DestinationMote[0]=0x00000000;
                        DestinationMote[1]=(uint32_t) msdu[1];

                        routed_packets++;
                        msdu[0] = routed_packets;


                                //set_txoptions(ack, gts, indirect_transmission,
security)
                        call MCPS_DATA.request(SHORT_ADDRESS,
MAC_PANID, SrcAddr, SHORT_ADDRESS, MAC_PANID,
DestinationMote, 1, msdu,1,set_txoptions(1,0,0,0));
                                                call Leds.led1On();
                }
                else
                {
                atomic{
                        call Leds.led0Toggle();
                        call Leds.led1Toggle();
                         printfUART("i_am_pan: %i\n", TYPE_DEVICE);

        }
                }
return SUCCESS;
}
}
```

simpleroutingexample.h

```
enum {
        COORDINATOR = 0x00,
        ROUTER =0x01,
        END_DEVICE = 0x02
        };

#define BEACON_ORDER 6
#define SUPERFRAME_ORDER 4
//the starting channel needs to be diferrent that the existent
coordinator operating channels
#define LOGICAL_CHANNEL 0x15


//#define TYPE_DEVICE END_DEVICE
#define TYPE_DEVICE COORDINATOR

//PAN VARIABLES
#define MAC_PANID 0x1234
```

Makefile

```
COMPONENT=SimpleRoutingExample

PFLAGS +=  -I$(TOSROOT)/tos/ieee802154/includes \
                        -I$(TOSROOT)/tos/ieee802154/mac \
                        -I$(TOSROOT)/tos/ieee802154/phy \
                        -I$(TOSROOT)/tos/ieee802154/timerasync \
                        -I$(TOSROOT)/tos/ieee802154/interfaces \
                        -I$(TOSROOT)/tos/ieee802154/interfaces/mac \
                        -I$(TOSROOT)/tos/ieee802154/interfaces/phy \
                        -I$(TOSROOT)/tos/chips/cc2420/ieee802154
include $(MAKERULES)
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκε ο κόμβος micaz για να μετακινεί remotely το boebot

BlinkToRadioAppC.nc

```
/*****************************************************/
/*
*    This application is the BlinkToRadio which is modified
*    to send over the RF the numbers 1-4
*
*     Modified by Constantinos Constantinides
*/
/*****************************************************/
#include <Timer.h>
```

```
#include "BlinkToRadio.h"

configuration BlinkToRadioAppC {
}
implementation {
  components MainC;
  components LedsC;
  components PlatformSerialC;
  components BlinkToRadioC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_BLINKTORADIO);
  components new AMReceiverC(AM_BLINKTORADIO);

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
  App.UartControl -> PlatformSerialC;
  App.UartByte    -> PlatformSerialC;
```

BlinkToRadioC.nc

```
/*****************************************************/
/*
*    This application is the BlinkToRadio which is modified
*    to send over the RF the numbers 1-4
*
*    Modified by Constantinos Constantinides
*/
/*****************************************************/

#include <Timer.h>
#include "BlinkToRadio.h"

module BlinkToRadioC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
```

```
  uses interface StdControl as UartControl;
  uses interface UartByte;

}
implementation {

  uint16_t counter;
  message_t pkt;
  bool busy = FALSE;


  void setLeds(uint16_t val) {

     call UartByte.send(('0'+val));
     call UartByte.send('\n');

    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04){
      call Leds.led2On();
    }else
      call Leds.led2Off();
  }

  event void Boot.booted() {
    call UartControl.start();
    call AMControl.start();
  }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer0.fired() {
```

```
    counter++;
    if (counter == 5){
     counter=1;
}
   // setLeds(counter);
    if (!busy) {
      BlinkToRadioMsg* btrpkt =
         (BlinkToRadioMsg*)(call Packet.getPayload(&pkt,
sizeof(BlinkToRadioMsg)));
      if (btrpkt == NULL) {
         return;
      }
      btrpkt->nodeid = TOS_NODE_ID;
      btrpkt->counter = counter;
      if (call AMSend.send(AM_BROADCAST_ADDR,
          &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
        busy = TRUE;
      }
    }
  }

  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload,
uint8_t len){
    if (len == sizeof(BlinkToRadioMsg)) {
      BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
      setLeds(btrpkt->counter);
    }
    return msg;
  }
}
}
```

BlinkToRadio.h

```
#ifndef BLINKTORADIO_H
#define BLINKTORADIO_H

enum {
  AM_BLINKTORADIO = 6,
  TIMER_PERIOD_MILLI = 2000
};
```

```
typedef nx_struct BlinkToRadioMsg {
  nx_uint16_t nodeid;
  nx_uint16_t counter;
} BlinkToRadioMsg;

#endif
```

makefile

```
COMPONENT=BlinkToRadioAppC
include $(MAKERULES)
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκε οπου είναι ενσωματωμένο το boebot

BlinkToRadioAppC.nc

```
/*****************************************************/
/*
*   This application is the BlinkToRadio which is modified
*   to receive from the RF the numbers 1-4 and send them over
*   the uart.
*
*   Modified by Constantinos Constantinides
*/
/*****************************************************/
#include <Timer.h>
#include "BlinkToRadio.h"
configuration BlinkToRadioAppC {
}
implementation {
  components MainC;
  components LedsC;
  components PlatformSerialC;
  components BlinkToRadioC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_BLINKTORADIO);
  components new AMReceiverC(AM_BLINKTORADIO);

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
```

```
  App.Receive -> AMReceiverC;
  App.UartControl -> PlatformSerialC;
  App.UartByte    -> PlatformSerialC;
}
```

BlinkToRadioC.nc

```
/*****************************************************/
/*
*   This application is the BlinkToRadio which is modified
*   to receive from the RF the numbers 1-4 and send them over
*   the uart.
*
*   Modified by Constantinos Constantinides
*/
/*****************************************************/
#include <Timer.h>
#include "BlinkToRadio.h"

module BlinkToRadioC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface Receive;
  uses interface SplitControl as AMControl;
  uses interface StdControl as UartControl;
  uses interface UartByte;

}
implementation {

  uint16_t counter;
  message_t pkt;
  bool busy = FALSE;


  void setLeds(uint16_t val) {

     call UartByte.send(('0'+val));
     call UartByte.send('\n');

    if (val & 0x01)
      call Leds.led0On();
    else
      call Leds.led0Off();
    if (val & 0x02)
```

```
      call Leds.led1On();
    else
      call Leds.led1Off();
    if (val & 0x04){
      call Leds.led2On();
    }else
      call Leds.led2Off();
  }

  event void Boot.booted() {
    call UartControl.start();
    call AMControl.start();
  }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call Timer0.startPeriodic(TIMER_PERIOD_MILLI);
    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
  }

  event void Timer0.fired() {
    counter++;
    if (counter == 5){
     counter=1;
}
   // setLeds(counter);
    if (!busy) {
      BlinkToRadioMsg* btrpkt =
         (BlinkToRadioMsg*)(call Packet.getPayload(&pkt,
sizeof(BlinkToRadioMsg)));
      if (btrpkt == NULL) {
        return;
      }
      btrpkt->nodeid = TOS_NODE_ID;
      btrpkt->counter = counter;
      /*if (call AMSend.send(AM_BROADCAST_ADDR,
          &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
        busy = TRUE;
      }*/
    }
  }
```

```
  event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {
      busy = FALSE;
    }
  }

  event message_t* Receive.receive(message_t* msg, void* payload,
uint8_t len){
    if (len == sizeof(BlinkToRadioMsg)) {
      BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
      setLeds(btrpkt->counter);
    }
    return msg;
  }
}
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκαν οι κόμβος telosb για την multihop επικοινωνία με τη χρήση του πρωτοκόλλου CTP

MultihopOscilloscopeAppC.nc

```
/*********************************************/
/*
* This application is the MultihopOscilloscope
* modified to get the temperature and send it to
* the root of the tree (telosb motes)
*
* Modified by Constantinos Constantinides
*/
/*********************************************/
configuration MultihopOscilloscopeAppC {}
implementation {
  components MainC, MultihopOscilloscopeC, LedsC, new TimerMilliC(),
new SensirionSht11C();
    //new DemoSensorC() as Sensor;
    //MainC.SoftwareInit -> Sensor;

    // Changed by NXT team
    // We commented out the demoSensorC component and we replace
it with SensirionSht11C()
    // to read the temperature from the sensor board


  MultihopOscilloscopeC.Boot -> MainC;
  MultihopOscilloscopeC.Timer -> TimerMilliC;

   // Changed by NXT team
   // MultihopOscilloscopeC.Read -> Sensor;
```

```
  // We connnect the SensirionSht11C.Temperature to the
MultihopOscilloscopeC.Read

  MultihopOscilloscopeC.Read -> SensirionSht11C.Temperature;
  MultihopOscilloscopeC.Leds -> LedsC;

  //
  // Communication components.  These are documented in TEP 113:
  // Serial Communication, and TEP 119: Collection.
  //
  components CollectionC as Collector,  // Collection layer
    ActiveMessageC,                     // AM layer
    new CollectionSenderC(AM_OSCILLOSCOPE), // Sends multihop RF
    SerialActiveMessageC,               // Serial messaging
    new SerialAMSenderC(AM_OSCILLOSCOPE);   // Sends to the serial
port

  MultihopOscilloscopeC.RadioControl -> ActiveMessageC;
  MultihopOscilloscopeC.SerialControl -> SerialActiveMessageC;
  MultihopOscilloscopeC.RoutingControl -> Collector;

  MultihopOscilloscopeC.Send -> CollectionSenderC;
  MultihopOscilloscopeC.SerialSend -> SerialAMSenderC.AMSend;
  MultihopOscilloscopeC.Snoop ->
Collector.Snoop[AM_OSCILLOSCOPE];
  MultihopOscilloscopeC.Receive ->
Collector.Receive[AM_OSCILLOSCOPE];
  MultihopOscilloscopeC.RootControl -> Collector;

  components new PoolC(message_t, 10) as UARTMessagePoolP,
    new QueueC(message_t*, 10) as UARTQueueP;

  MultihopOscilloscopeC.UARTMessagePool -> UARTMessagePoolP;
  MultihopOscilloscopeC.UARTQueue -> UARTQueueP;

  components new PoolC(message_t, 20) as DebugMessagePool,
    new QueueC(message_t*, 20) as DebugSendQueue,
    new SerialAMSenderC(AM_CTP_DEBUG) as DebugSerialSender,
    UARTDebugSenderP as DebugSender;

  DebugSender.Boot -> MainC;
  DebugSender.UARTSend -> DebugSerialSender;
  DebugSender.MessagePool -> DebugMessagePool;
  DebugSender.SendQueue -> DebugSendQueue;
  Collector.CollectionDebug -> DebugSender;

}
```

MultihopOscilloscopeC.nc

```
/*********************************************/
/*
* This application is the MultihopOscilloscope
* modified to get the temperature and send it to
* the root of the tree (telosb motes)
*
* Modified by Constantinos Constantinides
*/
/*********************************************/

#include "Timer.h"
#include "MultihopOscilloscope.h"

module MultihopOscilloscopeC {
  uses {
    // Interfaces for initialization:
    interface Boot;
    interface SplitControl as RadioControl;
    interface SplitControl as SerialControl;
    interface StdControl as RoutingControl;

    // Interfaces for communication, multihop and serial:
    interface Send;
    interface Receive as Snoop;
    interface Receive;
    interface AMSend as SerialSend;
    interface CollectionPacket;
    interface RootControl;

    interface Queue<message_t *> as UARTQueue;
    interface Pool<message_t> as UARTMessagePool;

    // Miscalleny:
    interface Timer<TMilli>;
    interface Read<uint16_t>;
    interface Leds;
  }
}

implementation {
  task void uartSendTask();
  static void startTimer();
  static void fatal_problem();
  static void report_problem();
  static void report_sent();
  static void report_received();
```

```
  uint8_t uartlen;
  message_t sendbuf;
  message_t uartbuf;
  bool sendbusy=FALSE, uartbusy=FALSE;

  /* Current local state - interval, version and accumulated readings */
  oscilloscope_t local;

  uint8_t reading; /* 0 to NREADINGS */

  /* When we head an Oscilloscope message, we check it's sample
count. If it's ahead of ours, we "jump" forwards (set our count to the
received
     count). However, we must then suppress our next count increment.
This is a very simple form of "time" synchronization (for an abstract
     notion of time). */
  bool suppress_count_change;

  //
  // On bootup, initialize radio and serial communications, and our
  // own state variables.
  //
  event void Boot.booted() {
    local.interval = DEFAULT_INTERVAL;
    local.id = TOS_NODE_ID;
    local.version = 0;

    // Beginning our initialization phases:
    if (call RadioControl.start() != SUCCESS)
      fatal_problem();

    if (call RoutingControl.start() != SUCCESS)
      fatal_problem();
  }

  event void RadioControl.startDone(error_t error) {
    if (error != SUCCESS)
      fatal_problem();

    if (sizeof(local) > call Send.maxPayloadLength())
      fatal_problem();

    if (call SerialControl.start() != SUCCESS)
      fatal_problem();
   }

  event void SerialControl.startDone(error_t error) {
    if (error != SUCCESS)
```

```
      fatal_problem();

    // This is how to set yourself as a root to the collection layer:

    // Changed by NXT team
    // The rood is the mode with the id number 69
    // previusly--->  if (local.id % 500 == 0)
   if (local.id == 69){
    call Leds.led0On();
    call Leds.led1On();
    call Leds.led2On();
    call RootControl.setRoot();
    }
    startTimer();reading = 0;
  }

  static void startTimer() {
    if (call Timer.isRunning()) call Timer.stop();
    call Timer.startPeriodic(local.interval);
    reading = 0;
  }

  event void RadioControl.stopDone(error_t error) {}
  event void SerialControl.stopDone(error_t error) {}

  //
  // Only the root will receive messages from this interface; its job
  // is to forward them to the serial uart for processing on the pc
  // connected to the sensor network.
  //
  event message_t*
  Receive.receive(message_t* msg, void *payload, uint8_t len) {
    oscilloscope_t* in = (oscilloscope_t*)payload;
    oscilloscope_t* out;
    if (uartbusy == FALSE) {
      out = (oscilloscope_t*)call SerialSend.getPayload(&uartbuf,
sizeof(oscilloscope_t));
      if (len != sizeof(oscilloscope_t) || out == NULL) {
        return msg;
      }
      else {
        memcpy(out, in, sizeof(oscilloscope_t));
      }
      uartlen = sizeof(oscilloscope_t);
      post uartSendTask();
    } else {
      // The UART is busy; queue up messages and service them when
the
```

```
      // UART becomes free.
      message_t *newmsg = call UARTMessagePool.get();
      if (newmsg == NULL) {
        // drop the message on the floor if we run out of queue space.
        report_problem();
        return msg;
      }

      //Serial port busy, so enqueue.
      out = (oscilloscope_t*)call SerialSend.getPayload(newmsg,
sizeof(oscilloscope_t));
      if (out == NULL) {
         return msg;
      }
      memcpy(out, in, sizeof(oscilloscope_t));

      if (call UARTQueue.enqueue(newmsg) != SUCCESS) {
        // drop the message on the floor and hang if we run out of
        // queue space without running out of queue space first (this
        // should not occur).
        call UARTMessagePool.put(newmsg);
        fatal_problem();
        return msg;
      }
    }

    return msg;
  }

  task void uartSendTask() {
    if (call SerialSend.send(0xffff, &uartbuf, uartlen) != SUCCESS) {
      report_problem();
    } else {
      uartbusy = TRUE;
    }
  }

  event void SerialSend.sendDone(message_t *msg, error_t error) {
    uartbusy = FALSE;
    if (call UARTQueue.empty() == FALSE) {
      // We just finished a UART send, and the uart queue is
      // non-empty.  Let's start a new one.
      message_t *queuemsg = call UARTQueue.dequeue();
      if (queuemsg == NULL) {
        fatal_problem();
        return;
      }
      memcpy(&uartbuf, queuemsg, sizeof(message_t));
```

```
      if (call UARTMessagePool.put(queuemsg) != SUCCESS) {
        fatal_problem();
        return;
      }
      post uartSendTask();
    }
  }
  //
  // Overhearing other traffic in the network.
  //
  event message_t*
  Snoop.receive(message_t* msg, void* payload, uint8_t len) {
    oscilloscope_t *omsg = payload;

    report_received();

    // If we receive a newer version, update our interval.
    if (omsg->version > local.version) {
      local.version = omsg->version;
      local.interval = omsg->interval;
      startTimer();
    }

    // If we hear from a future count, jump ahead but suppress our own
    // change.
    if (omsg->count > local.count) {
      local.count = omsg->count;
      suppress_count_change = TRUE;
    }

    return msg;
  }
  /* At each sample period:
     - if local sample buffer is full, send accumulated samples
     - read next sample
  */
  event void Timer.fired() {
    if (reading == NREADINGS) {
      if (!sendbusy) {
        oscilloscope_t *o = (oscilloscope_t *)call
Send.getPayload(&sendbuf, sizeof(oscilloscope_t));
        if (o == NULL) {
          fatal_problem();
          return;
        }
        memcpy(o, &local, sizeof(local));
        if (call Send.send(&sendbuf, sizeof(local)) == SUCCESS)
          sendbusy = TRUE;
```

```
      else
        report_problem();
    }

    reading = 0;
    /* Part 2 of cheap "time sync": increment our count if we didn't
       jump ahead. */
    if (!suppress_count_change)
      local.count++;
    suppress_count_change = FALSE;
  }

  if (call Read.read() != SUCCESS)
    fatal_problem();
}

event void Send.sendDone(message_t* msg, error_t error) {
  if (error == SUCCESS)
    report_sent();
  else
    report_problem();

  sendbusy = FALSE;
}
event void Read.readDone(error_t result, uint16_t data) {
  if (result != SUCCESS) {
    data = 0xffff;
    report_problem();
  }
  local.readings[reading++] = -38.4 + 0.0098 * data;
}

// Use LEDs to report various status issues.
static void fatal_problem() {
// commented by nxt team
// call Leds.led0On();
  call Leds.led1On();
  call Leds.led2On();
  call Timer.stop();
}

static void report_problem() { call Leds.led0Toggle(); }
static void report_sent() { call Leds.led1Toggle(); }
static void report_received() { call Leds.led2Toggle(); }
}
```

MultihopOscilloscope.h

```
/*********************************************/
```

```
/*
* This application is the MultihopOscilloscope
* modified to get the temperature and send it to
* the root of the tree (telosb motes)
*
* Modified by Constantinos Constantinides
*/
/*****************************************/

#ifndef MULTIHOP_OSCILLOSCOPE_H
#define MULTIHOP_OSCILLOSCOPE_H

enum {
  /* Number of readings per message. If you increase this, you may have
to
     increase the message_t size. */
  NREADINGS = 5,
  /* Default sampling period. */
  DEFAULT_INTERVAL = 1024,
  AM_OSCILLOSCOPE = 0x93
};

typedef nx_struct oscilloscope {
  nx_uint16_t version; /* Version of the interval. */
  nx_uint16_t interval; /* Samping period. */
  nx_uint16_t id; /* Mote id of sending mote. */
  nx_uint16_t count; /* The readings are samples count * NREADINGS
onwards */
  nx_uint16_t readings[NREADINGS];
} oscilloscope_t;

#endif
```

makefile

```
COMPONENT=MultihopOscilloscopeAppC
CFLAGS += -I$(TOSDIR)/lib/net/ -I$(TOSDIR)/lib/net/ctp -
I$(TOSDIR)/lib/net/4bitle
CFLAGS += "-DCC2420_DEF_RFPOWER=3"
include $(MAKERULES)
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκαν οι κόμβος micaz για την multihop επικοινωνία με τη χρήση του πρωτοκόλλου Surge (με χρήση αισθητήρα mts400 θερμοκρασίας)

Surge.nc

```
/*****************************************************/
/*
* This application is the Surge modified
```

```
* to get the temperature and send it over
* RF to the base station
*
*/
/*****************************************************/
includes Surge;
includes SurgeCmd;
includes MultiHop;


configuration Surge {
}
implementation {
  components Main, SurgeM, TimerC, LedsC, NoLeds,
SensirionHumidity,CC2420RadioC, RandomLFSR,
    GenericCommPromiscuous as Comm, Bcast, MultiHopRouter as
multihopM, QueuedSend/*, Sounder*/;

  Main.StdControl -> SurgeM.StdControl;
  //Main.StdControl -> Photo;
  Main.StdControl -> Bcast.StdControl;
  Main.StdControl -> multihopM.StdControl;
  Main.StdControl -> QueuedSend.StdControl;
  Main.StdControl -> TimerC;
  Main.StdControl -> Comm;
  //  multihopM.CommControl -> Comm;

  // enosi gia meiwsi tou transmision power
  SurgeM.CC2420Control -> CC2420RadioC.CC2420Control;

   SurgeM.TempHumControl -> SensirionHumidity;
   SurgeM.Humidity -> SensirionHumidity.Humidity;
  SurgeM.TemperatureError -> SensirionHumidity.TemperatureError;
    SurgeM.ADC   -> SensirionHumidity.Temperature;
  //SurgeM.ADC   -> Photo;
  SurgeM.Timer -> TimerC.Timer[unique("Timer")];
  SurgeM.Leds  -> LedsC; // NoLeds;
  //SurgeM.Sounder -> Sounder;

  SurgeM.Bcast -> Bcast.Receive[AM_SURGECMDMSG];
  Bcast.ReceiveMsg[AM_SURGECMDMSG] ->
Comm.ReceiveMsg[AM_SURGECMDMSG];

  SurgeM.RouteControl -> multihopM;
  SurgeM.Send -> multihopM.Send[AM_SURGEMSG];
  multihopM.ReceiveMsg[AM_SURGEMSG] ->
Comm.ReceiveMsg[AM_SURGEMSG];
  //multihopM.ReceiveMsg[AM_MULTIHOPMSG] ->
```

```
Comm.ReceiveMsg[AM_MULTIHOPMSG];
}
```

SurgeM.nc

```
/*********************************************************/
/*
* This application is the Surge modified
* to get the temperature and send it over
* RF to the base station
*
*/
/*********************************************************/

includes Surge;
includes SurgeCmd;

/*
 *  Data gather application
 */

module SurgeM {
  provides {
    interface StdControl;
  }
  uses {
    interface ADC;
    interface Timer;
    interface Leds;
    //interface StdControl as Sounder;
    interface Send;
    interface Receive as Bcast;
    interface RouteControl;
        //interface gia miwsi tou transmission power
        interface CC2420Control;

        interface SplitControl as TempHumControl;
    interface ADC as Humidity;


    interface ADCError as TemperatureError;
  }
}

implementation {
```

```
  enum {
    TIMER_GETADC_COUNT = 1,          // Timer ticks for ADC
    TIMER_CHIRP_COUNT = 10,          // Timer on/off chirp count
  };

  bool sleeping;                     // application command state
  bool focused;
  bool rebroadcast_adc_packet;

  TOS_Msg gMsgBuffer;
  norace uint16_t gSensorData;               // protected by gfSendBusy
flag
  bool gfSendBusy;


  int timer_rate;
  int timer_ticks;
  /*****************************************************************
   * Initialization
   *****************************************************************/

  static void initialize() {
    timer_rate = INITIAL_TIMER_RATE;
    atomic gfSendBusy = FALSE;
    sleeping = FALSE;
    rebroadcast_adc_packet = FALSE;
    focused = FALSE;
  }

  task void SendData() {
    SurgeMsg *pReading;
    uint16_t Len;
    dbg(DBG_USR1, "SurgeM: Sending sensor reading\n");

    if (pReading = (SurgeMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) {
      pReading->type = SURGE_TYPE_SENSORREADING;
      pReading->parentaddr = call RouteControl.getParent();
      pReading->reading = gSensorData;

      if ((call Send.send(&gMsgBuffer,sizeof(SurgeMsg))) != SUCCESS)
        atomic gfSendBusy = FALSE;
    }

  }

  command result_t StdControl.init() {
      call TempHumControl.init();    //init Sensirion
```

```
        call Leds.init();
    initialize();
    return SUCCESS;
  }

  command result_t StdControl.start() {
    call CC2420Control.SetRFPower(3);
    call TempHumControl.start();
    call TemperatureError.enable();              // in case Sensirion doesn't
respond

    return call Timer.start(TIMER_REPEAT, timer_rate);
    return SUCCESS;
  }

  command result_t StdControl.stop() {
     call TempHumControl.stop();
    return call Timer.stop();
  }

  /****************************************************************
   * Commands and events
   ****************************************************************/

  event result_t Timer.fired() {
    dbg(DBG_USR1, "SurgeM: Timer fired\n");
    timer_ticks++;
    if (timer_ticks % TIMER_GETADC_COUNT == 0) {
      call ADC.getData();
    }
    // If we're the focused node, chirp
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
    //  call Sounder.start();
    }
    // If we're the focused node, chirp
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
    //  call Sounder.stop();
    }
    return SUCCESS;
  }

  async event result_t ADC.dataReady(uint16_t data) {
    //SurgeMsg *pReading;
    //uint16_t Len;
    dbg(DBG_USR1, "SurgeM: Got ADC reading: 0x%x\n", data);
    atomic {
      if (!gfSendBusy) {
        gfSendBusy = TRUE;
```

```
        gSensorData =-38.4 + 0.0098 * data;
        post SendData();
      }
    }
    return SUCCESS;
  }

  event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
    dbg(DBG_USR2, "SurgeM: output complete 0x%x\n", success);
    //call Leds.greenToggle();
    atomic gfSendBusy = FALSE;
    return SUCCESS;
  }


  /* Command interpreter for broadcasts
   *
   */

  event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload,
uint16_t payloadLen) {
    SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload;

    dbg(DBG_USR2, "SurgeM: Bcast  type 0x%02x\n", pCmdMsg->type);

    if (pCmdMsg->type == SURGE_TYPE_SETRATE) {       // Set timer
rate
      timer_rate = pCmdMsg->args.newrate;
      dbg(DBG_USR2, "SurgeM: set rate %d\n", timer_rate);
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, timer_rate);

    } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
      // Go to sleep - ignore everything until a SURGE_TYPE_WAKEUP
      dbg(DBG_USR2, "SurgeM: sleep\n");
      sleeping = TRUE;
      call Timer.stop();
      call Leds.greenOff();
      call Leds.yellowOff();

    } else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
      dbg(DBG_USR2, "SurgeM: wakeup\n");

      // Wake up from sleep state
      if (sleeping) {
        initialize();
       call Timer.start(TIMER_REPEAT, timer_rate);
        sleeping = FALSE;
```

```
    }

  } else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
    dbg(DBG_USR2, "SurgeM: focus %d\n", pCmdMsg-
>args.focusaddr);
    // Cause just one node to chirp and increase its sample rate;
    // all other nodes stop sending samples (for demo)
    if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
      // OK, we're focusing on me
      focused = TRUE;
      //call Sounder.init();
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
    } else {
      // Focusing on someone else
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
    }

  } else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
    // Return to normal after focus command
    dbg(DBG_USR2, "SurgeM: unfocus\n");
    focused = FALSE;
   // call Sounder.stop();
    call Timer.stop();
    call Timer.start(TIMER_REPEAT, timer_rate);
  }
  return pMsg;
}

event result_t TempHumControl.startDone() {

    return SUCCESS;
}

event result_t TempHumControl.initDone() {
    return SUCCESS;
}

event result_t TempHumControl.stopDone() {
    return SUCCESS;
}

event result_t TemperatureError.error(uint8_t token) {

    return SUCCESS;
}
```

```
   async event result_t Humidity.dataReady(uint16_t data) {

      return SUCCESS;
  }

}
```

SurgeCmd.h

```
/*****************************************************/
/*
* This application is the Surge modified
* to get the temperature and send it over
* RF to the base station
*
*/
/*****************************************************/

typedef struct SurgeCmdMsg {
  uint8_t type;
  union {
    // FOR SURGE_TYPE_SETRATE
    uint32_t newrate;
    // FOR SURGE_TYPE_FOCUS
    uint16_t focusaddr;
  } args;

} __attribute__ ((packed)) SurgeCmdMsg;

enum {
  AM_SURGECMDMSG = 18
};
```

Surge.h

```
/*****************************************************/
/*
* This application is the Surge modified
* to get the temperature and send it over
* RF to the base station
*
*/
/*****************************************************/
int INITIAL_TIMER_RATE = 2048;
int FOCUS_TIMER_RATE = 1000;
int FOCUS_NOTME_TIMER_RATE = 1000;

enum {
```

```
  SURGE_TYPE_SENSORREADING = 0,
  SURGE_TYPE_ROOTBEACON = 1,
  SURGE_TYPE_SETRATE = 2,
  SURGE_TYPE_SLEEP = 3,
  SURGE_TYPE_WAKEUP = 4,
  SURGE_TYPE_FOCUS = 5,
  SURGE_TYPE_UNFOCUS = 6
};

typedef struct SurgeMsg {
  uint8_t type;
  uint16_t reading;
  uint16_t parentaddr;
} __attribute__ ((packed)) SurgeMsg;

enum {
  AM_SURGEMSG = 17
};
```

makefile

```
# Use SurgeTelos for telos
PLATFORMS=mica mica2 mica2dot micaz pc
COMPONENT=Surge
#SENSORBOARD=basicsb
SENSORBOARD=mts400

PFLAGS= -I%T/lib/Route -I%T/lib/Queue -I%T/lib/Broadcast

include ../Makerules
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκε ο κόμβος micaz για την επικοινωνία RF με το boebot (TinyOs1.x)

CounterToFour.nc

```
//////////////////////////////////////////////////////////
// Counter.nc modified to count from 1 to 4 forever     //
//                                                      //
// Modified by Constantinos Constantinides              //
// Date last modified:  3/02/09                         //
//                                                      //
//////////////////////////////////////////////////////////

module CounterToFour {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface IntOutput;
  }
```

```
}
implementation {
  // the counter
  int count;

  //initialization
  command result_t StdControl.init()
  {
    count = 1;
    return SUCCESS;
  }

  //Start
  command result_t StdControl.start()
  {
    return call Timer.start(TIMER_REPEAT, 2000);
  }

  //Stop
  command result_t StdControl.stop()
  {
    return call Timer.stop();
  }

  // Timer event expired
  event result_t Timer.fired()
  {
    //Output the counter
    if (call IntOutput.output(count))
          //increase the counter
      count++;
          //The counter counts from 1 to 4
          if (count==5){
          count=1;
          }

    return SUCCESS;
  }

  event result_t IntOutput.outputComplete(result_t success)
  {
    //if the counter fail to output then the counter decreases
    if(success == 0) count --;
    return SUCCESS;
  }
}
```

CounterToLedsAndRfm.nc

```
//////////////////////////////////////////////////////////////
//                                                          //
//  CntToLedsAndRfm modified to use CounterToFour.nc in place of //
//  Counter.nc                                              //
// This application blinks the LEDS as a binary counter and also //
//  send a radio packet sending the current value of the counter //
//                                                          //
// Author: Constantinos Constantinides                      //
// Date last modified:  3/02/09                             //
//                                                          //
//////////////////////////////////////////////////////////////


configuration CounterToLedsAndRfm {
}
implementation {

  components Main, CounterToFour, IntToLeds,IntToRfm, TimerC;
  Main.StdControl -> CounterToFour.StdControl;
  Main.StdControl -> IntToLeds.StdControl;
  Main.StdControl -> IntToRfm.StdControl;
  Main.StdControl -> TimerC.StdControl;
  CounterToFour.Timer -> TimerC.Timer[unique("Timer")];
  IntToLeds <- CounterToFour.IntOutput;
  CounterToFour.IntOutput -> IntToRfm;
}
```

makefile

```
include Makefile.component
include ../../MakeXbowlocal
PFLAGS=-I%T/lib/Counters
# Automatically add certain command-line goals:
GOALS += basic freq route
include $(MAKERULES)
```

Makefile.component

```
COMPONENT=CounterToLedsAndRfm
SENSORBOARD=micasb
INCLUDES= -I%T/lib/Counters
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκε ο κόμβος micaz που ήταν ενσωματωμένος με το boebot (TinyOs1.x)

IntToLedsAndUart.nc

```
//////////////////////////////////////////////////////////////
```

```
//                                                                  //
// Configuration                                                    //
// Displays the integer on the Leds (three most signifigant bits)   //
// and send them over the UART                                      //
//                                                                  //
// Modified by: Constantinos Constantinides                         //
// Date last modified:  3/02/09                                     //
//                                                                  //
//////////////////////////////////////////////////////////////////////

configuration IntToLedsAndUart
{
  provides interface IntOutput;
  provides interface StdControl;
}
implementation
{
  components IntToLedsAndUartM,LedsC,HPLUARTC;

  IntOutput = IntToLedsAndUartM.IntOutput;
  StdControl = IntToLedsAndUartM.StdControl;
  IntToLedsAndUartM.Leds -> LedsC.Leds;
  IntToLedsAndUartM.HPLUART -> HPLUARTC;

}
```

IntToLedsAndUartM.nc

```
//////////////////////////////////////////////////////////////////////
//                                                                  //
//       Module - Implementation                                    //
// Displays the integer on the Leds (three most signifigant bits)   //
// and send them over the UART                                      //
//                                                                  //
// Modified by: Constantinos Constantinides                         //
// Date last modified:  3/02/09                                     //
//                                                                  //
//////////////////////////////////////////////////////////////////////
module IntToLedsAndUartM {
  provides interface IntOutput;
  provides interface StdControl;
  uses interface Leds;
  uses interface HPLUART;
}
implementation
{
  //inisialization
  command result_t StdControl.init()
```

```
{
  //initialization of leds and begin switched off
  call Leds.init();
  call Leds.redOff();
  call Leds.yellowOff();
  call Leds.greenOff();

      //to initialize the uart
      call HPLUART.init();
      //outp(0,UBRR0H);
      //outp(95,UBRR0L);

  return SUCCESS;
}

//Start
command result_t StdControl.start() {
  return SUCCESS;
}

//Stop
command result_t StdControl.stop() {
  return SUCCESS;
}

// signals the result of an incoming <<output>> from outsite
task void outputDone()
{
  signal IntOutput.outputComplete(1);
}

// processing of the incoming <<output>> value
command result_t IntOutput.output(uint16_t value)
{
//send Value over uart
  call HPLUART.put('0'+ value);
  call HPLUART.put('\n');

// Display value on the Leds
  if (value & 1) call Leds.redOn();
  else call Leds.redOff();
  if (value & 2) call Leds.greenOn();
  else call Leds.greenOff();
  if (value & 4) call Leds.yellowOn();
  else call Leds.yellowOff();
  post outputDone();
  return SUCCESS;
}
```

```
 //implementation of HPLUART's event get
async  event result_t HPLUART.get(uint8_t data) {
   // call Leds.greenToggle();
    return SUCCESS;
  }

//implementation of HPLUART's event putDone
async  event result_t HPLUART.putDone() {
   // call Leds.yellowToggle();
    return SUCCESS;
  }

}
```

RfmToLedsAndUart.nc

```
//////////////////////////////////////////////////////////////
//                                                          //
// This application will receive packets (numbers) from RF, //
// display them on the Leds and send them over UART         //
//                                                          //
// Modified by Constantinos Constantinides                  //
// Date last modified:  3/02/09                             //
//                                                          //
//////////////////////////////////////////////////////////////

configuration RfmToLedsAndUart {
}
implementation {
  components Main, RfmToInt, IntToLedsAndUart;

  Main.StdControl -> IntToLedsAndUart.StdControl;
  Main.StdControl -> RfmToInt.StdControl;
  RfmToInt.IntOutput -> IntToLedsAndUart.IntOutput;
}
```

makefile

```
include Makefile.component
include ../../MakeXbowlocal
PFLAGS=-I%T/lib/Counters
# Automatically add certain command-line goals:
GOALS += basic freq route
include $(MAKERULES)
```

Makefile.component

```
COMPONENT=RfmToLedsAndUart
SENSORBOARD=micasb
INCLUDES= -I%T/lib/Counters
```

Κώδικας γραμμένος σε nesC που προγραμματίστηκαν οι κόμβος micaz για τη multihop επικοινωνία με τη χρήση του πρωτοκόλλου Surge (με χρήση αισθητήρα mts400 φωτός)

SurgeM.nc (Surge_mst_light)

```
/*
* This application is the Surge modified
* to get the light readinds and send it over
* RF to the base station
*
*/

includes Surge;
includes SurgeCmd;

/*
 *  Data gather application
 */

module SurgeM {
  provides {
    interface StdControl;
  }
  uses {
    interface ADC;
    interface Timer;
    interface Leds;
    //interface StdControl as Sounder;
    interface Send;
    interface Receive as Bcast;
    interface RouteControl;


    interface SplitControl as photoControl;
    //interface ADC as photo;
  }
}

implementation {

  enum {
    TIMER_GETADC_COUNT = 1,          // Timer ticks for ADC
    TIMER_CHIRP_COUNT = 10,          // Timer on/off chirp count
  };

  bool sleeping;                     // application command state
  bool focused;
  bool rebroadcast_adc_packet;
```

```
TOS_Msg gMsgBuffer;
norace uint16_t gSensorData;         // protected by gfSendBusy flag
bool gfSendBusy;

int timer_rate;
int timer_ticks;

/*****************************************************************
 * Initialization
 *****************************************************************/
static void initialize() {
  timer_rate = INITIAL_TIMER_RATE;
  atomic gfSendBusy = FALSE;
  sleeping = FALSE;
  rebroadcast_adc_packet = FALSE;
  focused = FALSE;
}

task void SendData() {
  SurgeMsg *pReading;
  uint16_t Len;
  dbg(DBG_USR1, "SurgeM: Sending sensor reading\n");

  if (pReading = (SurgeMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) {
    pReading->type = SURGE_TYPE_SENSORREADING;
    pReading->parentaddr = call RouteControl.getParent();
    pReading->reading = gSensorData-65400;

    if ((call Send.send(&gMsgBuffer,sizeof(SurgeMsg))) != SUCCESS)
      atomic gfSendBusy = FALSE;
  }

}

command result_t StdControl.init() {
call photoControl.init();
  call Leds.init();
  initialize();
  return SUCCESS;
}

command result_t StdControl.start() {
  call photoControl.start();
  return call Timer.start(TIMER_REPEAT, timer_rate);
  return SUCCESS;
}
```

```
command result_t StdControl.stop() {
 call photoControl.stop();
  return call Timer.stop();
}

/*****************************************************************
 * Commands and events
 *****************************************************************/

event result_t Timer.fired() {
  dbg(DBG_USR1, "SurgeM: Timer fired\n");
  timer_ticks++;
  if (timer_ticks % TIMER_GETADC_COUNT == 0) {
    call ADC.getData();
  }
  // If we're the focused node, chirp
  if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
    //call Sounder.start();
  }
  // If we're the focused node, chirp
  if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
   // call Sounder.stop();
  }
  return SUCCESS;
}

async event result_t ADC.dataReady(uint16_t data) {
  //SurgeMsg *pReading;
  //uint16_t Len;
  dbg(DBG_USR1, "SurgeM: Got ADC reading: 0x%x\n", data);
  atomic {
    if (!gfSendBusy) {
       gfSendBusy = TRUE;
       gSensorData = data;
       post SendData();
    }
  }
  return SUCCESS;
}

event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
  dbg(DBG_USR2, "SurgeM: output complete 0x%x\n", success);
  //call Leds.greenToggle();
  atomic gfSendBusy = FALSE;
  return SUCCESS;
}
```

```
  /* Command interpreter for broadcasts
   *
   */

  event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload,
uint16_t payloadLen) {
    SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload;

    dbg(DBG_USR2, "SurgeM: Bcast  type 0x%02x\n", pCmdMsg->type);

    if (pCmdMsg->type == SURGE_TYPE_SETRATE) {    // Set timer rate
      timer_rate = pCmdMsg->args.newrate;
      dbg(DBG_USR2, "SurgeM: set rate %d\n", timer_rate);
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, timer_rate);

    } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
      // Go to sleep - ignore everything until a SURGE_TYPE_WAKEUP
      dbg(DBG_USR2, "SurgeM: sleep\n");
      sleeping = TRUE;
      call Timer.stop();
      call Leds.greenOff();
      call Leds.yellowOff();

    } else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
      dbg(DBG_USR2, "SurgeM: wakeup\n");

      // Wake up from sleep state
      if (sleeping) {
        initialize();
       call Timer.start(TIMER_REPEAT, timer_rate);
        sleeping = FALSE;
      }

    } else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
      dbg(DBG_USR2, "SurgeM: focus %d\n", pCmdMsg-
>args.focusaddr);
      // Cause just one node to chirp and increase its sample rate;
      // all other nodes stop sending samples (for demo)
      if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
        // OK, we're focusing on me
        focused = TRUE;
        //call Sounder.init();
        call Timer.stop();
        call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
      } else {
        // Focusing on someone else
        call Timer.stop();
```

```
        call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
      }

    } else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
      // Return to normal after focus command
      dbg(DBG_USR2, "SurgeM: unfocus\n");
      focused = FALSE;
      //call Sounder.stop();
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, timer_rate);
    }
    return pMsg;
  }
  event result_t photoControl.startDone() {

      return SUCCESS;
  }
  event result_t photoControl.initDone() {
      return SUCCESS;
  }
  event result_t photoControl.stopDone() {
      return SUCCESS;
  }

}

}
```

Surge.nc (Surge_mst_light)

```
/*
* This application is the Surge modified
* to get the light readinds and send it over
* RF to the base station
*
*/

includes Surge;
includes SurgeCmd;
includes MultiHop;

configuration Surge {
}
implementation {
  components Main, SurgeM, TimerC, LedsC, NoLeds, RandomLFSR,
    GenericCommPromiscuous as Comm, Bcast, MultiHopRouter as
multihopM, QueuedSend,TaosPhoto/*, Sounder*/;

  Main.StdControl -> SurgeM.StdControl;
```

```
  //Main.StdControl -> Photo;
  Main.StdControl -> Bcast.StdControl;
  Main.StdControl -> multihopM.StdControl;
  Main.StdControl -> QueuedSend.StdControl;
  Main.StdControl -> TimerC;
  Main.StdControl -> Comm;

  //to create
  SurgeM.photoControl ->TaosPhoto;
  //connect to the  light sensor of mts400
  SurgeM.ADC -> TaosPhoto.ADC[0];

  //  multihopM.CommControl -> Comm;

  //SurgeM.ADC   -> Photo;
  SurgeM.Timer -> TimerC.Timer[unique("Timer")];
  SurgeM.Leds  -> LedsC; // NoLeds;
  //SurgeM.Sounder -> Sounder;

  SurgeM.Bcast -> Bcast.Receive[AM_SURGECMDMSG];
  Bcast.ReceiveMsg[AM_SURGECMDMSG] ->
Comm.ReceiveMsg[AM_SURGECMDMSG];

  SurgeM.RouteControl -> multihopM;
  SurgeM.Send -> multihopM.Send[AM_SURGEMSG];
  multihopM.ReceiveMsg[AM_SURGEMSG] ->
Comm.ReceiveMsg[AM_SURGEMSG];
  //multihopM.ReceiveMsg[AM_MULTIHOPMSG] ->
Comm.ReceiveMsg[AM_MULTIHOPMSG];
}
```

Makefile (Surge_mst_light)

```
PLATFORMS=mica mica2 mica2dot micaz pc
COMPONENT=Surge
SENSORBOARD=mts400

PFLAGS= -I%T/lib/Route -I%T/lib/Queue -I%T/lib/Broadcast

include ../Makerules
```

Κώδικας γραμμένος σε PBASIC που προγραμματίστηκε το boebot για την απλή κίνηση μέσω εντολών του ενσωματομένου micaz

BS2P-move.BS2

```
' {$STAMP BS2p}
' {$PBASIC 2.5}

pulse_count VAR Word
CHOICE VAR Word
TOP:
'57600 baud rate
SEROUT 16, 23, 10, ["Enter number between 1 and 4", CR, 10]
SERIN 16, 23, [DEC CHOICE]

IF CHOICE = 1 THEN
  GOTO forward
ELSEIF CHOICE=2 THEN
  GOTO backward
ELSEIF CHOICE=3 THEN
  GOTO right
ELSEIF CHOICE=4 THEN
  GOTO left
ENDIF

GOTO TOP

forward:
FOR pulse_count =0 TO 50
  PULSOUT 12,1250
  PULSOUT 13,2500
  PAUSE 20
NEXT
GOTO TOP

backward:
FOR pulse_count = 0 TO 50
    PULSOUT 12, 2500
    PULSOUT 13, 1250
    PAUSE 20

NEXT
GOTO TOP

right:
FOR pulse_count = 0 TO 35   'turns to right
    PULSOUT 12, 2500
    PULSOUT 13, 2500
    PAUSE 20
NEXT
```

```
GOTO TOP

left:
FOR pulse_count = 0 TO 35
   PULSOUT 12, 1250
   PULSOUT 13, 1250
   PAUSE 20
NEXT
GOTO TOP
```

Κώδικας γραμμένος σε PBASIC που προγραμματίστηκε το boebot για την κίνηση του για τη διασχιση του δικτύου κατά την εφαρμογή του αλγορίθμου στο πιλοτικό δίκτυο

BS2P-moveArray.BS2

```
' {$STAMP BS2p}
' {$PBASIC 2.5}

pulse_count VAR Word
command_in VAR Byte
value_in VAR Byte
path_commands  VAR  Byte(10)
path_values  VAR  Byte(10)
commands_counter VAR Byte
loop_counter VAR Byte

START:
commands_counter=0

TOP:

'57600 baud rate
SEROUT 16, 23, 10, ["Enter number between 1 and 4", CR, 10]
SERIN 16, 23, [DEC command_in]
SERIN 16, 23, [DEC value_in]

IF command_in < 5 THEN
   path_commands(commands_counter)=command_in
   path_values(commands_counter)=value_in
   commands_counter=commands_counter+1
   DEBUG "commands_counter=",DEC commands_counter,"
command_in=",DEC command_in, " value_in=",DEC value_in, CR
   GOTO TOP
ELSE
  DEBUG "executing path....",CR
  GOTO execute_path
  DEBUG "finished executing path",CR
ENDIF
```

```
execute_path:
  DEBUG "inside executing path.... commands_counter=",DEC
commands_counter,CR

FOR loop_counter =0 TO commands_counter-1
 IF path_commands(loop_counter)=1 THEN
   GOSUB forward
 ELSEIF path_commands(loop_counter)=2 THEN
   GOSUB backward
 ELSEIF path_commands(loop_counter)=3 THEN
   GOSUB right
 ELSEIF path_commands(loop_counter)=4 THEN
   GOSUB left
 ENDIF

NEXT
  GOTO START
  DEBUG "STOP", CR
  STOP

forward:
DEBUG "forward loop_counter=", DEC loop_counter, " val=", DEC
path_values(loop_counter), CR
FOR pulse_count =0 TO 5 * path_values(loop_counter)
  PULSOUT 12,1250
  PULSOUT 13,2500
  PAUSE 20
NEXT
  RETURN

backward:
DEBUG "backward loop_counter=", DEC loop_counter, " val=", DEC
path_values(loop_counter), CR
FOR pulse_count = 0 TO 5 * path_values(loop_counter)
    PULSOUT 12, 2500
    PULSOUT 13, 1250
    PAUSE 20

NEXT
  RETURN

right:
DEBUG "right loop_counter=", DEC loop_counter, " val=", DEC
path_values(loop_counter), CR
FOR pulse_count = 0 TO path_values(loop_counter) /4   'turns to right
    PULSOUT 12, 2500
    PULSOUT 13, 2500
```

```
    PAUSE 20
NEXT
  RETURN


left:
DEBUG "left loop_counter=", DEC loop_counter, " val=", DEC path_values(loop_counter), CR
FOR pulse_count = 0 TO path_values(loop_counter) /4
   PULSOUT 12, 1250
   PULSOUT 13, 1250
   PAUSE 20
NEXT
  RETURN
```

Παράρτημα Γ

Πώς εγκαθιστάς και τρέχεις Java πάνω σε Lego Mindstorms NXT χρησιμοποιώντας το Eclipse

Διαβάζοντας αυτό το αρχείο θα μάθεις για το πώς μπορείς να εγκαταστήσεις και να ρυθμίσεις όλα τα απαραίτητα λογισμικά για την δημιουργία προγραμμάτων γραμμένων σε γλώσσα Java για το LEGO Mindstorms NXT.Η Java είναι παρέχει περισσότερες δυνατότητες και είναι πιο ευέλικτη από το NXT-G software που παρέχει η LEGO. Θα χρησιμοποιήσουμε το περιβάλλον που παρέχει το **Eclipse** για το γράψιμο κώδικα java και την εγκατάσταση του στο NXT. Ο συνδυασμός αυτός επιτρέπει την εύκολη δημιουργία λογισμικού και το γρήγορο έλεγχο του στο NXT.

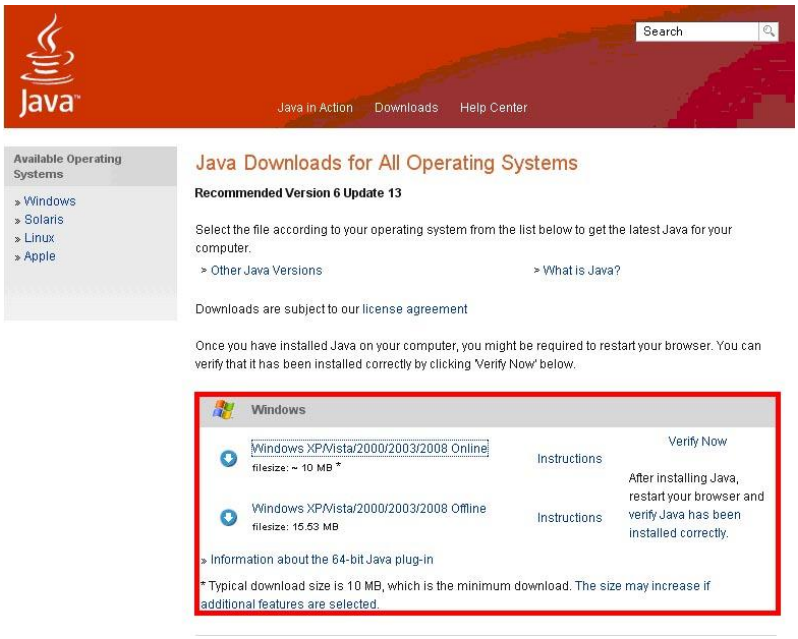
Οι πιο κάτω οδηγίες επικεντρώνονται σε **Windows XP** πλατφόρμα.

- Εγκατάσταση της Java στον υπολογιστή.
- Εγκατάσταση του Installing LEGO NXT USB driver στον υπολογιστή.
- Εγκατάσταση του Lejos στον υπολογιστή και στο Mindstorms NXT.
- Εγκατάσταση και ρύθμιση του Eclipse στον υπολογιστή.

Εγκατάσταση της Java στον υπολογιστή

Κατέβασε και εγκατέστησε το **Java SE** (Standard Edition) **JRE** (Java Runtime Environment). Δεν χρειάζεται να εγκαταστήσεις το JDK (Java Developer Kit). Θα εγκαταστήσουμε το Eclipse αργότερα, το οποίο ήδη συμπεριλαμβάνει όλα τα εργαλεία που χρειάζεσαι για να γράψεις και να μεταγλωττίσεις τα προγράμματά σου.

Μετά την εγκατάσταση δεν χρειάζεται να ορίσεις καμία από τις Path ή Classpath variables για την Java. Η Java θα εγκατασταθεί στο in "C:\Program Files\Java".



Αν έχεις ήδη εγκατεστημένη την Java πρέπει να ελέγξεις, αν έχεις τουλάχιστον την 5 έκδοση για το JRE.

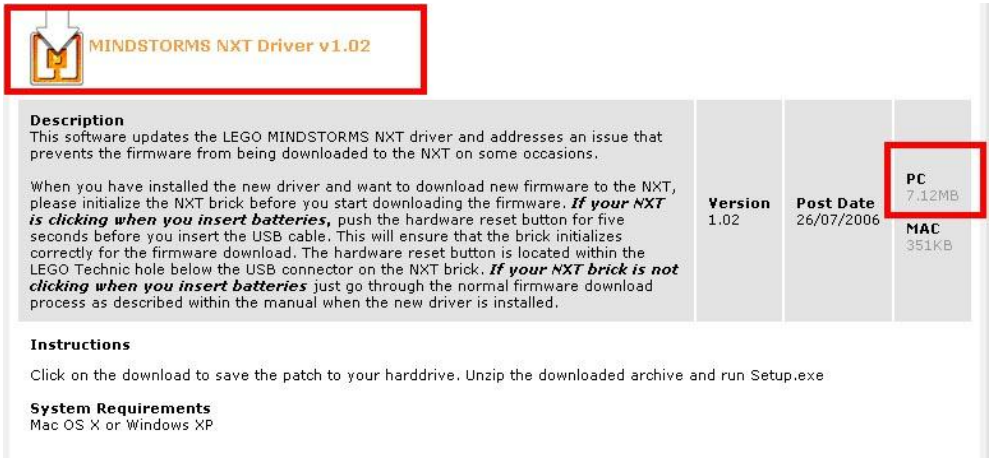
Εγκατάσταση του Installing LEGO NXT USB driver στον υπολογιστή

Το NXT μπορεί να ενωθεί στον υπολογιστή μέσο USB ή Bluetooth. Η επικοινωνία μέσο USB είναι πιο αξιόπιστη και πιο γρήγορη από αυτή του Bluetooth. Επιπλέων, το Bluetooth εξαρτάται και από το μοντέλο του υπολογιστή για να δουλέψει σωστά.

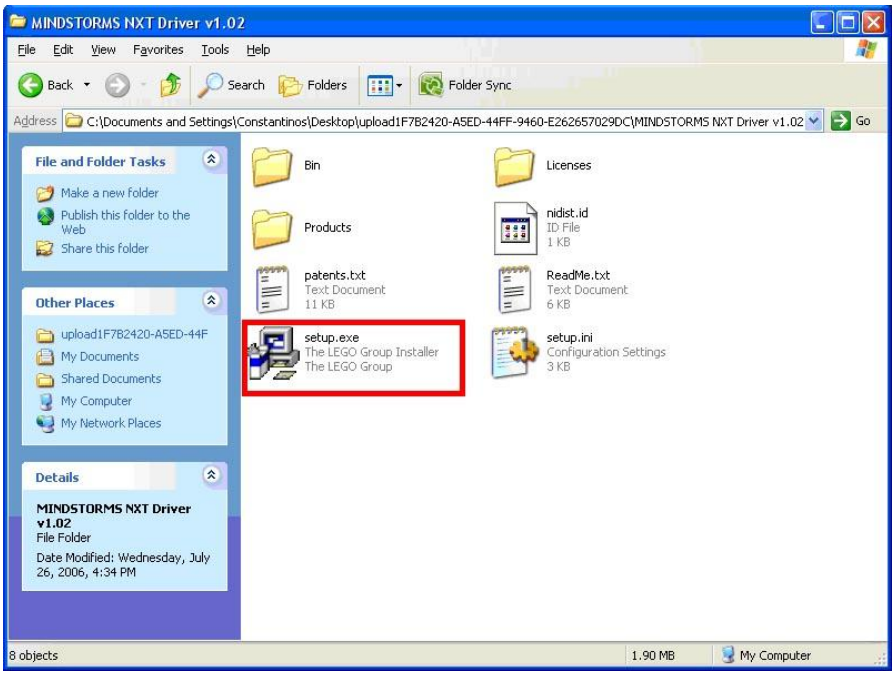
Αντίθετα, το USB είναι πιο "standardized". Πρέπει πρώτα να εγκαταστήσεις το driver και μόνο τότε θα μπορείς να ενωθείς με το NXT χρησιμοποιώντας το καλώδιο USB. Δεν χρειάζεται να εγκαταστήσεις οποιοδήποτε λογισμικό το οποίο βρίσκεται στο Mindstorms CD, γιατί δεν θα προγραμματίσουμε πάνω σε NXT-G (Το λογισμικό της LEGO το οποίο βασίζεται σε Labview). Χρειάζεται να εγκαταστήσουμε μόνο τοUSB driver

το οποίο είναι διαθέσιμο στην ιστοσελίδα της **Mindstorm** (http://mindstorms.lego.com/Support/Updates/). Αν έχεις ήδη εγκατεστημένο το λογισμικό της Mindstorm, τότε δεν χρειάζεται να το αποκαταστήσεις, απλά έλεγξε αν υπάρχει πιο νέο version για το USB driver.

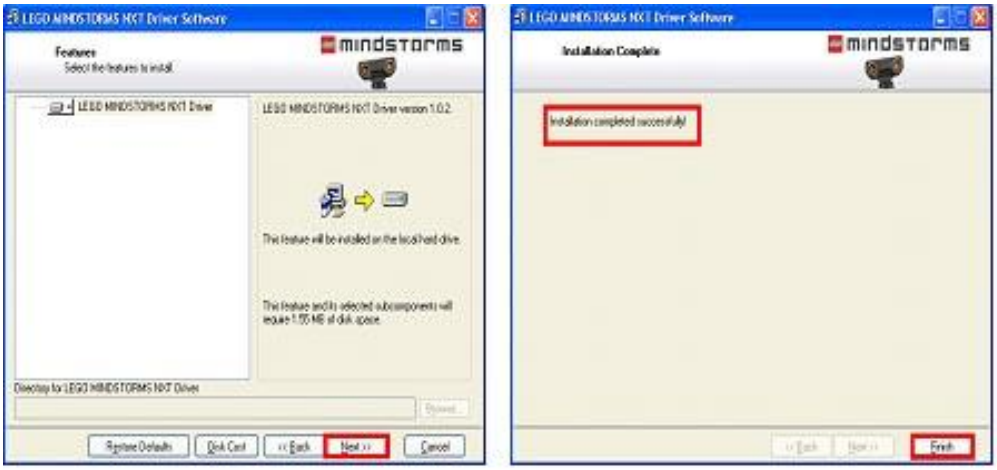
Κατέβασε το Mindstorms NXT Driver.



Κάνε unzip το file και ξεκίνα το setup.exe



Κάνε click μέσω του προγράμματος εγκαταστάσεων



Το λειτουργικό σύστημα μπορεί να σε ρωτήσει για να επανεκκινήσεις τον υπολογιστή σου μετά την εγκατάσταση. Στη συνέχεια ένωσε το NXT με τον υπολογιστή σου χρησιμοποιώντας το καλώδιο USB. Έλεγξε αν έχει γίνει σωστά η εγκατάσταση του driver κοιτάζοντας τον device manager. Δεξί click στο "My Computer" και επέλεξε το "Properties". Ακολούθως επέλεξε το hardware tab και κάνε click στο "**Device manager".** Θα πρέπει να υπάρχει το "Lego Devices => Lego Mindstorms NXT".

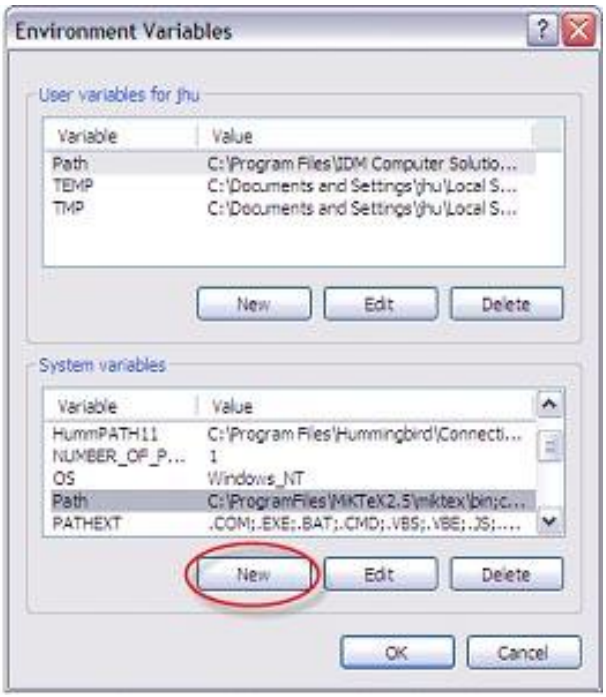
Εγκατάσταση του Lejos στον υπολογιστή και στο Mindstorms NXT

Κατέβασε το Lejos
(http://lejos.sourceforge.net/p_technologies/nxt/nxj/downloads.php)

Δημιούργησε ένα directory με το όνομα "ProgramFiles" στο σκληρό δίσκο. Το directory που ήδη υπάρχει με το όνομα "Program Files" δεν περιέχει **space character** μεταξύ του "Program" και του "Files" το οποίο προκαλεί μερικές φορές προβλήματα με τη χρήση της τεχνολογίας. Αργότερα θα εγκαταστήσεις και το Eclipse μέσα στο directory "ProgramFiles".

Κάνε unzip το Lejos ZIP file. Θα εμφανιστεί ένα directory με το όνομα "lejos_nxj" . Τοποθέτησε το στο directory "ProgramFiles".

Τώρα πρέπει να ενημερώσουμε την Java για την ύπαρξη της βιβλιοθήκης Lejos. Δεξί click στο on "My Computer" και επέλεξε "Properties". Στη συνέχεια στο Advanced" tab κάνε click στο "**Environment Variables**" κάτω χαμηλά. Δημιούργησε μια νέα "**System variable**" κάνοντας click στο on "New". ΜΗΝ δημιουργήσεις "User variable".



Ονόμασε τη νέα system variable:

LEJOS_HOME

Και δώσε της την τιμή (value):

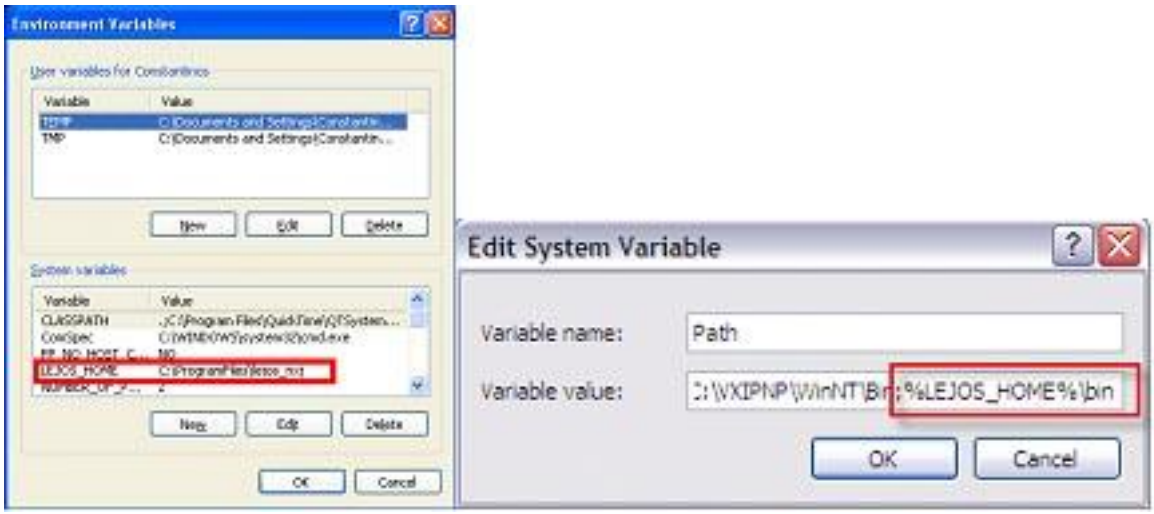
C:\ProgramFiles\lejos_nxj

Το Lejos χρειάζεται να ξέρει που έχει εγκατασταθεί. Πρόσεξε ότι τώρα χρησιμοποιείς το νέο directory "ProgramFiles" το οποίο δεν έχει χαρακτήρα space μεταξύ των λέξεων. Όταν συμπληρώσεις την τιμή πάτησε το "OK".

Έλεγξε ότι η νέα system variable εμφανίζεται στη λίστα. Στη συνέχεια, πρέπει να προσθέσουμε την LEJOS_HOME variable στο system variable "Path". Επέλεξε το "Path" από τη λίστα και κάνε click στο "Edit". Πρόσθεσε

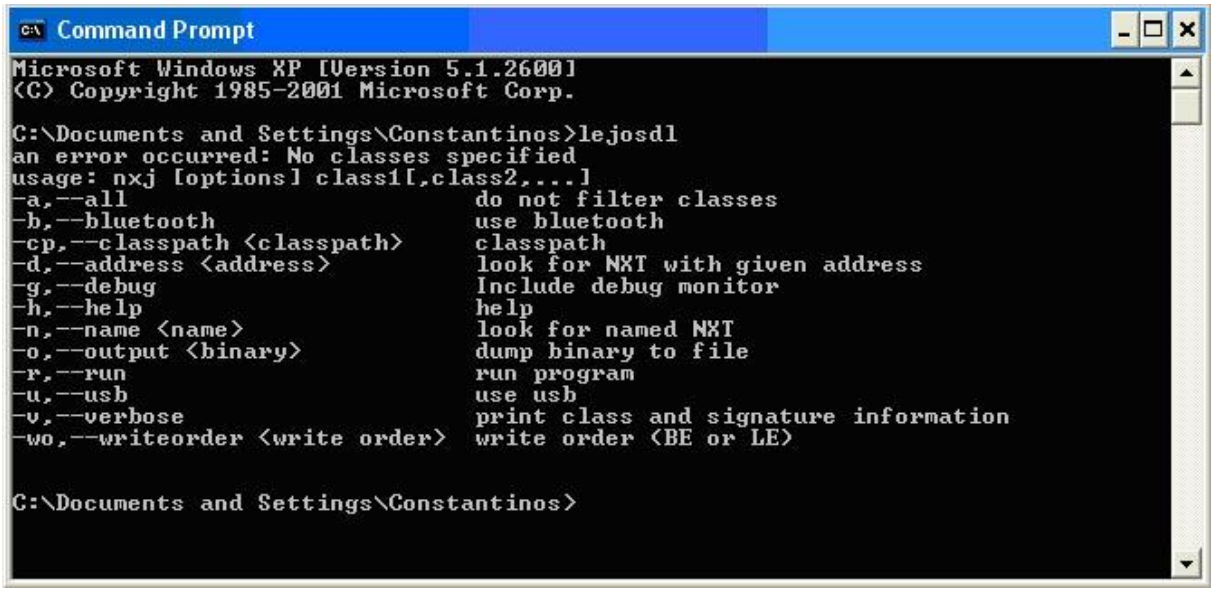
;%LEJOS_HOME%\bin

στο τέλος της τιμής. Οι διαφορετικές τιμές είναι χωρισμένες με ένα ερωτηματικό (;) και δεν χρειάζεται να προστεθεί ερωτηματικό στο τέλος της γραμμής.

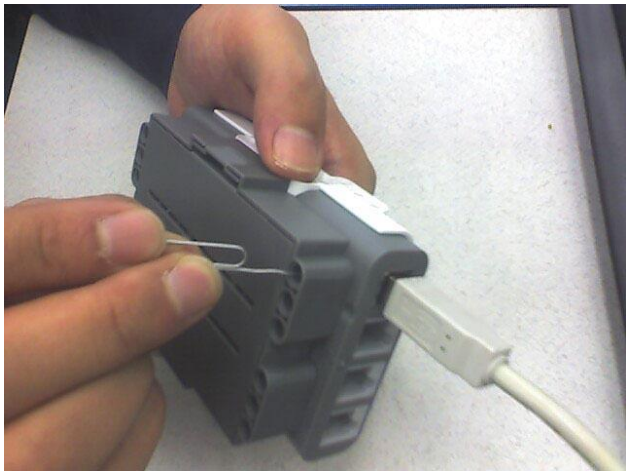


Τώρα θα ελέγξο, αν το Lejos έχει εγκατασταθεί σωστά χρησιμοποιώντας το DOS command prompt. (Κάνε click στο "Start" των Windows και μετά στο "All Programs => Accessories => Command Prompt"). Γράψε μέσα στο command prompt:
Lejosdl

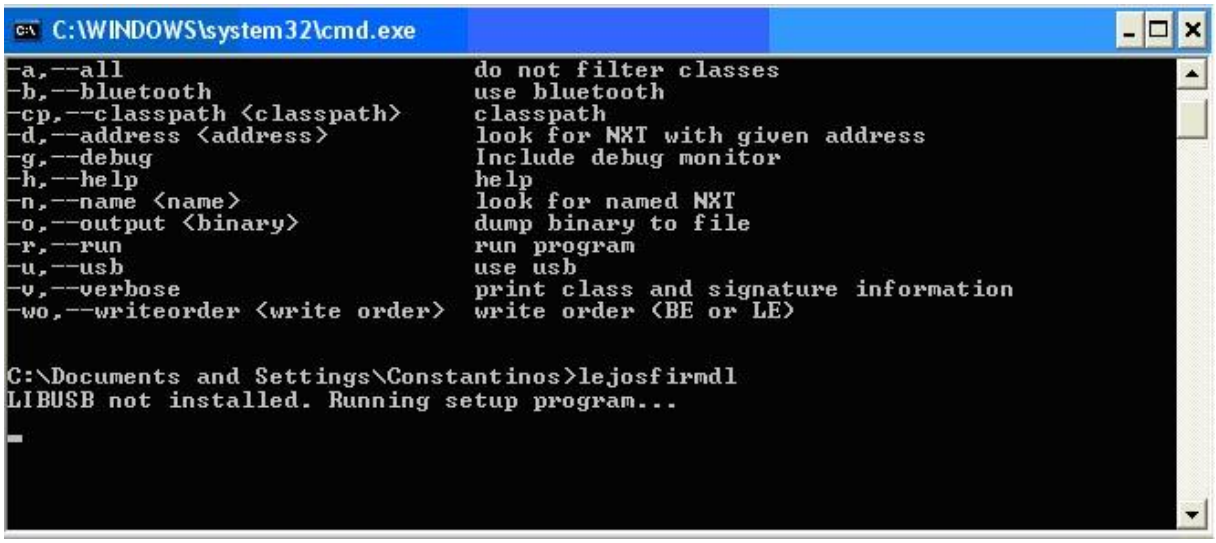
Αν το αποτέλεσμα είναι το ίδιο με αυτό της εικόνας πιο κάτω τότε όλα πήγαν μια χαρά.



Τώρα θα αντικαταστήσουμε το original firmware το οποίο ήρθε μαζί με το NXT με αυτό του Lejos. Το μόνο που χρειάζεται να γίνει είναι να εκτελεστεί το πιο κάτω μια φορά. Το Lejos, μετά από αυτό, θα λειτουργεί ως το λειτουργικό σύστημα του NXT. Υλοποιεί μια Java Virtual Machine πάνω στην οποία μπορούν να τρέξουν προγράμματα Java. Σιγουρέψου ότι το NXT είναι ενωμένο με τον υπολογιστή μέσω ενός καλωδίου USB cable και ότι έχει αναγνωριστεί από το Λειτουργικό Σύστημα. Σε περίπτωση αμφιβολίας έλεγξε το device manager ξανά. Πρώτα πρέπει να φέρεις το NXT στο firmware upload mode. Χρησιμοποίησε ένα paper clip για να πατήσεις και να κρατήσεις πατημένο το κουμπί reset. Το NXT θα παίξει ένα ήχο ο οποίος θα επιβεβαιώνει την ενέργεια.



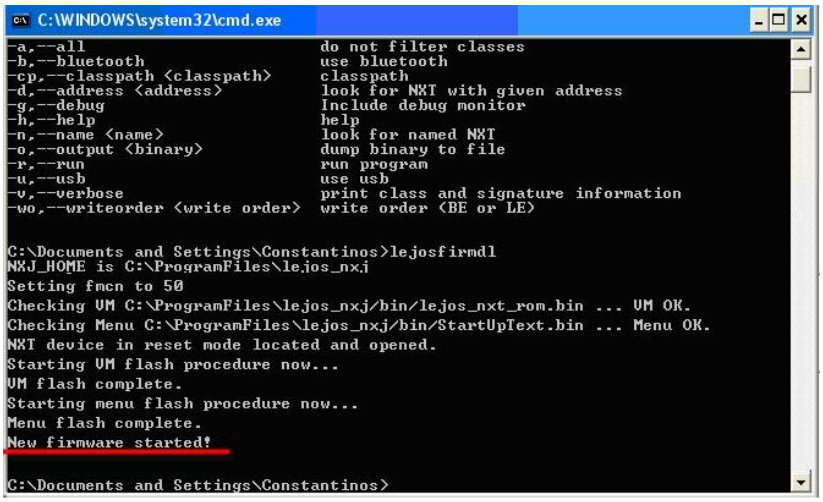
Πληκτρολόγησε στο “Command Prompt” :
lejosfirmdl
και πάτησε ENTER.



Το Lejos απαιτεί από το Libusb να επικοινωνήσει με το NXT. Αν αυτό το δωρεάν λογισμικό δεν έχει εγκατασταθεί πάνω στον υπολογιστή σου, θα αρχίσει αυτόματα η εγκατάσταση του Libusb μέσω του wizard. Το Libusb επιτρέπει στο πρόγραμμα να έχει πρόσβαση σε οποιαδήποτε συσκευή USB στα Windows με generic τρόπο. Σιγουρέψου ότι δεν θα τρέξει το test application μετά το τελευταίο dialog box. Σε περίπτωση που το **LeJOS είναι ήδη εγκατεστημένο στο** NXT **θα πρέπει να εγκαταστήσεις το LibUSB** για να έχεις τη δυνατότητα να κατεβάζεις τα προγράμματά σου στο

NXT. Θα το βρεις στο "lejos_nxj\3rdparty\lib\libusb-win32-filter-bin-0.1.12.1.exe".(Το όνομα μπορεί να αλλάξει αν βγει νέο version του libusb).

Όταν το Libusb εγκατασταθεί, το Lejos firmware installer θα συνεχίσει και θα ολοκληρώσει τη διαδικασία με επιτυχία. Πρέπει τα αποτελέσματα να είναι τα ίδια με αυτά της πιο κάτω εικόνας.



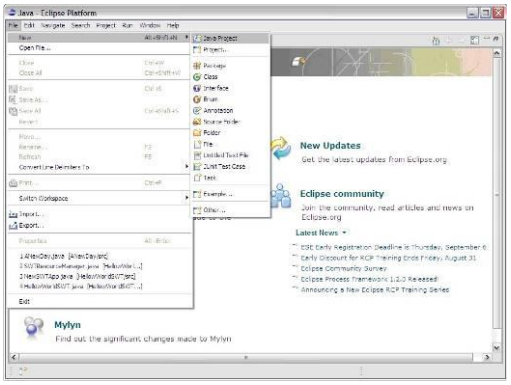
Το NXT ακολούθως θα κάνει reboot and θα εμφανιστεί το Lejos Logo και ακολούθως το main menu.

Το NXT είναι τώρα έτοιμο να τρέξει Java προγράμματα.

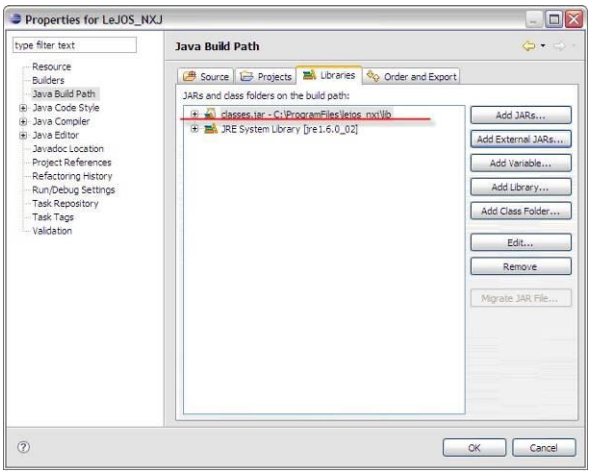
Εγκατάσταση και ρύθμιση του Eclipse στον υπολογιστή

Κατεβάστε το Eclipse IDE για Java Developers.Το Eclipse είναι γραμμένο σε γλώσσα Java και δεν περιέχει installer (setup.exe). Έρχεται σαν ένα απλό ZIP file το οποίο πρέπει να κάνεις unzip στο directory που δημιούργησες πιο πριν (“ProgramFiles”). Αν θέλεις, μπορείς να δημιουργήσεις ένα shortcut του “eclipse.exe” στο desktop για δική σου ευκολία.
Όταν ξεκινήσεις το Eclipse, θα σου ζητηθεί να διαλέξεις ένα workspace. Το workspace θα περιέχει όλα τα files that που θα δημιουργείς. Καλό θα ήταν να δημιουργήσεις το directory σε ένα τόπο όπου το path δεν θα περιέχει κανένα χαρακτήρα space. Στο παράδειγμα χρησιμοποιείται το directory “d:\programming\workspace”. Στη συνέχεια θα εμφανιστεί η οθόνη καλωσορίσματος.

Για τη δημιουργία ενός νέου project κάνουμε click στο “File => New => Java Project”. Δίνουμε στο project ένα νέο όνομα το οποίο να μην περιέχει κενούς χαρακτήρες (πχ “LeJOS_NXJ”)

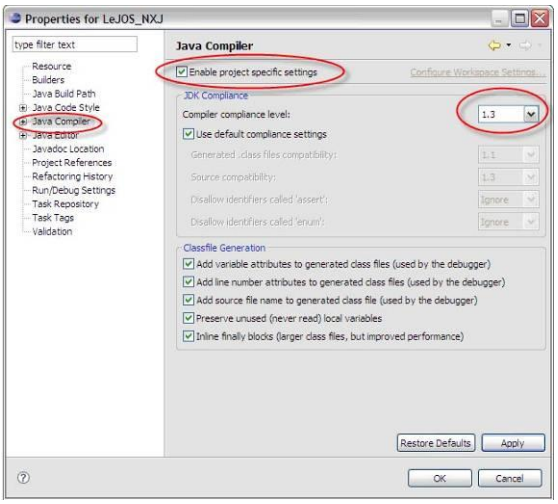


Ακολούθως αυτό το project πρέπει να μετατραπεί σε Lejos project. Κάνε right-click πάνω στο project και επέλεξε το "Properties". Στη συνέχεια επέλεξε το "Java Build Path" που βρίσκεται στα αριστερά και κάνε click στο "Libraries" tab. Μετά κάνε click στο "Add External JARs…" και βρες το directory "lib" στο directory "ProgramFiles\lejos_nxj". Επέλεξε το "classes.jar" και πάτησε "Open". Η Lejos library θα εμφανιστεί στη λίστα:

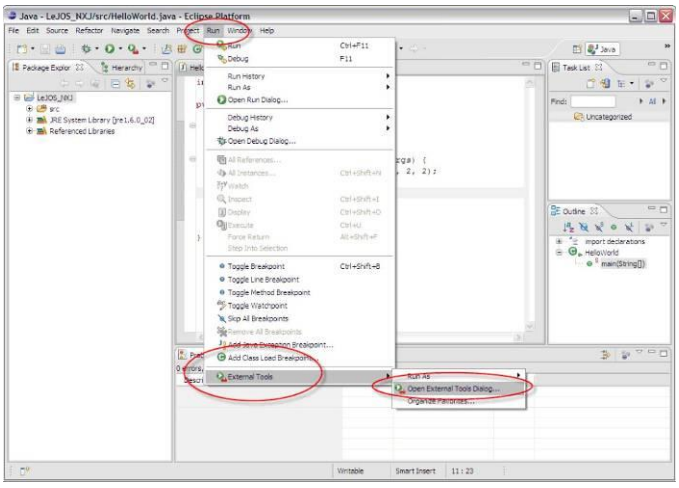


Στη συνέχεια μετακινούμαστε στο section "Java Compiler" το οποίο βρίσκεται στο αριστερό panel. Βάλε τικ στο κουτάκι "Enable project specific settings" και επέλεξε level 1.3 για το compiler compliance level. Αυτό θα αναγκάσει τον compiler να κάνει optimize τον κώδικα σε ένα πιο παλιό version της Java. Αυτό το παλιότερο version είναι πιο κατάλληλο για το NXT, γιατί είχε δημιουργηθεί για embedded systems. Χρειάζεται

πολύ λιγότερα resources από το τελευταίο version της Java. Κάνε click στο “Apply” και μετά στο”OK” για να βγεις από τα properties



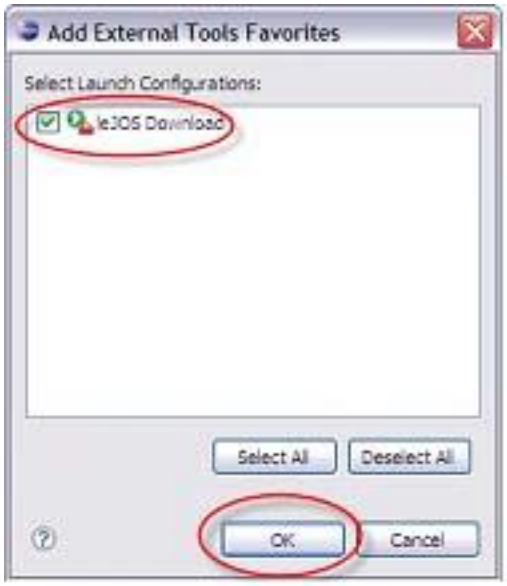
Τώρα πρέπει να ρυθμιστεί το Eclipse για να κατεβάσει το software στο NXT. Κάνε click στο “Run => External Tools => Open External Tools Dialog…”.



Επέλεξε “Program” στα αριστερά και μετά κάνε click στο εικονίδιο “New”. Ονόμασε το εργαλείο “leJOS Download”. Στο “Main” tab, γράψε την τοποθεσία του αρχείου “lejosdl.bat” το οποίο πρέπει να βρίσκεται στο directory “lejos-nxj\bin\” κάνοντας click στο “Browse File System”. Στη συνέχεια, γράψε

${project_loc}\bin
μέσα στο "Working directory" και ακολούθως γράψε
${java_type_name}
στο Arguments section.

Τώρα θα δημιουργήσουμε ένα shortcut με τις ρυθμίσεις για το κατέβασμα του προγράμματος στο NXT.Κάνε click στο μικρό πράσινο εικονίδιο "run" και μετά επέλεξε "Organize Favorites…". Κάνε click στο "Add…" στο popup window και κάνε check το κουτάκι μπροστά από το "leJOS Download", Το οποίο είναι το εξωτερικό εργαλείο το οποίο ρυθμίσαμε στα προηγούμενα βήματα. Πάτησε "OK" και μετά ξανά "Ok".

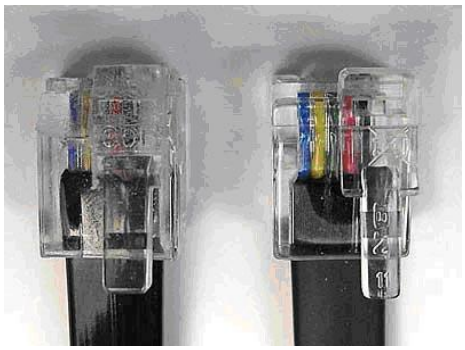


Τώρα είσαι σε θέση να γράψεις κώδικα Java στο Eclipse και πατώντας το "run" να τον μεταγλωττίσεις και να τον εγκαταστήσεις πάνω στο NXT.

Παράρτημα Δ

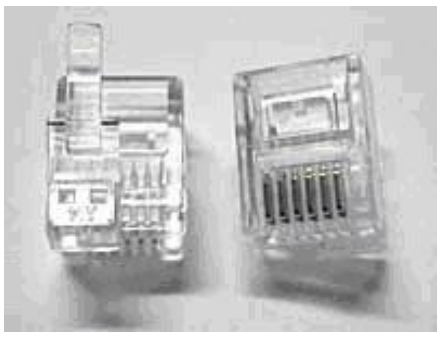
Δημιουργία καλωδίου NXT από καλώδιο τηλεφώνου με RJ-12 socket

Για να ενώσω το NXT brick και τον αισθητήρα micaz με ένωση I2C έπρεπε να καταστρέψω ένα καλώδιο του lego NXT κόβοντας το socket από τη μια του μεριά, ώστε να μπορώ να χρησιμοποιήσω ξεχωριστά καθένα από τα μικρά καλώδια που το αποτελούν.



Όμως επειδή δεν είχαμε περίσσευμα NXT καλωδίων αποφάσισα να τροποποιήσω καλώδιο τηλεφώνου και να το χρησιμοποιήσω στη θέση του καλωδίου NXT.

Πιο κάτω βλέπουμε πως είναι το socket του καλωδίου NXT του οποίου η ασφάλεια του βρίσκεται στα αριστερά του socket.



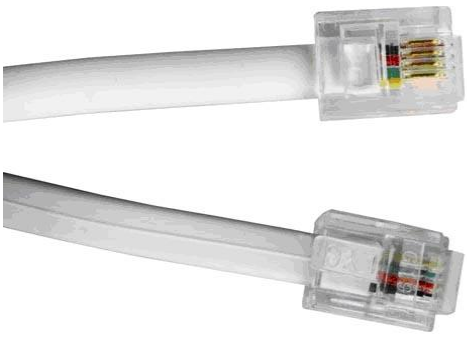
Πιο κάτω βλέπουμε πως είναι το RJ-12 socket του οποίου η ασφάλεια του βρίσκεται στο κέντρο του socket και είναι ελάχιστα πιο πλατιά.

Επειδή δεν είχα στη διάθεση μου καλώδιο τηλεφώνου, χρησιμοποίησα καλώδιο Ethernet (cat6) το οποίο έχει στο εσωτερικό του 8 καλώδια . Η διαδικασία είναι η ίδια, απλά χρησιμοποίησα 6 από τα 8 καλώδια που διέθετε.

Απαιτούμενα υλικά:

- 2 RJ-12 socket.
- Καλώδιο τηλεφώνου με 6 χρωματιστά καλώδια στο εσωτερικό του.
- 1 crimping tool (6p).
- 1 epoxy glue
- Λεπτό πριόνι.
- Λίμα

Αρχικά, ένωσα το καλώδιο με RJ-12 sockets και στις δύο μεριές του. Πέρασα στο socket 6 από τα 8 καλώδια του καλωδίου Ethernet. Εναλλακτικά θα μπορούσα να είχα αγοράσει ένα έτοιμο καλώδιο τηλεφώνου με RJ-12 sockets.

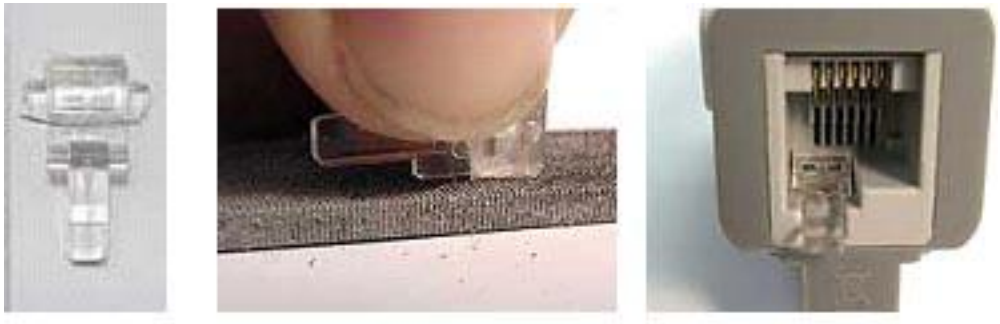


Πιο κάτω βλέπουμε το καλώδιο που δημιούργησα δίπλα σε ένα καλώδιο NXT

Τώρα πρέπει να διορθωθεί η ασφάλεια ώστε το custom - made καλώδιο να ταιριάζει στην υποδοχή του NXT brick και στα sensors.

Με το λεπτό πριόνι έκοψα την ασφάλεια ακολουθώντας την κόκκινη γραμμή όπως φαίνεται στην πιο κάτω εικόνα.

Ακολούθως λίμαρα τις δυο πλευρές της ασφάλειας ώστε να μειώσω το πλάτος της για να χωράει στην υποδοχή του NXT brick και των sensors του.

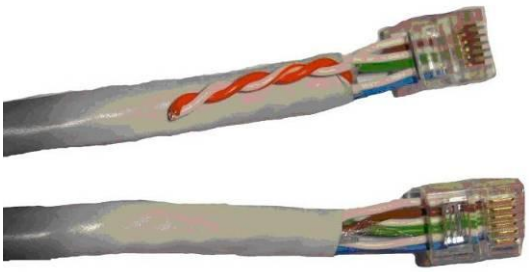


Όταν η ασφάλεια έφτασε στο σωστό μέγεθος την κόλλησα πίσω πάνω στο socket χρησιμοποιώντας epoxy glue, αλλά αντί να την τοποθετήσω στο κέντρο την τοποθέτησα στο δεξί άκρο, όπως είναι το socket του NXT καλωδίου. Στερέωσα την ασφάλεια στο socket με κολλητική ταινία για να μην μετακινηθεί κατά την επικόλληση της, όπως φαίνεται στην πιο κάτω εικόνα.

Όταν στέγνωσε η κόλλα ένωσα το καλώδιο με το NXT brick και ένα από τα 3 NXT motors για να ελέγξω τη λειτουργικότητά του.

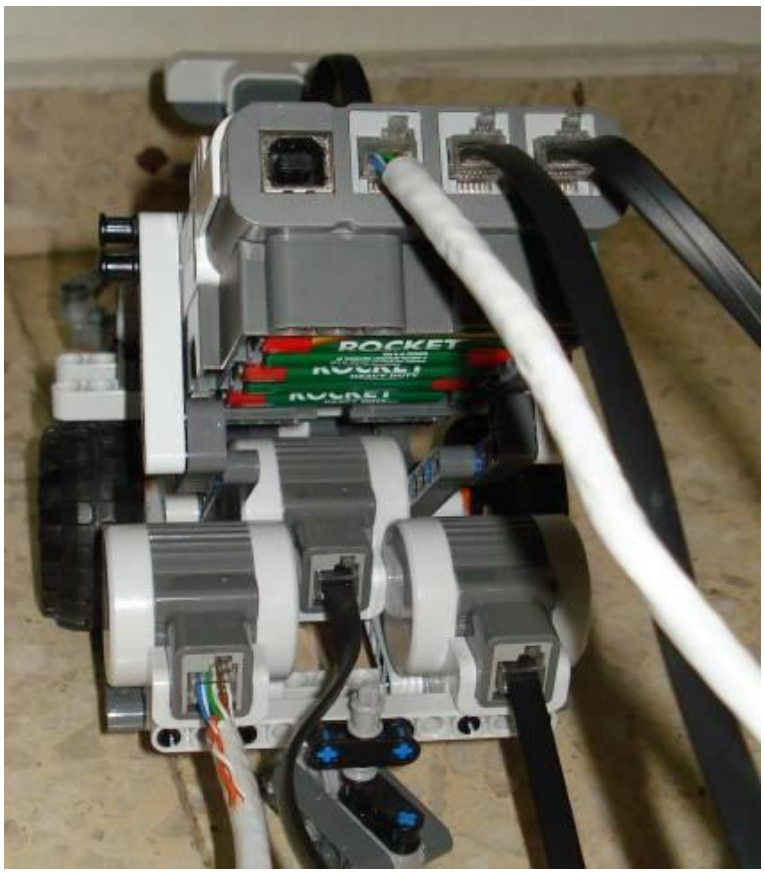
Αντιστοιχία χρωματισμού custom made καλωδίου με το καλώδιο NX

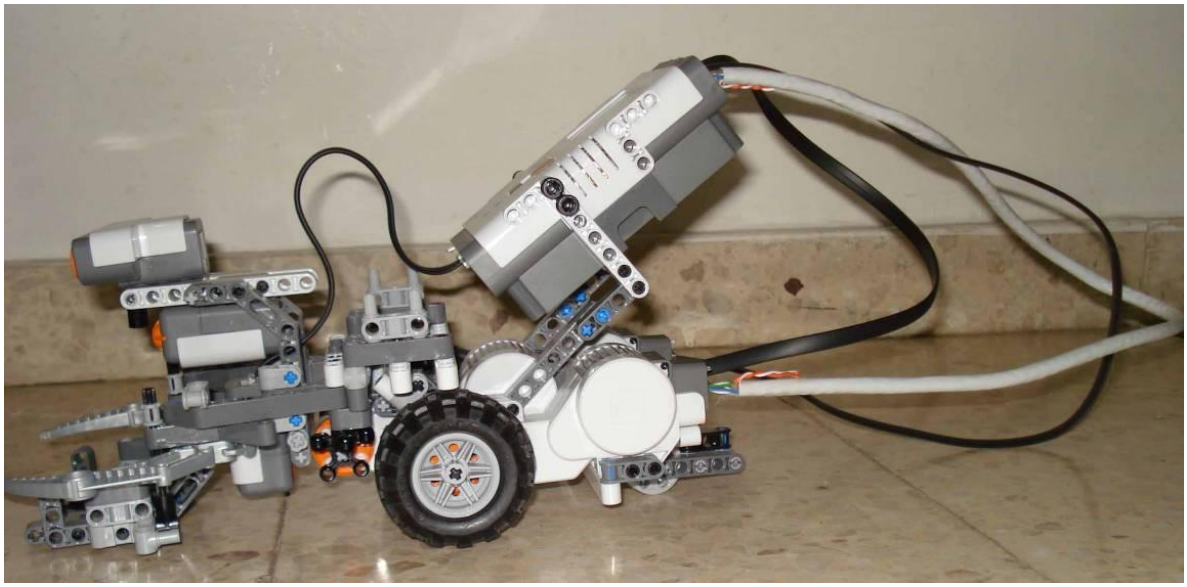
Καλώδιο NXT	**Custom-made καλώδιο**
μπλε	**μπλε**
κίτρινο	**άσπρο-μπλε**
πράσινο	**πράσινο**
κόκκινο	**άσπρο-πράσινο**
μαύρο	**καφέ**
άσπρο	**άσπρο-καφέ**



Η δημιουργία καλωδίου NXT από καλώδιο τηλεφώνου με RJ-12 socket έγινε με επιτυχία.

Στις πιο κάτω εικόνες φαίνεται η εφαρμοφή του καλωδίου πάνω στις υποδοχές του robot:





Το καλώδιο δουλεύει σωστά χωρίς κανένα απολύτως πρόβλημα.

Πιο κάτω παρουσιάζω τον κώδικα σε γλώσσα lejos για το γράψιμο του μηνύματος “Hello my master” στην οθόνη.

Κώδικας γραμμένος σε lejos για επικοινωνία του NXT με την LCD οθόνη

```
package i2c2;
import lejos.nxt.I2CSensor;
import lejos.nxt.LCD;
import lejos.nxt.SensorPort;
import lejos.nxt.*;

public class Hello {
        /**
         * @param args
         */
        public static void main(String[] args) throws Exception{
                // TODO Auto-generated method stub
                I2CSensor i2c = new I2CSensor(SensorPort.S1);
                i2c.setAddress(0x63);
                i2c.sendData(0, (byte)12);
                i2c.sendData(0, (byte)72);        //H
                i2c.sendData(0, (byte)101);       //e
                i2c.sendData(0, (byte)108);       //l
                i2c.sendData(0, (byte)108);       //l
                i2c.sendData(0, (byte)111);       //o
```

```
                i2c.sendData(0, (byte)32);   // (space)

                i2c.sendData(0, (byte)109);      //m
                i2c.sendData(0, (byte)121);      //y

                i2c.sendData(0, (byte)32);      //(space)

                i2c.sendData(0, (byte)109);      //m
                i2c.sendData(0, (byte)97);      //a
                i2c.sendData(0, (byte)115);      //s
                i2c.sendData(0, (byte)116);      //t
                i2c.sendData(0, (byte)101);      //e
                i2c.sendData(0, (byte)114);      //r

                System.out.println("BY Kotsios");
        //LCD.drawString(Integer.toString(5), 0, 0);
        Thread.sleep(3000);
        }
}
```