Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
στα Πληροφοριακά Συστήματα

Μεγιστοποίηση Διάρκειας Αξιοποίησης Ασύρματων Δικτύων Αισθητήρων (Wireless Sensor Networks)

Δημήτριος Ξηρομερίτης

Επιβλέπουσα Καθηγήτρια
Μαρία Ανδρέου

Ιανουάριος 2013

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεγιστοποίηση Διάρκειας Αξιοποίησης Ασύρματων Δικτύων Αισθητήρων (Wireless Sensor Networks)

Δημήτριος Ξηρομερίτης

Επιβλέπουσα Καθηγήτρια
Μαρία Ανδρέου

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Ιανουάριος 2013

Περιεχόμενα

Κεφάλαιο 1ο

Εισαγωγή

1.1 Γενική περιγραφή - Ορισμοί

Η εξέλιξη των τελευταίων χρόνων στο χώρο των ασύρματων επικοινωνών αλλά και των ηλεκτρονικών συσκευών, έχει ανοίξει το δρόμο για τη δημιουργία συσκευών μικρού μεγέθους και μικρού κόστους, με τη δυνατότητα να κάνουν μετρήσεις διαφόρων φυσικών μεγεθών από το περιβάλλον (θερμοκρασία, υγρασία κλπ), να κάνουν υπολογισμούς πάνω στις μετρήσεις αυτές, να επικοινωνούν ασύρματα μεταξύ τους αλλά και με κάποιον κεντρικό κόμβο, στον οποίο και να αποστέλλουν δεδομένα. Επιπλέον, μπορούν να λαμβάνουν δεδομένα από τον κεντρικό αυτό κόμβο, τον οποίο από του εξής θα ονομάζουμε Σταθμό Βάσης. Τις συσκευές αυτές, τις ονομάζουμε αισθητήρες. Οι συσκευές αυτές μπορούν να ομαδοποιούνται και να αυτο-οργανώνονται δημιουργώντας ένα ασύρματο δίκτυο. Μια συλλογή από τέτοιες συσκευές την ονομάζουμε Ασύρματο Δίκτυο Αισθητήρων (από του εξής: ΑΔΑ)[1]

Το ότι οι αισθητήρες έχουν χαμηλό κόστος, μικρό μέγεθος και τοποθετούνται στο χώρο σχετικά εύκολα, τους καθιστά κατάλληλους για εφαρμογές όπου χρειαζόμαστε να επιτηρήσουμε ή να πάρουμε μετρήσεις από μια μεγάλη γεωγραφική περιοχή [2].

Με βάση, λοιπόν, τα προαναφερθέντα, αντιλαμβανόμαστε ότι τα ΑΔΑ έχουν εφαρμογές σε πολλούς τομείς, όπως: μέτρηση και παρατήρηση φυσικών μεγεθών σε κάποια γεωγραφική περιοχή (θερμοκρασία, υγρασία, πίεση κλπ), στην υγεία, στις στρατιωτικές επιχειρήσεις (πεδία μάχης κλπ), στις μεταφορές, στην πρόβλεψη του καιρού, στην ανίχνευση κίνησης κλπ [3]

Ένα ΑΔΑ, είναι μια συλλογή από τέτοιους αισθητήρες (οι οποίοι αποτελούν και τους κόμβους του δικτύου), που έχουν τη δυνατότητα να «αισθάνονται» το περιβάλλον τους, να κάνουν υπολογισμούς και να επικοινωνούν μεταξύ τους αλλά και με το Σταθμό Βάσης.

Με βάση τα πιο πάνω, ίσως κάποιος να υποθέσει ότι τα ΑΔΑ είναι αρκετά όμοια με τα «κλασικά» ενσύρματα ή ασύρματα δίκτυα, απλά οι κόμβοι είναι κάποιας ειδικής μορφής. Τα πράγματα όμως δεν είναι ακριβώς έτσι. Στα παραδοσιακά δίκτυα δεν έχουμε

περιορισμούς ούτε στην ενέργεια (μπαταρία), ούτε στην υπολογιστική ισχύ, ούτε στη μνήμη που διαθέτει ο κάθε κόμβος, ούτε και στην επικοινωνία. Συνήθως, έχουμε κεντρική δομή σταθερού δικτύου (backbone) και μεγάλη ακτίνα επικοινωνίας. Ενώ στα ΑΔΑ, οι κόμβοι-αισθητήρες έχουν περιορισμένους ενεργειακούς πόρους, μικρότερη επεξεργαστική ισχύ, λιγότερη διαθέσιμη μνήμη και περιορισμένη ακτίνα επικοινωνίας [9][3].

Έτσι, υπάρχουν αρκετές τεχνολογικές «προκλήσεις», όσο αφορά το σχεδιασμό των ΑΔΑ και τη λειτουργικότητα που θα επιτελούν. Αυτές οι προκλήσεις περιλαμβάνουν:

- την αποδοτική διαχείριση της επικοινωνίας,
- τη δυνατότητα ενός ΑΔΑ να λειτουργεί χωρίς επίβλεψη (unattended),
- την αύξηση του χρόνου ζωής του δικτύου (network lifetime)
- την ανοχή στα λάθη (fault-tolerance) κ.α. [10].

Για να βελτιώσουμε την απόδοση ενός ΑΔΑ, θα πρέπει να εξετάσουμε όλους τους πιο πάνω παράγοντες. Το μείζον, όμως, θέμα που θα πρέπει να διαχειριστούμε είναι η όσο το δυνατόν καλύτερη αξιοποίηση των πόρων του δικτύου. Και όταν λέμε «πόρων», εννοούμε τόσο την ενέργεια, όσο και το εύρος ζώνης της ασύρματης επικοινωνίας.

Στα ΑΔΑ, κάθε κόμβος-αισθητήρας επικοινωνεί με το Σταθμό Βάσης (όχι πάντα απευθείας, αλλά και μέσω πολλαπλών «αλμάτων»-hops), ο Σταθμός Βάσης με τον αισθητήρα αλλά και οι αισθητήρες μεταξύ τους, με τελικό σκοπό οι μετρήσεις που πραγματοποιούνται στους αισθητήρες να μεταδοθούν στο Σταθμό Βάσης, όπου μπορεί να γίνει μια περεταίρω επεξεργασία και (ίσως) τελικά η αποστολή των δεδομένων αυτών μέσω του internet στον τελικό χρήστη-administrator ενός τέτοιου δικτύου. Έτσι, για να μπορέσουμε να βελτιστοποιήσουμε την επικοινωνία θα πρέπει να επιλέξουμε αποδοτικά πρωτοκόλλα δρομολόγησης των δεδομένων αλλά και ταυτόχρονα να έχουμε τη δυνατότητα συντήρησης των μονοπατιών δρομολόγησης (routing paths)[7].

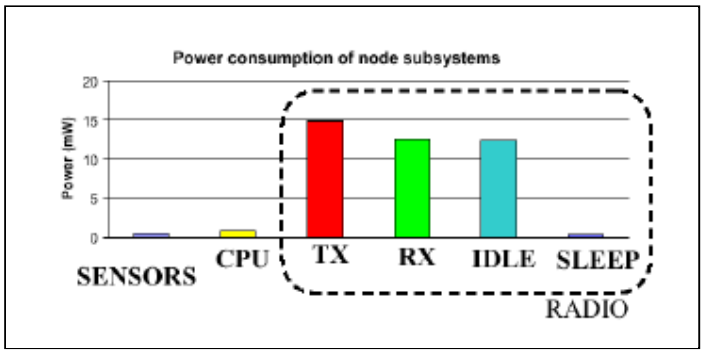
1.2 Σκοπός της εργασίας

Όπως φαίνεται και από τίτλο της παρούσης εργασίας, σκοπός μας είναι να εξετάσουμε με ποιον τρόπο θα μπορέσουμε να επιτύχουμε τη μεγιστοποίηση του χρόνου αξιοποίησης ενός ΑΔΑ. Ως «χρόνο αξιοποίησης» ή «χρόνο ζωής» (network lifetime) ενός ΑΔΑ θα μπορούσαμε να ορίσουμε το χρονικό διάστημα κατά το οποίο το δίκτυο παραμένει λειτουργικό. Σύμφωνα με το [35], ως χρόνος ζωής ενός ΑΔΑ, ορίζεται το χρονικό διάστημα μέχρις ότου ένα ποσοστό α% από τους κόμβους τεθεί εκτός λειτουργίας λόγω εξάντλησης

των ενεργειακών τους πόρων (μπαταρίας). Το ποσοστό αυτό (α%) ορίζεται από το σχεδιαστή του δικτύου. Για παράδειγμα, σε εφαρμογές όπου είναι σημαντικό να λειτουργούν όλοι οι αισθητήρες ενός ΑΔΑ, ως χρόνο ζωής του δικτύου ορίζουμε το χρονικό διάστημα μέχρι να «πεθάνει» ο πρώτος κόμβος. Στην περίπτωση αυτή το α θα ήταν 1/n, όπου n ο αριθμός των κόμβων του δικτύου. Ένας τρόπος για να μπορέσουμε να μεγιστοποιήσουμε αυτό το χρονικό διάστημα, είναι να εφαρμόσουμε κατάλληλα πρωτόκολλα δρομολόγησης έτσι ώστε να εξισορροπήσουμε την κατανάλωση ενέργειας σε όλους τους κόμβους. Όπως θα παρουσιάσουμε και στη συνέχεια της εργασίας, ο κάθε κόμβος ξοδεύει ενέργεια για να επιτελέσει 3 εργασίες:

- να κάνει μετρήσεις φυσικών μεγεθών του περιβάλλοντος (sensing)
- να κάνει υπολογισμούς (computing)
- να επικοινωνήσει με τους υπόλοιπους κόμβους αλλά και με το σταθμό βάσης. (communication)

Από τις 3 αυτές εργασίες, αυτή που απαιτεί την περισσότερη ενέργεια είναι η ασύρματη επικοινωνία [4]. Η ενέργεια που ξοδεύεται για τη λήψη των μετρήσεων από τον αισθητήρα και για την επεξεργασία των δεδομένων είναι πάρα πολύ μικρή. Παραδείγματος χάριν, η ενέργεια που καταναλώνεται από έναν αισθητήρα Berkley mote [22], για τη μετάδοση 1 bit δεδομένων είναι ισοδύναμη με την ενέργεια που απαιτείται για την εκτέλεση 800 εντολών του επεξεργαστή [24][25]. Όπως βλέπουμε και στην εικ.1, η κατανάλωση ενέργειας για τις καταστάσεις μετάδοσης (Tx, transmit), λήψης (Rx, receive) και αδράνειας (Idle) είναι πολλαπλάσια από την ενέργεια για υπολογισμούς (CPU).



Εικ.1: Κατανάλωση ενέργειας σε έναν αισθητήρα όπως περιγράφεται στο [22]

Η εργασία περιλαμβάνει τα πιο κάτω κεφάλαια: Κεφάλαιο 2, όπου θα αναφερθούμε σε διάφορες εφαρμογές των Ασύρματων Δικτύων Αισθητήρων. Κεφάλαιο 3, όπου θα περιγράψουμε και θα αναλύσουμε την αρχιτεκτονική των ΑΔΑ, τόσο σε επίπεδο κόμβου-αισθητήρα, όσο και σε επίπεδο δόμησης ολόκληρου του δικτύου. Συγκεκριμένα θα δούμε με ποιον τρόπο σχετίζεται η τοπολογία του δικτύου με τη δρομολόγηση των δεδομένων στο συγκεκριμένο δίκτυο. Στο κεφάλαιο 4 θα μελετήσουμε τα πρωτόκολλα δρομολόγησης στα ΑΔΑ. Συγκεκριμένα, θα παρουσιάσουμε τα σημαντικότερα πρωτόκολλα δρομολόγησης σε ιεραρχικά δομημένα δίκτυα, και ειδικότερα σε δίκτυα δομημένα σε συστάδες (cluster based) και σε δίκτυα με δενδρική δομή (tree based).

Κεφάλαιο 2ο

Εφαρμογές των Ασύρματων Δικτύων Αισθητήρων

2.1 Κατηγορίες Αισθητήρων που χρησιμοποιούνται στα ΑΔΑ

Με τη χρήση των ΑΔΑ είναι σίγουρο ότι όχι μόνο θα διευκολυνθούν πολλές υπάρχουσες εφαρμογές, αλλά και ότι θα δημιουργηθούν εντελώς νέοι τομείς εφαρμογών.

Για τα περισσότερα φυσική μεγέθη, υπάρχει ήδη η κατάλληλη τεχνολογία που απαιτείται για την κατασκευή των αισθητήρων. Έτσι, έχουν κατασκευαστεί αισθητήρες που: [12][13]

- Μετρούν τη θερμοκρασία είτε του περιβάλλοντος είτε κάποιου αντικειμένου (θερμικοί αισθητήρες).
- Μετρούν το ποσοστό υγρασίας (αισθητήρες υγρασίας).
- Ανιχνεύουν ακουστικό σήμα (ηχητικοί αισθητήρες).
- Ανιχνεύουν το φως αλλά και την υπέρυθρη ακτινοβολία (οπτικοί, υπέρυθροι αισθητήρες).
- Ανιχνεύουν δονήσεις (για ανίχνευση σεισμικών διαταραχών – αισθητήρες δόνησης)
- Ανιχνεύουν διάφορους τύπους αερίων ή και τη σύσταση του εδάφους (χημικοί αισθητήρες).
- Μηχανικής αντοχής, μαγνητικοί (για την ανίχνευση διέλευσης οχημάτων) κ.α.

Οι παραπάνω ικανότητες αίσθησης-μέτρησης φυσικών μεγεθών σε συνδυασμό με την ικανότητα επικοινωνίας μεταξύ τους, αλλά και της επεξεργασίας δεδομένων, καθιστούν τους αισθητήρες κατάλληλους για μια ευρεία γκάμα εφαρμογών. Μια εφαρμογή μπορεί να χρησιμοποιεί οποιονδήποτε τύπο αισθητήρα ή και πολλούς τύπους αισθητήρων

ταυτόχρονα. Στη συνέχεια θα παρουσιάσουμε διάφορους τύπους εφαρμογών, πολλές από τις οποίες έχουν ήδη υλοποιηθεί.

2.2 Στρατιωτικές εφαρμογές των ΑΔΑ

Η γρήγορη τοποθέτηση, η ικανότητα αυτο-οργάνωσης των κόμβων–αισθητήρων σε δίκτυο καθώς και η ανοχή στα σφάλματα, είναι τα βασικά χαρακτηριστικά των ΑΔΑ που τα καθιστούν κατάλληλα για στρατιωτικές εφαρμογές. [11]

Πιο συγκεκριμένα, μπορούμε να χρησιμοποιήσουμε ΑΔΑ για:

Παρακολούθηση στρατιωτικών δυνάμεων, στρατιωτικού εξοπλισμού και πυρομαχικών: κάθε στρατιώτης, όχημα, εξοπλισμός και σημαντικό πυρομαχικό μπορεί να εφοδιαστεί με έναν αισθητήρα, έτσι ώστε οι αξιωματικοί να μπορούν να είναι ενημερωμένοι για τη διαθεσιμότητα τους.

Παρακολούθηση πεδίου μάχης: κάθε κρίσιμο πεδίο μάχης, καθώς και οι διαδρομές πρόσβασης στα στρατόπεδα μπορούν να παρακολουθούνται μέσω αισθητήρων, έτσι ώστε οι κινήσεις των εχθρικών δυνάμεων να γίνονται άμεσα αντιληπτές.

Στόχευση: οι αισθητήρες μπορούν να ενσωματωθούν στα συστήματα οδήγησης «έξυπνων» πυρομαχικών.

Αποτίμηση ζημιάς μάχης: μετά το πέρας μιας μάχης, μπορεί να εγκατασταθεί ένα ΑΔΑ στην περιοχή, για να γίνει συλλογή δεδομένων σχετικά με τις ζημιές που έχει υποστεί το στράτευμα και ο εξοπλισμός.

Ανίχνευση και αναγνώριση πυρηνικής, βιολογικής και χημικής επίθεσης: σε περίπτωση βιολογικής ή χημικής επίθεσης έχει μεγάλη σημασία για την έγκαιρη ενημέρωση, η ύπαρξη πληροφοριών από το σημείο της επίθεσης. Τα ΑΔΑ μπορούν να λειτουργήσουν σαν συστήματα ανίχνευσης τέτοιων επιθέσεων συμβάλλοντας έτσι στην ασφάλεια των πολιτών. Επιπλέον, μπορεί να ανιχνευθεί το είδος της επίθεσης έτσι ώστε να οργανωθεί η αντιμετώπιση της κατάστασης με τον καλύτερο τρόπο. Π.χ. διαφορετική αντιμετώπιση σε περίπτωση ραδιενέργειας και διαφορετική σε περίπτωση βιολογικών όπλων.

2.3 Περιβαλλοντικές εφαρμογές

Σε σχέση με το περιβάλλον θα μπορούσαμε να έχουμε εφαρμογές όπως: [11]

Ανίχνευση πυρκαγιάς σε δάσος: οι κόμβοι-αισθητήρες που χρησιμοποιούνται σε μια τέτοια εφαρμογή, θα μπορούσαν να ήταν θερμικοί αισθητήρες. Να μετρούν τη θερμοκρασία του περιβάλλοντος και όταν αυτή υπερβεί κάποιο κρίσιμο όριο να στέλνουν στο Σταθμό Βάσης την πληροφορία αυτή αλλά και τη θέση τους, είτε σε σχέση με γειτονικούς κόμβους είτε σε απόλυτες γεωγραφικές συντεταγμένες. Τους κόμβους μπορούμε να τους διασκορπίσουμε είτε από κάποιο εναέριο μέσο (αεροσκάφος, ελικόπτερο), είτε να τους τοποθετήσουμε με το χέρι σε συγκεκριμένα σημεία. Έτσι μπορεί να δημιουργηθεί κάτι σαν «χάρτης θερμοκρασίας» του δάσους. Αν δε, το δίκτυο έχει μεγάλη πυκνότητα κόμβων τότε μπορούμε να εντοπίσουμε την εστία της φωτιάς αρκετά νωρίς και με ακρίβεια, και να την περιορίσουμε προτού εξαπλωθεί. Στην κατηγορία αυτή των εφαρμογών, οι κόμβοι-αισθητήρες μπορούν να είναι εφοδιασμένοι με ηλιακούς συλλέκτες [14], για να έχουν μεγαλύτερη διάρκεια ζωής.

Ανίχνευση πλημμύρας: [17] στις ΗΠΑ εφαρμόζεται ήδη το σύστημα ALERT [18] για την ανίχνευση πλημμυρών. Το σύστημα αποτελείται από αισθητήρες για τη βροχόπτωση, τη στάθμη του νερού και τις μετεωρολογικές συνθήκες. Όλοι οι κόμβοι στέλνουν τα αντίστοιχα δεδομένα σε μια τελική βάση δεδομένων, όπου γίνεται η αξιολόγησή τους και η λήψη αποφάσεων.

Εφαρμογές στη γεωργία: σε ό,τι αφορά τη γεωργία, μπορούμε να έχουμε αποτελεσματικότερη άρδευση αλλά και λίπανση του εδάφους με χρήση αισθητήρων οι οποίοι να μετρούν την υγρασία και τη σύνθεση του εδάφους. Επιπλέον τα ΑΔΑ μπορούν να παρακολουθούν τα επίπεδα διαφόρων χημικών ουσιών, όπως εντομοκτόνα ή φυτοφάρμακα στο πόσιμο νερό, της διάβρωσης του εδάφους αλλά και της μόλυνσης του αέρα, και όλα αυτά σε πραγματικό χρόνο. Όσο δε αφορά την κτηνοτροφία, θα μπορούσαμε να εφοδιάσουμε κάθε εκτρεφόμενο ζώο (π.χ. αγελάδα), με αισθητήρες που να μετρούν διάφορα ζωτικά σημεία του ζώου (π.χ. θερμοκρασία).

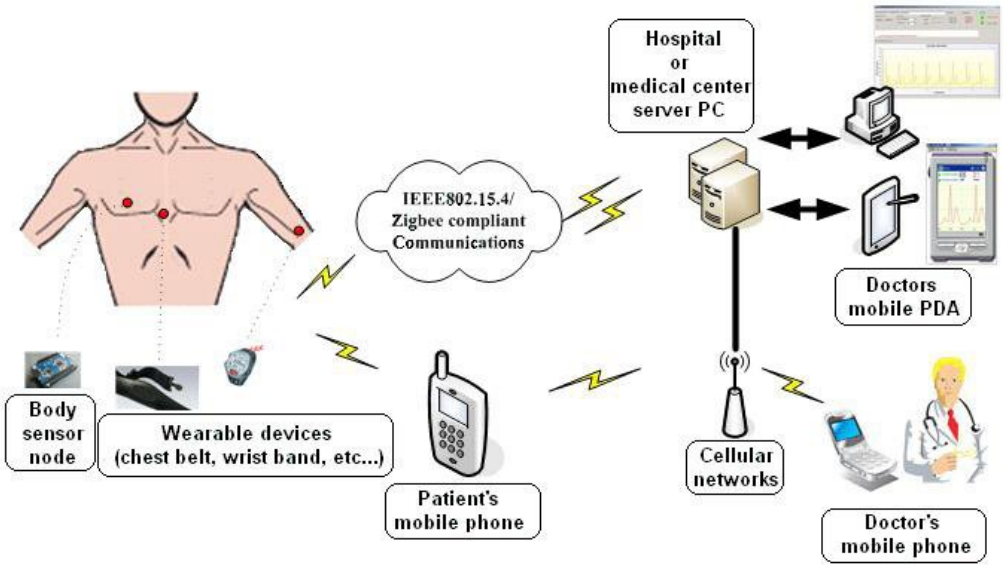
2.4 Εφαρμογές στην υγεία

Ορισμένες εφαρμογές των ΑΔΑ στο χώρο της υγείας αφορούν τη διάγνωση ασθενειών, την παρακολούθηση ασθενών, τη χορήγηση φαρμάκων στα νοσοκομεία, την

τηλεπαρακολούθηση ζωτικών σημείων, τον εντοπισμό και την παρακολούθηση γιατρών και ασθενών μέσα στο νοσοκομείο και άλλα [15][19][20][21].

Εντοπισμός και παρακολούθηση γιατρών και ασθενών μέσα στο νοσοκομείο. Αν κάθε γιατρός έχει έναν αισθητήρα επάνω του, τότε είναι πολύ εύκολο να εντοπιστεί από τους άλλους γιατρούς και το νοσηλευτικό προσωπικό σε περίπτωση ανάγκης. Οι ασθενείς μπορούν να έχουν έναν ελαφρύ και μικρό κόμβο-αισθητήρα επάνω τους, ο οποίος μπορεί να παρακολουθεί κάποια συγκεκριμένα ζωτικά σημεία, όπως τους παλμούς της καρδιάς και την πίεση του αίματος, για τη συνεχή και έγκαιρη ενημέρωση των γιατρών.

Τηλεπαρακολούθηση των ζωτικών σημείων. Τα στοιχεία που συγκεντρώνουν οι κόμβοι μπορούν να αποθηκεύονται για μεγάλα χρονικά διαστήματα και στη συνέχεια να τα επεξεργάζονται οι γιατροί. Επιπλέον μπορούν να παρακολουθούνται και να ανιχνεύονται συμβάντα σε ηλικιωμένους ανθρώπους όπως χτύπημα ή πτώση [16]. Αυτό επιτρέπει μεγαλύτερη ελευθερία κινήσεων στους ανθρώπους που παρακολουθούνται και προσφέρει έγκαιρη διάγνωση πιθανών νοσημάτων μέσω της ανίχνευσης συγκεκριμένων συμπτωμάτων. (Εικ.2)



Εικ.2 Χρήση ΑΔΑ και κινητής τηλεφωνίας για την τηλεπαρακολούθηση ασθενών [16]

2.5 Άλλες εφαρμογές

Οικιακοί αυτοματισμοί (smart home). Για τέτοιες εφαρμογές θα πρέπει να τοποθετηθούν κόμβοι αισθητήρων και «ενεργοποιητές» (actuators) σε διάφορες οικιακές συσκευές, όπως στην ηλεκτρική κουζίνα, στο φούρνο μικροκυμάτων κ.α. Οι κόμβοι μπορούν να επικοινωνούν μεταξύ τους αλλά και με το εξωτερικό δίκτυο μέσω του internet ή και μέσω δορυφόρου. Έτσι οι χρήστες μπορούν να διαχειρίζονται εύκολα τις οικιακές συσκευές είτε τοπικά, είτε απομακρυσμένα. [27]

Έξυπνα κτήρια (smart buildings). Τα κτήρια ξοδεύουν μεγάλα ποσά ενέργειας λόγω μη-βέλτιστης ρύθμισης του συστήματος θέρμανσης, ψύξης και ανακύκλωσης αέρα. Με τη βοήθεια των ΑΔΑ μπορούμε σε πραγματικό χρόνο να παρατηρούμε και να ρυθμίζουμε τη θερμοκρασία, την υγρασία και τη ροη του αέρα έτσι ώστε να επιτύχουμε μείωση της κατανάλωσης ενέργειας.

Παρακολούθηση μηχανημάτων και προληπτική συντήρηση. Η βασική ιδέα σε αυτήν την περίπτωση είναι η τοποθέτηση κόμβων σε απομακρυσμένα μηχανήματα ή μηχανικά μέρη που δεν είναι εύκολα προσβάσιμα. Αυτοί οι κόμβοι θα ανιχνεύουν συγκεκριμένα μοτίβα δονήσεων που υποδηλώνουν την ανάγκη συντήρησης.

Παρακολούθηση αποθήκης. Εφοδιάζοντας κάθε προϊόν μιας αποθήκης με κατάλληλο κόμβο-αισθητήρα, μπορούμε να έχουμε άμεση ενημέρωση για το απόθεμα του συγκεκριμένου προϊόντος αλλά και της θέσης του στην αποθήκης. Με αυτόν τον τρόπο γίνεται ευκολότερη η οργάνωση και η εποπτεία της αποθήκης.

Κεφάλαιο 3ο

Αρχιτεκτονική των Ασύρματων Δικτύων Αισθητήρων

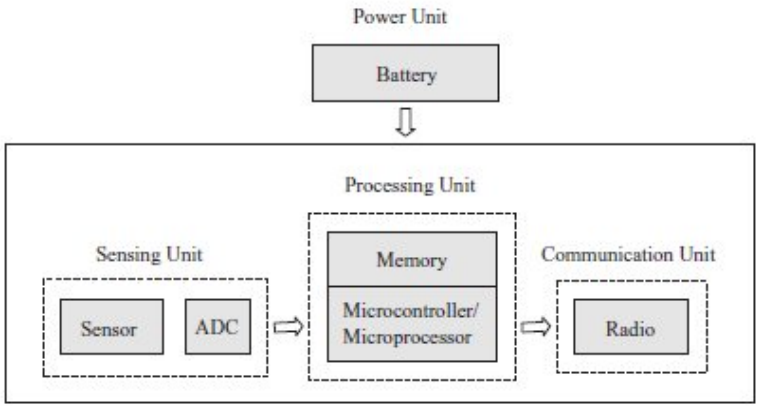
3.1 Εισαγωγή

Η αρχιτεκτονική του δικτύου και τα πρωτόκολλα δρομολόγησης που θα χρησιμοποιηθούν είναι πολύ σημαντικά στοιχεία στο σχεδιασμό των ΑΔΑ. Εξ αιτίας των μεγάλων περιορισμών όσο αφορά τη διαθέσιμη ενέργεια των κόμβων, η αρχιτεκτονική του δικτύου έχει μεγάλη σημασία για την κατανάλωση ενέργειας, και συνεπώς για το χρόνο αξιοποίησης ενός τέτοιου δικτύου. Έτσι, χρειάζεται μια στοίβα πρωτοκόλλων δικτύου για την υλοποίηση του έλεγχου και της διαχείρισης του δικτύου, όπως π.χ. συγχρονισμό, αυτο-οργάνωση, έλεγχο πρόσβασης στο μέσο (Medium Access Control), δρομολόγηση, συνάθροιση των δεδομένων (data aggregation), σύστημα εντοπισμού των κόμβων (node localization) καθώς και την ασφάλεια του δικτύου.

Ωστόσο, τα υπάρχοντα πρωτόκολλα δικτύου για τα παραδοσιακά ασύρματα δίκτυα, όπως κυψελωτά δίκτυα (cellular) και MANETs (Mobile Adhoc NETworks), δεν μπορούν να εφαρμοστούν επακριβώς στα δίκτυα αισθητήρων, διότι δεν λαμβάνουν υπ' όψιν τους ειδικούς περιορισμούς των ΑΔΑ σε ενέργεια, αποθηκευτικό χώρο και υπολογιστική ισχύ. Εκτός αυτού, τα περισσότερα ΑΔΑ αναπτύσσονται με βάση τη συγκεκριμένη εφαρμογή και ως εκ τούτου έχουν διαφορετικές κάθε φορά απαιτήσεις.

Για όλους τους παραπάνω λόγους χρειάζεται μια καινούρια σουίτα πρωτοκόλλων δικτύου που θα λαμβάνει υπ' όψιν όχι μόνο τους περιορισμούς των ΑΔΑ όσο αφορά τους πόρους των κόμβων, αλλά και τις απαιτήσεις των συγκεκριμένων εφαρμογών.

3.2 Αρχιτεκτονική του κόμβου-αισθητήρα



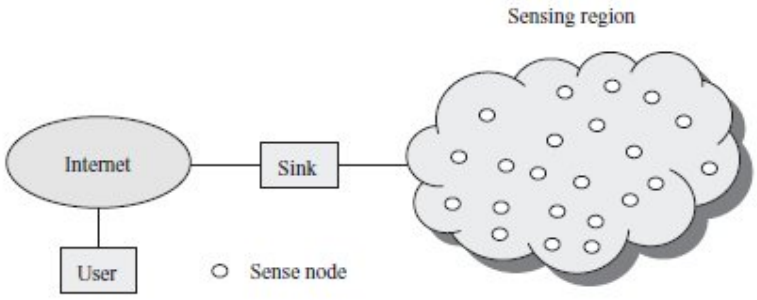
Εικ.3 Δομή ενός κόμβου-αισθητήρα [28]

Ένας κόμβος-αισθητήρας αποτελείται τυπικά από 4 βασικά στοιχεία: μια μονάδα «αίσθησης», μια μονάδα επεξεργασίας, μια μονάδα υπεύθυνη για την επικοινωνία και μια μονάδα για την παροχή ισχύος-ενέργειας. Η μονάδα «αίσθησης» (sensing) αποτελείται από έναν ή περισσότερους αισθητήρες και αντίστοιχους μετατροπείς από αναλογικό σε ψηφιακό σήμα. Οι αισθητήρες κάνουν μετρήσεις διαφόρων φυσικών μεγεθών ενός φαινομένου και παράγουν αναλογικά σήματα. Τμήμα της μονάδας «αίσθησης» (sensing unit) είναι και ο μετατροπέας από αναλογικό σε ψηφιακό σήμα. Τα κυκλώματα αυτά (Analog to Digital Converters) μετατρέπουν το αναλογικό σήμα σε ψηφιακό και στέλνουν το ψηφιακό σήμα στη μονάδα επεξεργασίας. Η μονάδα επεξεργασίας (Processing Unit) συνήθως αποτελείται από έναν μικροελεγκτή ή από έναν μικροεπεξεργαστή με μνήμη (π.χ. StrongARM microprocessor της Intel ή AVR microprocessor της Atmel), ο οποίος προσδίδει μια «ευφυΐα» (intelligent control) στον αισθητήρα. Η μονάδα ασύρματης επικοινωνίας (Communication Unit) αποτελείται από ένα κύκλωμα ασύρματης μετάδοσης-εκπομπής μικρής εμβέλειας (short range radio), και είναι υπεύθυνη για τη μετάδοση και λήψη δεδομένων στο ασύρματο κανάλι. Η μονάδα ενέργειας (Power Unit) αποτελείται από μια μπαταρία για να δίνει ενέργεια σε όλες τις υπόλοιπες μονάδες του κόμβου-αισθητήρα. Επιπροσθέτως ένας κόμβος-αισθητήρας μπορεί να περιλαμβάνει και άλλες μονάδες που εξαρτώνται από κάποιες συγκεκριμένες εφαρμογές. Π.χ. μπορεί να περιλαμβάνει μονάδα GPS για την περίπτωση που κάποια εφαρμογή χρειάζεται πληροφορίες για τη γεωγραφική

θέση του αισθητήρα, έτσι ώστε να λειτουργήσει το δίκτυο. Επίσης μπορεί να υπάρχει ένας μικρός κινητήρας (μοτέρ) σε περίπτωση που έχουμε αισθητήρες με δυνατότητα κίνησης.

3.3 Αρχιτεκτονική του δικτύου

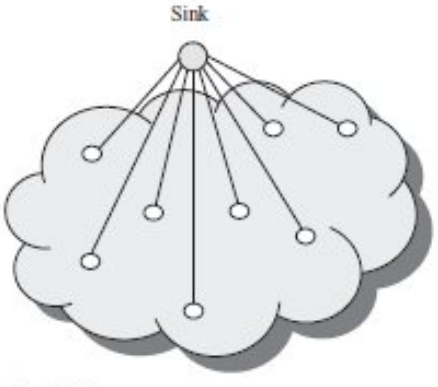
Ένα δίκτυο αισθητήρων αποτελείται από ένα μεγάλο αριθμό κόμβων-αισθητήρων που είναι διεσπαρμένοι πυκνά σε μια περιοχή που μας ενδιαφέρει προς παρακολούθηση, και υπάρχουν ένας ή περισσότεροι Σταθμοί Βάσης οι οποίοι είναι τοποθετημένοι κοντά ή και μέσα στην υπό εποπτεία περιοχή όπως φαίνεται στην εικ.4



Εικ.4 Αρχιτεκτονική Ασύρματου Δικτύου Αισθητήρων [28]

Ο Σταθμός Βάσης στέλνει «ερωτήματα» (queries) ή εντολές στους κόμβους-αισθητήρες και οι κόμβοι συνεργάζονται μεταξύ τους για να κάνουν τις μετρήσεις και να στείλουν τα δεδομένα πίσω στο Σταθμό Βάσης. Εν τω μεταξύ, ο Σταθμός Βάσης λειτουργεί και ως πύλη (gateway), προς εξωτερικά δίκτυα όπως το internet. Συλλέγει δεδομένα από τους αισθητήρες, τα επεξεργάζεται και τα στέλνει στους χρήστες που τα έχουν ζητήσει.

Για να στείλει δεδομένα στο Σταθμό Βάσης ένας κόμβος μπορεί να χρησιμοποιήσει μετάδοση «ενός βήματος» (single-hop) μεγάλης εμβέλειας. Στην περίπτωση αυτή έχουμε αρχιτεκτονική απευθείας μετάδοσης (single-hop) όπως φαίνεται στην εικ.5.



Εικ.5 ΑΔΑ ενός-άλματος (single-hop) [28]

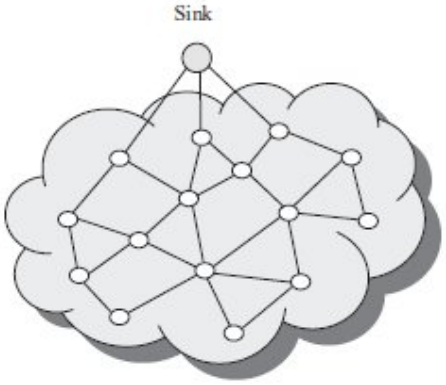
Ωστόσο, τέτοιου είδους επικοινωνία είναι δαπανηρή όσο αφορά την κατανάλωση ενέργειας. Όπως αναφέραμε και στην εισαγωγή, στα δίκτυα αισθητήρων η κατανάλωση ενέργειας για την ασύρματη επικοινωνία είναι πολύ μεγαλύτερη σε σχέση με την ενέργεια που καταναλώνεται για τη πραγματοποίηση μετρήσεων των φυσικών μεγεθών (sensing) και για τους υπολογισμούς. Για παράδειγμα, η ενέργεια που απαιτείται για τη μετάδοση 1 bit δεδομένων σε απόσταση 100m, είναι όση απαιτείται για την εκτέλεση 3000 εντολών στον επεξεργαστή [2]. Ο λόγος (κλάσμα) της ενέργειας για τη μετάδοση 1 bit πληροφορίας σε σχέση με την ενέργεια για επεξεργασία αυτής της πληροφορίας είναι από 1000 μέχρι και 10,000 [3,4] Έτσι, η κυριότερη κατανάλωση ενέργειας σε ένα ΑΔΑ (και σε ένα ασύρματο δίκτυο γενικότερα) είναι η κατανάλωση για ασύρματη επικοινωνία, διότι η ενέργεια αυτή αυξάνεται εκθετικά σε σχέση με την απόσταση. Έτσι είναι επιθυμητό να περιορίσουμε τις μεταδόσεις αλλά και την απόσταση μετάδοσης για να αυξήσουμε το χρόνο αξιοποίησης ενός ΑΔΑ.

Για αυτόν το λόγο προτιμούμε μεταδόσεις πολλαπλών βημάτων (multihop) και σε μικρή απόσταση. Στα ΑΔΑ, όπου, κατά κανόνα, οι κόμβοι είναι πυκνά τοποθετημένοι και κάθε κόμβος έχει κοντά του αρκετούς άλλους, είναι εφικτό να χρησιμοποιήσουμε τέτοιου είδους επικοινωνία (σε μικρή απόσταση με πολλαπλά άλματα-hops). Στη multihop επικοινωνία ένας κόμβος-αισθητήρας μεταδίδει τα δεδομένα του στο Σταθμό Βάσης μέσω ενός ή περισσότερων άλλων ενδιάμεσων κόμβων. Η αρχιτεκτονική ενός multihop δικτύου μπορεί να είναι 2 ειδών: επίπεδη (flat) και ιεραρχική (hierarchical) [5], περιπτώσεις που παρουσιάζονται στις αμέσως επόμενες ενότητες.

3.3.1 Επίπεδο δίκτυο (flat architecture)

Σε ένα επίπεδο δίκτυο, κάθε κόμβος παίζει τον ίδιο ρόλο στη λειτουργικότητα του δικτύου και όλοι οι κόμβοι είναι ισότιμοι. Εξ αιτίας του μεγάλου αριθμού των κόμβων στα ΑΔΑ, δεν είναι εφικτό να αποδώσουμε σε κάθε κόμβο ένα καθολικό-γενικό αναγνωριστικό (global ID). Έτσι, η συλλογή των δεδομένων γίνεται με χρήση πρωτοκόλλων δρομολόγησης προσανατολισμένα στα δεδομένα (data-centric), όπου ο Σταθμός Βάσης μεταδίδει ένα «ερώτημα» (query) προς όλους τους κόμβους στην περιοχή που γίνεται παρακολούθηση, μέσω «πλημμύρας» (flooding), αλλά στην επερώτηση αυτή απαντούν μόνο οι κόμβοι οι οποίοι έχουν δεδομένα που αφορούν (match) με τη συγκεκριμένη επερώτηση. Κάθε κόμβος επικοινωνεί με το Σταθμό Βάσης με μονοπάτια πολλαπλών

βημάτων (multihop path), χρησιμοποιώντας τους γείτονες του ως ενδιάμεσους κόμβους (relays). Στην εικ.6 φαίνεται η δομή ενός τέτοιου δικτύου.



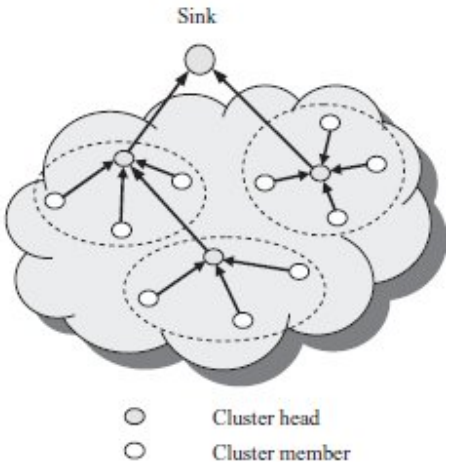
Εικ.6: Αρχιτεκτονική «επίπεδου» δικτύου [28]

3.3.2 Ιεραρχική αρχιτεκτονική (Hierarchical Architecture)

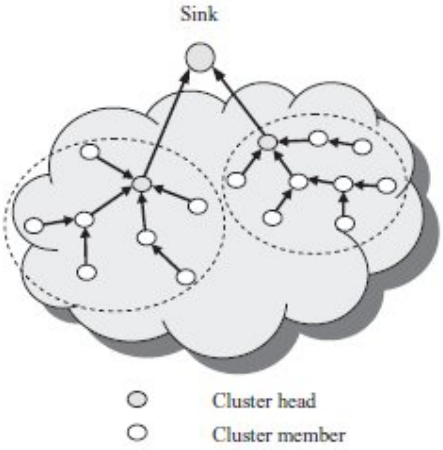
Σε ένα ιεραρχικά δομημένο δίκτυο, οι κόμβοι-αισθητήρες οργανώνονται σε συστάδες (clusters). Κάθε συστάδα έχει έναν επικεφαλής (cluster head). Οι κόμβοι στέλνουν τα δεδομένα τους στον επικεφαλής, οι οποίοι επικεφαλείς λειτουργούν ως αναμεταδότες (relays) για να μεταδοθούν τα δεδομένα στο Σταθμό Βάσης. Έτσι, ένας κόμβος με χαμηλή ενέργεια μπορεί να στείλει τα δεδομένα του στον επικεφαλής που είναι αρκετά κοντά, και ένας άλλος κόμβος με περισσότερη περίσσεια ενέργειας μπορεί να γίνει επικεφαλής και να μεταδώσει τα δεδομένα από τη συστάδα του στο Σταθμό Βάσης. Αυτή η διαδικασία όχι μόνο μειώνει την κατανάλωση της ενέργειας στο δίκτυο, αλλά επιπλέον εξισορροπεί το φορτίο επικοινωνίας και κάνει το δίκτυο να είναι επεκτάσιμο (scalable) σε περίπτωση που θέλουμε (π.χ.) να προσθέσουμε επιπλέον κόμβους. Επειδή όλοι οι κόμβοι έχουν την ίδια δυνατότητα μετάδοσης, πρέπει ανά τακτά χρονικά διαστήματα να γίνεται ανα-συσταδοποίηση έτσι ώστε να έχουμε εξισορρόπηση του φορτίου μεταξύ των κόμβων. Επιπλέον στους επικεφαλής μπορούμε να κάνουμε και συνάθροιση δεδομένων (data aggregation) έτσι ώστε να περιοριστεί ακόμη περισσότερο ο όγκος των προς μετάδοση δεδομένων στο Σταθμό Βάσης και έτσι να βελτιωθεί ακόμη περισσότερο η αποδοτικότητα του δικτύου.[6]

Το κυρίως πρόβλημα με τη συσταδοποίηση (clustering) είναι το πώς να επιλέξουμε τον επικεφαλής και πώς να οργανώσουμε τις συστάδες. Για το πρόβλημα αυτό υπάρχουν

αρκετές στρατηγικές συσταδοποίησης. Δυο διαφορετικές προσεγγίσεις στο συγκεκριμένο πρόβλημα φαίνονται στις εικ.7 και εικ.8. Στην πρώτη περίπτωση, σε κάθε συστάδα η επικοινωνία μεταξύ των κόμβων με τον επικεφαλής είναι ενός-βήματος (single hop). Στην περίπτωση αυτή, βέβαια, οι επικεφαλής των συστάδων μπορεί να επικοινωνούν είτε απευθείας με το Σταθμό Βάσης (single hop) είτε μέσω άλλων επικεφαλής συστάδων (multihop).



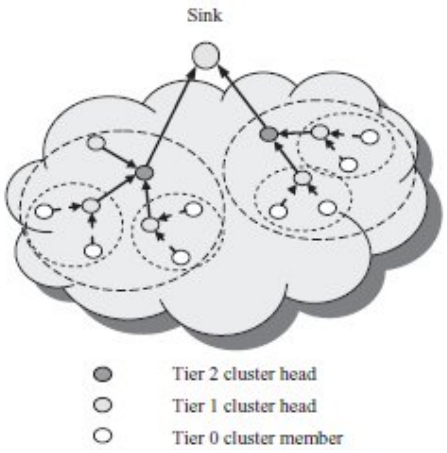
Εικ.7:Αρχιτεκτονική single-hop clustering



Εικ.8: Αρχιτεκτονική multi-hop clustering

Στη δεύτερη περίπτωση (εικ.8), ο κάθε επικεφαλής της συστάδας, είναι στην πραγματικότητα η ρίζα ενός δένδρου, οι ακμές του οποίου δένδρου είναι τα μονοπάτια δρομολόγησης (routing paths) των δεδομένων από τους αισθητήρες που ανήκουν στη συγκεκριμένη συστάδα στον επικεφαλής. Στη συνέχεια οι επικεφαλής, είτε επικοινωνούν άμεσα (single hop) με το Σταθμό Βάσης, είτε με πολλαπλά βήματα (multihop) μέσω ενδιάμεσων κόμβων (επικεφαλής άλλων συστάδων).

Επιπλέον μπορούμε να δομήσουμε το δίκτυο και σε επίπεδα (tiers). Έτσι μπορούμε να έχουμε αρχιτεκτονικές συσταδοποίησης ενός (single-tier) ή πολλαπλών επιπέδων (multi-tier) . Στην εικ.9 έχουμε ένα δίκτυο οργανωμένο σε συστάδες πολλαπλών επιπέδων (multi-tier clustering). Για την αντιμετώπιση του προβλήματος της συσταδοποίησης έχουν προταθεί διάφοροι αλγόριθμοι [7][13].



Εικ.9: Αρχιτεκτονική multi-tier clustering [28]

Κεφάλαιο 4ο

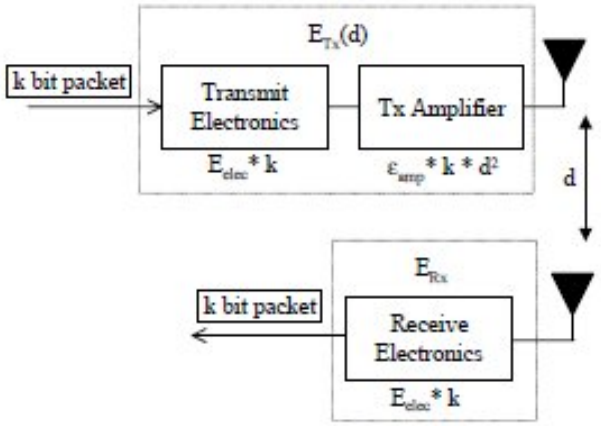
Πρωτόκολλα Δρομολόγησης στα Ασύρματα Δίκτυα Αισθητήρων

4.1 Το Μοντέλο Ασύρματης Επικοινωνίας στα ΑΔΑ (First Order Radio Model)

Πριν παρουσιάσουμε τα διάφορα πρωτόκολλα δρομολόγησης στα ΑΔΑ, θα αναλύσουμε το Μοντέλο Ασύρματης Επικοινωνίας (First Order Radio Model), που χρησιμοποιείται από τους ερευνητές για τη μελέτη των ΑΔΑ.

Στις ήμερες μας, γίνεται αρκετή έρευνα στην περιοχή της ασύρματης μετάδοσης έτσι ώστε να επιτύχουμε όσο το δυνατόν μικρότερη κατανάλωση ενέργειας για τη λειτουργία του ασύρματου καναλιού. Για τη μελέτη της κατανάλωσης ενέργειας στα Ασύρματα Δίκτυα Αισθητήρων, θα χρησιμοποιήσουμε το Μοντέλο Ασύρματης Επικοινωνίας (First Order Radio Model) που παρουσιάζεται στο LEACH από την Heinzelman [8]. Όλοι οι υπόλοιποι ερευνητές και των άλλων πρωτοκόλλων δρομολόγησης στα ΑΔΑ έχουν χρησιμοποιήσει αυτό το μοντέλο ασύρματης επικοινωνίας.

Στην Εικ.10 φαίνεται ένα σχεδιάγραμμα με δυο κόμβους αισθητήρες, όπου ο ένας είναι ο πομπός και ο άλλος ο δέκτης, οι οποίοι απέχουν απόσταση d μέτρα. Στο διάγραμμα σημειώνονται τα ποσά ενέργειας που καταναλώνονται σε κάθε μονάδα των αισθητήρων.



Εικ.10: Μοντέλο Ασύρματης Επικοινωνίας για τα ΑΔΑ (First Order Radio Model)

Συγκεκριμένα, στο επάνω τμήμα της εικόνας είναι ο κόμβος-πομπός και στο κάτω τμήμα απεικονίζεται ο κόμβος-δέκτης. Κάθε αισθητήρας έχει ηλεκτρονικά κυκλώματα που ενεργοποιούνται τόσο στη φάση της μετάδοσης, όσο και στη φάση της λήψης δεδομένων (transceiver), Επιπλέον, υπάρχει και ο Ενισχυτής Μετάδοσης (Transmit Amplifier) ο οποίος ενεργοποιείται μόνο στην περίπτωση που ο αισθητήρας πρέπει να μεταδώσει δεδομένα. Υποθέτουμε ότι η απόσταση μεταξύ των δυο κόμβων είναι d μέτρα. Στη συνέχεια θα δώσουμε τις τυπικές τιμές στα μεγέθη που προαναφέραμε.

Λειτουργία	Κατανάλωση Ενέργειας
Transmitter Electronics ($E_{Tx\text{-}elec}$) Receiver Electronics ($E_{Rx\text{-}elec}$) ($E_{Tx\text{-}elec} = E_{Rx\text{-}elec} = E_{elec}$)	50nJ/bit
Transmit Amplifier (Eamp)	100 pJ/b/m^2

Πινακας1.

Σύμφωνα με τον πινακα1, για να μεταδώσει ένας κόμβος ένα πακέτο δεδομένων k bits σε d μέτρα, πρέπει να καταναλώσει:

$E_{Tx}(k,d) = E_{Tx\text{-}elec}(k) + E_{Tx\text{-}amp}(k.d)$, που γίνεται:

$$\mathbf{E_{Tx}(k,d) = E_{elec} * k + E_{amp} * k * d^2} \quad (1)$$

Και για να γίνει λήψη του μηνύματος αυτού, ο δέκτης θα καταναλώσει ενέργεια:

$E_{Rx}(k) = E_{Rx\text{-}elec}(k)$

$$E_{Rx}(k) = E_{elec} * k \quad (2)$$

Από τις εξισώσεις (1),(2) βγαίνει το συμπέρασμα ότι, σε ένα ΑΔΑ έχουμε κατανάλωση ενέργειας, όχι μόνο στην περίπτωση μετάδοσης δεδομένων από τον πομπό, αλλά και από τον δέκτη, κατά τη λήψη του σήματος. Αυτό το χαρακτηριστικό είναι ένα στοιχείο το οποίο έχει ληφθεί σοβαρά υπ' όψιν στο σχεδιασμό των πρωτοκόλλων δρομολόγησης στα ΑΔΑ, όπως θα δούμε και στις επόμενες παραγράφους.

4.2 Δρομολόγηση στα ΑΔΑ – Γενικές παρατηρήσεις

Στην ενότητα αυτή θα εξετάσουμε τα πρωτόκολλα δρομολόγησης που χρησιμοποιούνται στα ΑΔΑ. Γενικά η δρομολόγηση στα ΑΔΑ, μπορεί να χωριστεί σε τρεις κατηγορίες σε σχέση με την τοπολογία του δικτύου: δρομολόγηση σε επίπεδα δίκτυα, δρομολόγηση σε ιεραρχικά δομημένα δίκτυα και δρομολόγηση βασισμένη στη γεωγραφική θέση των κόμβων. Στα επίπεδα δίκτυα όλοι οι κόμβοι έχουν ιδίους ρόλους και λειτουργικότητα. Στα δίκτυα τα οποία είναι δομημένα ιεραρχικά, υπάρχουν κάποιοι κόμβοι οι οποίοι μπορεί να επιτελούν διαφορετικές λειτουργίες . Τέλος, στα πρωτόκολλα δρομολόγησης που βασίζονται στη θέση των κόμβων για τη δρομολόγηση των δεδομένων μέσα στο δίκτυο χρησιμοποιούνται οι γεωγραφικές θέσεις των κόμβων.

Επιπλέον, τα πρωτόκολλα μπορούν να ταξινομηθούν ανάλογα με τη λειτουργία του πρωτοκόλλου σε: πολλαπλών διαδρομών (multipath-based), βασισμένα στις επερωτήσεις (query-based), βασισμένα σε «διαπραγμάτευση» (negotiation based), βασισμένα στην ποιότητα της υπηρεσίας (Quality of Service based) κ.α.

Επιπροσθέτως τα πρωτόκολλα δρομολόγησης μπορούν να ταξινομηθούν σε τρεις επιπλέον κατηγορίες: proactive, reactive και υβριδικά (hybrid) όσο αφορά το με ποιον τρόπο υπολογίζεται το μονοπάτι δρομολόγησης (routing path) από την πηγή (source) προς τον προορισμό (destination). Proactive: το μονοπάτι υπολογίζεται πριν γίνει η δρομολόγηση (χρησιμοποιούνται πίνακες δρομολόγησης), στα reactive, το μονοπάτι υπολογίζεται κατ' απαίτηση (on demand), δηλ. όταν δημιουργηθούν τα δεδομένα στην πηγή (source) τότε το δίκτυο «βρίσκει» το μονοπάτι μέσω του οποίου θα δρομολογηθούν τα δεδομένα στον προορισμό τους. Στα υβριδικά πρωτόκολλα, γίνεται ένας συνδυασμός των δυο άλλων τεχνικών (proactive-reactive). [28]

Στην περίπτωση που το δίκτυο αποτελείται από στατικούς κόμβους (μη μετακινουμένους) είναι προτιμότερο να χρησιμοποιήσουμε proactive πρωτόκολλα δηλ. πρωτόκολλα που χρησιμοποιούν πίνακες δρομολόγησης, αντί για reactive. Διότι στα reactive πρωτόκολλα ξοδεύεται αρκετή ενέργεια τόσο στην οργάνωση του μονοπατιού (route setup and discovery). (from Al-Karaki)

Από τις διάφορες κατηγορίες πρωτοκόλλων που μόλις παρουσιάσαμε, για τις ανάγκες της συγκεκριμένης εργασίας θα αναλύσουμε διεξοδικότερα τα πρωτόκολλα που σχετίζονται με την τοπολογία του δικτύου και συγκεκριμένα τα πρωτόκολλα δρομολόγησης σε δίκτυα που έχουν ιεραρχική δομή (hierarchical networks). Στα ιεραρχικά δομημένα δίκτυα διακρίνουμε δυο υποκατηγορίες: βασισμένα σε συστάδες (cluster based) και βασισμένα σε δενδρική δομή (tree based).

4.3 Διαφορές μεταξύ ΑΔΑ και ασύρματων ad hoc δικτύων.

Τα ΑΔΑ αναπτύχθηκαν σαν μια ειδική μορφή των AdHoc δικτύων. Όμως, παρουσιάζουν αρκετές διαφορές από τα AdHoc, γι' αυτό και δεν μπορούμε να εφαρμόσουμε ακριβώς τις ίδιες τεχνικές δρομολόγησης και στις δυο κατηγορίες δικτύων [11].

Τις διαφορές αυτές θα μπορούσαμε να τις συνοψίσουμε στις πιο κάτω:

- Ο αριθμός των κόμβων σε ένα ΑΔΑ είναι πολύ μεγαλύτερος από αυτόν ενός AdHoc δικτύου. Γενικά στα ΑΔΑ μπορούμε να έχουμε από δεκάδες μέχρι και χιλιάδες κόμβους-αισθητήρες.
- Οι κόμβοι στα ΑΔΑ είναι διεσπαρμένοι περισσότερο πυκνά απ' ότι στα AdHoc.
- Οι κόμβοι στα ΑΔΑ είναι περισσότερο «επιρρεπείς» στα λάθη (failure prone)
- Η τοπολογία ενός ΑΔΑ αλλάζει αρκετά συχνά, είτε όταν «αφαιρούνται» κόμβοι διότι έχει τελειώσει η μπαταρία τους, είτε όταν προσθέτουμε καινούριους κόμβους για να επεκτείνουμε το δίκτυο.

- Στα ΑΔΑ χρησιμοποιείται κυρίως η μετάδοση «εκπομπής» (broadcasting), σε αντίθεση με τα AdHoc όπου χρησιμοποιείται κυρίως η μετάδοση σημείο-προς-σημείο (point-to-point).
- Οι κόμβοι στα ΑΔΑ έχουν περιορισμούς στην ενέργεια (μπαταρία), στις υπολογιστικές δυνατότητες και στη μνήμη, άρα και στην επικοινωνία.
- Οι κόμβοι στα ΑΔΑ δεν είναι δυνατόν να έχουν ένα global ID (identification), εξ αιτίας του μεγάλου overhead και του μεγάλου αριθμού των κόμβων. (σημ.: όταν λέμε global ID, εννοούμε κάθε κόμβος να πάρει μια IP address. Αυτό δεν είναι δυνατόν στα ΑΔΑ διότι έχουμε πολύ μεγάλο αριθμό κόμβων και έτσι θα είχαμε πολύ μεγάλο overhead στην επικοινωνία στο δίκτυο, ιδίως κατά τη φάση της αρχικής «οργάνωσης» του δικτύου, configuration or setup phase).

4.4 Ιεραρχικά δομημένα δίκτυα

Στις αρχές της δεκαετίας το 2000, είχαν προταθεί διάφορα πρωτόκολλα βασισμένα σε δίκτυα «επίπεδης» αρχιτεκτονικής, όπως το Directed Diffusion. Μια τέτοια αρχιτεκτονική, όμως, μπορεί να προκαλέσει επιπλέον επιβάρυνση στο Σταθμό Βάσης, και έτσι να αδειάσει γρήγορα η μπαταρία του ΣΒ. Ο «θάνατος» του Σταθμού Βάσης προκαλεί και την κατάρρευση της λειτουργικότητας όλου του δικτύου. Έπρεπε, λοιπόν, να αναπτυχθούν καινούρια πρωτόκολλα στα οποία να μην έχει τόσο μεγάλη ενεργειακή επιβάρυνση ο Σταθμός Βάσης. Αυτή η κατηγορία πρωτοκόλλων ήταν τα ιεραρχικά πρωτόκολλα.

Όπως έχουμε αναφέρει και σε προηγούμενη παράγραφο για τα ιεραρχικά δομημένα δίκτυα, μπορούμε να οργανώσουμε ένα ΑΔΑ σε συστάδες (clusters). Κάθε συστάδα έχει έναν επικεφαλής (cluster head). Ο επικεφαλής της κάθε συστάδας είναι ένας κόμβος «ειδικού» σκοπού, που λειτουργεί ως «αναμεταδότης» (relay) για τη δρομολόγηση των δεδομένων από τους κόμβους-αισθητήρες προς το Σταθμό Βάσης. Αλλά και στην περίπτωση που θα κάνουμε «συνάθροιση» δεδομένων (data aggregation), αυτό γίνεται στους επικεφαλής των συστάδων και έτσι έχουμε μείωση των μηνυμάτων δεδομένων που αποστέλλονται στο Σταθμό Βάσης. Αυτό βελτιώνει την ενεργειακή απόδοση του δικτύου.(Rajagopalan)

Στη συνέχεια θα παρουσιάσουμε τα κυριότερα ιεραρχικά πρωτόκολλα δρομολόγησης και θα επισημάνουμε τα προτερήματα αλλά και τους περιορισμούς τους.

4.5 Δρομολόγηση σε δίκτυα συστάδων (cluster based).

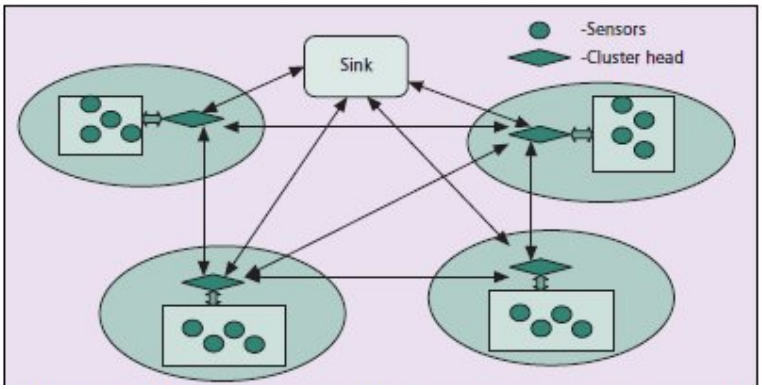
Σε δίκτυα τα οποία έχουν ενεργειακούς περιορισμούς, όπως τα ΑΔΑ, δεν είναι αποτελεσματικό (όπως έχουμε αναφέρει και πιο πάνω) κάθε κόμβος να μεταδίδει απευθείας στο Σταθμό Βάσης. Σε αυτές τις περιπτώσεις τα ΑΔΑ μπορούν να οργανωθούν σε συστάδες (clusters). Κάθε συστάδα έχει έναν κεντρικό κόμβο, τον επικεφαλής (cluster head). Οι κόμβοι-αισθητήρες μπορούν να μεταδίδουν τα δεδομένα τους στον επικεφαλής της συστάδας, ο οποίος επιπλέον έχει τη δυνατότητα να «συναθροίζει» (aggregate) τα δεδομένα από τους κόμβους που ανήκουν στη συγκεκριμένη συστάδα (cluster), και να μεταδίδει το επεξεργασμένο αποτέλεσμα στο Σταθμό Βάσης. (Figure2) Αυτή η πρακτική έχει σαν αποτέλεσμα να γίνεται μεγάλη οικονομία στην ενέργεια που καταναλώνεται τόσο σε κάθε κόμβο αλλά και στο δίκτυο γενικότερα.

Όπως έχουμε αναφέρει και στην παράγραφο για την αρχιτεκτονική των ΑΔΑ, οι επικεφαλής των συστάδων (cluster heads), μπορούν να επικοινωνούν με το Σταθμό Βάσης είτε μέσω μεταδόσεων μεγάλης εμβέλειας (long range transmissions) η με μεταδόσεις πολλαπλών βημάτων (multihop), μέσω άλλων επικεφαλής από άλλες συστάδες. Έχουν προταθεί αρκετά πρωτόκολλα που βασίζονται στη συσταδοποίηση (cluster based). Εδώ θα παρουσιάσουμε δυο από αυτά: το LEACH (Low-Energy Adaptive Clustering Hierarchy) και το HEED (Hybrid Energy-Efficient Distributed clustering approach).

4.5.1 LEACH

Η Heinzelman [8], ήταν η πρώτη που παρουσίασε το 2000 ένα πρωτόκολλο δρομολόγησης που να είναι cluster based. Το πρωτόκολλο είναι κατανεμημένο (distributed) και οι κόμβοι-αισθητήρες αυτο-οργανώνονται σε συστάδες για να στείλουν τα δεδομένα τους στο Σταθμό Βάσης. Ένας ορισμένος κόμβος γίνεται επικεφαλής (cluster head) στην κάθε συστάδα και μεταδίδει τα δεδομένα στο Σταθμό Βάσης. Στον επικεφαλής γίνεται και «συνάθροιση» των δεδομένων (data aggregation). Αυτό μειώνει τον όγκο της πληροφορίας που μεταδίδεται στο Σταθμός Βάσης. Η «συνάθροιση» πραγματοποιείται

περιοδικά στον επικεφαλής. Το LEACH είναι κατάλληλο για εφαρμογές όπου έχουμε συνεχή παρακολούθηση και αποστολή μετρήσεων σε τακτά χρονικά διαστήματα. (resident queries)



Εικ.10: Δίκτυο βασισμένο σε συστάδες (cluster based network)[26]

4.5.1.1 Τρόπος λειτουργίας του LEACH.

Στο LEACH υπάρχουν 2 φάσεις: η φάση «οργάνωσης» (setup phase) και η φάση «σταθερής κατάστασης» (steady state phase). Στη φάση της «οργάνωσης» γίνεται η δημιουργία των συστάδων (clusters) και η επιλογή των επικεφαλής (cluster heads). Στο steady-state phase, γίνεται η συνάθροιση των δεδομένων στους επικεφαλής (cluster head) και η αποστολή του τελικού αποτελέσματος στο Σταθμό Βάσης.

Η επιλογή των επικεφαλής των συστάδων γίνεται ως εξής: Αρχικά, από το σχεδιαστή του δικτύου έχει αποφασιστεί ένα «κλάσμα» (ποσοστό) f από τους κόμβους που θα γίνουν cluster heads. Κάθε κόμβος i συγκρίνει έναν τυχαίο αριθμό n που ανήκει στο διάστημα (0,1] με ένα «κατώφλι» (threshold) ηi. Αν ο αριθμός αυτός n είναι μεγαλύτερος του ηi, τότε ο κόμβος i γίνεται επικεφαλής. Το κατώφλι ηi δίνεται από τον τύπο:

$$\eta_i = \frac{f}{1 - f(n \bmod (1/f))}$$

όπου mod είναι το υπόλοιπο της ευκλείδειας διαίρεσης (modulus). Δηλ. πρακτικά, ένας κόμβος που έχει γίνει επικεφαλής μια φορά, έχει πιθανότητα να ξαναγίνει μετά από (τουλάχιστον) f «γύρους» «συλλογής» δεδομένων (data gathering rounds). Όλοι οι κόμβοι που έχουν επιλεγεί ως επικεφαλής στέλνουν ένα μήνυμα (hello message) σε όλους τους υπόλοιπους κόμβους του δικτύου «ανακοινώνοντας» τους ότι είναι οι καινούριοι επικεφαλής (cluster heads). Οι υπόλοιποι κόμβοι που δεν είναι επικεφαλής επιλέγουν σε ποια συστάδα (δηλ. σε ποιον επικεφαλής) θα «υπαχθούν», με βάση την ισχύ του σήματος

που έχουν λάβει από τους κόμβους που είναι ήδη επικεφαλής. Το LEACH αλλάζει τους επικεφαλής κυκλικά αλλά με τυχαίο τρόπο. Πετυχαίνει βελτίωση στην κατανάλωση ενέργειας σε σχέση με την απ' ευθείας μετάδοση (Direct Transmission-δηλ. ο κάθε κόμβος να μεταδίδει απ ευθείας στο σταθμό βάσης) κατά 8 φορές. [8]

Σε σύγκριση με το MTE (Minimum Transmission Energy) όπου ο επόμενος κόμβος στο μονοπάτι δρομολόγησης (routing path) είναι αυτός με την μικρότερη ευκλείδεια απόσταση (αθροίσματος των τετραγώνων των αποστάσεων μεταξύ δυο κόμβων) το LEACH μεταδίδει 10 φορές περισσότερα δεδομένα για τον ίδιο αριθμό κόμβων που τίθενται εκτός λειτουργίας (node deaths).

4.5.1.2 Περιορισμοί του LEACH

Το LEACH, αν και βελτιώνει το χρόνο αξιοποίησης του δικτύου (network lifetime), έχει και κάποιους περιορισμούς. Πρώτον, το πρωτόκολλο υποθέτει ότι όλοι οι κόμβοι έχουν αρκετή ενέργεια έτσι ώστε να μπορούν να επικοινωνήσουν απ ευθείας με το Σταθμό Βάσης αφού (δυνητικά) όλοι οι κόμβοι μπορούν να γίνουν επικεφαλής σε μια συστάδα. Αυτή η υπόθεση όμως μπορεί να μην ισχύει στην περίπτωση αισθητήρων με περιορισμό στην ενέργεια ή σε πολύ μεγάλα δίκτυα. Ας θυμηθούμε ότι (τυπικά) η ακτίνα εκπομπής ενός κόμβου αισθητήρα δεν ξεπερνά τα 30 με 50 μέτρα. Επίσης, το LEACH υποθέτει ότι οι κόμβοι έχουν δεδομένα να στείλουν στο Σταθμό Βάσης σε περιοδική βάση (resident queries). Άρα, το συγκεκριμένο πρωτόκολλο δεν είναι το καταλληλότερο για εφαρμογές όπως π.χ. η ανίχνευση κίνησης.

Όπως αναφέραμε, το LEACH είναι κατανεμημένος αλγόριθμος, δηλ. η δημιουργία των συστάδων (cluster setup) γίνεται «τοπικά» και όχι σε κεντρικό επίπεδο από το Σταθμό Βάσης. Στο [9] έχει προταθεί μια κεντρικοποιημένη παραλλαγή του πρωτοκόλλου, όπου η οργάνωση των συστάδων γίνεται κεντρικά από το Σταθμό Βάσης. Το πρωτόκολλο αυτό είναι το LEACH-C (Centralized LEACH) [9]. Το LEACH-C βελτιώνει την απόδοση του LEACH από 20% έως 40% σε όρους «γύρων» συλλογής δεδομένων (data gathering rounds).

4.5.2 HEED

Ο Younis έχει προτείνει το HEED (Hybrid Energy-Efficient Distributed clustering approach) [10], που ο σκοπός του είναι να δημιουργεί συστάδες με τέτοιον τρόπο, έτσι ώστε να μεγιστοποιείται ο χρόνος αξιοποίησης του δικτύου (network lifetime). Η βασική υπόθεση του HEED είναι το να υπάρχουν διαφορετικά επίπεδα ενέργειας στους κόμβους. Για να γίνει η επιλογή των επικεφαλής της κάθε συστάδας χρησιμοποιείται ένας συνδυασμός του επιπέδου της ενέργειας του κάθε κόμβου (υπόλοιπο μπαταρίας-residual energy), και ένας δευτερεύων παράγοντας που είναι είτε το πόσο κοντά είναι ο υποψήφιος επικεφαλής στον «απλό» κόμβο (minimum distance) είτε ο βαθμός του κόμβου (node degree), δηλ. του υποψήφιου να γίνει επικεφαλής. Όταν λέμε «βαθμός του κόμβου», εννοούμε πόσες εισερχόμενες και εξερχόμενες συνδέσεις (incoming+outcoming links) έχει αυτός ο κόμβος. Επειδή κάθε κόμβος έχει πάντα μια εξερχόμενη σύνδεση (one outcoming link), ο βαθμός του καθορίζεται από τις εισερχόμενες συνδέσεις. Έτσι εάν π.χ. δυο κόμβοι υποψήφιοι για επικεφαλής έχουν το ίδιο ποσό εναπομείνασας ενέργειας (residual energy) θα επιλεγεί σαν επικεφαλής εκείνος που έχει τις λιγότερες εισερχόμενες συνδέσεις, δηλ. έχει λιγότερους κόμβους συνδεδεμένους μαζί του. Στο πρωτόκολλο έχει εισαχθεί και ένας καινούριος όρος, το «κόστος επικοινωνίας» της κάθε συστάδας. Το κόστος επικοινωνίας της συστάδας ορίζεται ως ο μέσος όρος της ελάχιστης ενέργειας που απαιτείται από όλους τους κόμβους μέσα στη συστάδα για να επικοινωνήσουν με τον επικεφαλής, και στο [10] ονομάζεται AMRP (Average Minimum Reachability Power). Το AMRP είναι μια εκτίμηση για το κόστος επικοινωνίας των «απλών» κόμβων με τον επικεφαλής μέσα στην κάθε συστάδα..

4.5.2.1 Τρόπος λειτουργίας του HEED

Σε κάθε επανάληψη του HEED, κάθε κόμβος που δεν είναι επικεφαλής, θέτει την πιθανότητά του να εκλεγεί ως επικεφαλής, Pch ως εξής:

$$P_{CH} = C \times \frac{E_{residual}}{E_{\max}}$$

όπου C είναι το αρχικό ποσοστό των cluster heads που έχει ορίσει ο σχεδιαστής του δικτύου, (όπως το f στο πρωτόκολλο LEACH), Eresidual είναι το υπόλοιπο της μπαταρίας

του συγκεκριμένου κόμβου εκείνη τη στιγμή, ενώ το Emax είναι η μέγιστη ενέργεια του κόμβου, αυτή δηλ. που αντιστοιχεί σε μια πλήρως φορτισμένη μπαταρία.

Κάθε κόμβος κάνει broadcast ένα «ειδικό» μήνυμα, το cluster_head_msg. Επιπλέον υπάρχει και μια άλλη παράμετρος-«μεταβλητή» το selection status (κατάσταση «επιλεξιμότητας»). Αυτή η παράμετρος μπορεί να πάρει 2 τιμές: είτε δοκιμαστική (tentative ο συγκεκριμένος κόμβος ακόμη «δοκιμάζει» εάν θα είναι επικεφαλής) είτε τελική (final – ο συγκεκριμένος κόμβος είναι ήδη επικεφαλής). Εάν η πιθανότητα Pch είναι μικρότερη από 1, τότε το selection status είναι tentative (δοκιμαστικό), και εάν το Pch είναι 1, τότε το selection status είναι final, δηλ. ο κόμβος είναι ήδη επικεφαλής.

Ένας κόμβος επιλέγει σαν επικεφαλής τον κόμβο με το μικρότερο κόστος (AMRP) από το σύνολο των tentative (δοκιμαστικών) cluster heads. Τότε κάθε κόμβος κάνει την πιθανότητά του Pch σε min(2 X Pch, 1), στην επόμενη επανάληψη. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου κάθε κόμβος να «υπαχθεί» σε κάποιον επικεφαλής.

4.5.2.2 Σύγκριση του HEED με άλλα πρωτόκολλα

Έχει γίνει σύγκριση του HEED με το LEACH. Τα πρωτόκολλα προσομοιώθηκαν σε διάφορα μεγέθη δικτύων. Το simulation έδειξε ότι το HEED βελτιώνει το χρόνο ζωής του δικτύου σε σχέση με το LEACH. Στο LEACH η επιλογή των επικεφαλής είναι τυχαία, το οποίο μπορεί να οδηγήσει στο γρήγορο θάνατο κάποιων συγκεκριμένων κόμβων. Ενώ στο HEED η επιλογή των επικεφαλής γίνεται και με επιπρόσθετα κριτήρια (κόστος επικοινωνίας μέσα στη συστάδα-cluster). Επιπλέον το κόστος για την οργάνωση των συστάδων (clustering) είναι μικρότερο στο HEED απ’ ότι στο LEACH. Αυτό οφείλεται στο γεγονός ότι στο LEACH γίνεται κατά κάποιον τρόπο «διάδοση» (propagation) της υπολειπόμενης ενέργειας κάθε κόμβου στο δίκτυο. Συνοπτικά θα λέγαμε ότι το HEED αυξάνει το χρόνο ζωής του δικτύου και πετυχαίνει πιο ομοιόμορφη κατανομή των επικεφαλής των συστάδων (cluster heads).

4.6 Δρομολόγηση βασισμένη σε δενδρική δομή (Tree based routing)

Μια υποπερίπτωση της ιεραρχικής δόμησης ενός δικτύου είναι η δενδρική δομή. Στην περίπτωση αυτή, τα μονοπάτια δρομολόγησης (routing paths) από τον κάθε κόμβο-αισθητήρα (source) προς τον τελικό προορισμό (destination) ,που είναι ο Σταθμός Βάσης, είναι οι ακμές ενός δένδρου με κόμβους τους κόμβους-αισθητήρες του δικτύου. Μια περίπτωση όπου μπορεί να εφαρμοστεί η δενδρική «δόμηση» του δικτύου είναι π.χ. η μέτρηση της μέγιστης τιμής ακτινοβολίας σε έναν πυρηνικό αντιδραστήρα. Επειδή μας ενδιαφέρει μόνο η μέγιστη τιμή, μπορούμε να κάνουμε «συνάθροιση» των μετρήσεων στους ενδιάμεσους κόμβους και έτσι στο ΣΒ να φτάνει μόνο η μέγιστη τιμή, αυτή δηλ. που μας ενδιαφέρει. Ένα από τα κυριότερα προβλήματα σε ΑΔΑ με δενδρική δομή είναι η κατασκευή ενός «δένδρου δρομολόγησης» (data routing tree) τέτοιο ώστε να γίνεται η καλύτερη αξιοποίηση των ενεργειακών πόρων του δικτύου και συνεπώς η αύξηση του χρόνου ζωής του δικτύου.

Στη συνέχεια θα παρουσιάσουμε μερικά από τα πρωτόκολλα που χρησιμοποιούν δενδρική δομή για τη δρομολόγηση των δεδομένων από τους αισθητήρες στο Σταθμό Βάσης.

4.6.1 EADAT (efficient Energy-Aware Distributed heuristic to generate the Aggregation Tree, Ding – 2003)

Ο Ding [14] έχει προτείνει έναν «ευρετικό» (heuristic) αλγόριθμο για την κατασκευή και τη «συντήρηση» (maintenance) ενός τέτοιου δένδρου, το EADAT (efficient Energy-Aware Distributed heuristic to generate the Aggregation Tree).

4.6.1.1 Τρόπος λειτουργίας του EADAT

Αρχικά ο Σταθμός Βάσης μεταδίδει (κάνει broadcast) ένα «μήνυμα ελέγχου» (control message). Ο Σταθμός Βάσης θα είναι η ρίζα του δένδρου. Το control message έχει 5 πεδία: ID, parent, power, status, hopcount. ID: το «αναγνωριστικό» του κόμβου (sensor ID), parent: ο πατέρας του κόμβου, power: το υπόλοιπο της ενέργειας του κόμβου, status: η κατάσταση στην οποία βρίσκεται ο κόμβος. Υπάρχουν 3 καταστάσεις στις οποίες μπορεί να βρίσκεται ο κόμβος: φύλλο (leaf), όχι-φύλλο (non-leaf), και απροσδιόριστη (undefined), hopcount: πόσα «βήματα» (hops) «απέχει» ο κόμβος από το Σταθμό Βάσης.

Κάθε κόμβος, έστω v, έχει έναν μετρητή (counter). Όταν ο κόμβος λάβει το control message για πρώτη φορά, θέτει τον μετρητή του στην τιμή Tv. Ο Tv μετράει προς τα κάτω όσο το κανάλι είναι «άεργο» (idle). Σ αυτό το χρονικό διάστημα ο κόμβος επιλέγει σαν πατέρα στο δένδρο, τον κόμβο με τη μεγαλύτερη εναπομείνασα ενέργεια (max residual energy), και τον μικρότερο αριθμό βημάτων από το Σταθμό Βάσης (minimum hop count). Αυτές οι πληροφορίες περιέχονται στο control message. Όταν ο μετρητής μηδενιστεί, τότε ο κόμβος αυξάνει το hopcount κατά 1, και μεταδίδει (broadcasts) κι αυτός με τη σειρά του το control message.

Η διαδικασία συνεχίζεται μέχρις ότου κάθε κόμβος να μεταδώσει μια φορά το control message. Το αποτέλεσμα όλης αυτής της διαδικασίας είναι ένα δένδρο δρομολόγησης με ρίζα το Σταθμό Βάσης.

Το πλεονέκτημα του αλγορίθμου είναι ότι οι κόμβοι με περισσότερη εναπομείνασα ενέργεια (residual energy) έχουν μεγαλύτερη πιθανότητα να γίνουν «γονικοί» κόμβοι (relay-aggregation nodes), δηλ. να μην είναι φύλλα.

4.6.1.2 Συντήρηση (maintenance) του δένδρου

Κάθε κόμβος-αισθητήρας έχει ένα «κατώφλι» (threshold) ενέργειας Pth. Όταν η ενέργεια του κόμβου πέσει κάτω από αυτό το κατώφλι, ο κόμβος μεταδίδει «μηνύματα βοήθειας» (help messages) για Td «χρονικές μονάδες» (time units), και κλείνει τη μονάδα επικοινωνίας του. Ένας κόμβος-παιδί αυτού του κόμβου-πατέρα, μόλις λάβει ένα τέτοιο help message, τότε επιλέγει έναν άλλον κόμβο σαν πατέρα (parent), αλλιώς εισέρχεται σε κατάσταση κίνδυνου (danger state). Εάν ένας κόμβος που βρίσκεται σε danger state δεχτεί ένα hello message από έναν γειτονικό κόμβο v που είναι πιο κοντά στο Σταθμό Βάσης (με μικρότερο hop count), τότε επιλέγει σαν πατέρα τον καινούριο κόμβο.

4.6.1.3 Αποτελέσματα προσομοίωσης.

Το EADAT στο [14] προσομοιώθηκε σε μια περιοχή 160X160 m. Τα αποτελέσματα έδειξαν ότι, έχουμε οικονομία στην ενέργεια που καταναλώνεται στο δίκτυο αλλά και επιμήκυνση του χρόνου «ζωής» του δικτύου (network lifetime) σε σχέση με την περίπτωση να μην κάνουμε καθόλου «συνάθροιση» δεδομένων-μηνυμάτων (data aggregation).

Μια άλλη αξιοσημείωτη παρατήρηση είναι ότι το network lifetime αυξάνεται γραμμικά σε σχέση με την πυκνότητα του δικτύου (δηλ. για συγκεκριμένο αριθμό κόμβων, όσο πιο πυκνό είναι το δίκτυο τόσο μεγαλύτερο είναι το lifetime)

4.6.2 PEDAP (Power Efficient Data Gathering and Aggregation in WSN) (Korpeoglou 2003)

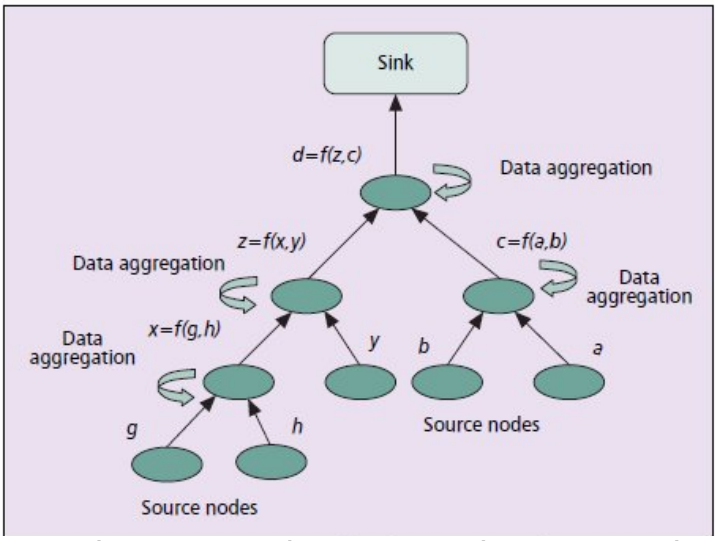
Ένα άλλο πρωτόκολλο δρομολόγησης που χρησιμοποιεί δενδρική δομή είναι το PEDAP, Korpeglou et al[15], (Power-Efficient Data gathering and Aggregation Protocol). Ο σκοπός του PEDAP είναι η μεγιστοποίηση του χρόνου ζωής ενός ΑΔΑ σε όρους «γύρων» συλλογής δεδομένων (data gathering rounds). Το PEDAP είναι ένα πρωτόκολλο που χρησιμοποιεί ένα Ελάχιστο Επικαλύπτον Δένδρο (Minimum Spanning Tree), σαν δομή δρομολόγησης των δεδομένων από τους κόμβους στο Σταθμό Βάσης. Για την κατασκευή του MST χρησιμοποιείται ο αλγόριθμος του Prim. Το πρωτόκολλο έχει καλή απόδοση ακόμη και στις περιπτώσεις όπου ο Σταθμός Βάσης βρίσκεται μέσα στην περιοχή που παρακολουθούμε, σε αντίθεση με το πρωτόκολλο LEACH, το οποίο δεν έχει καλή απόδοση όταν ο Σταθμός Βάσης βρίσκεται μέσα στην περιοχή παρακολούθησης.

Το PEDAP ελαχιστοποιεί την ενέργεια που καταναλώνεται σε κάθε γύρο συλλογής και μετάδοσης δεδομένων, κατασκευάζοντας ένα MST πάνω στο δίκτυο των αισθητήρων. Το ΑΔΑ προσομοιώνεται με έναν γράφο, και το βάρος κάθε ακμής είναι το ποσό ενέργειας που πρέπει να δαπανήσει ο κάθε κόμβος για να επικοινωνήσει με τους γειτονικούς του κόμβους, με βάση τον πιο κάτω τύπο:

$Cij(k) = 2 * Eelec *k + Eamp *k *(dij)^2$

όπου Cij(k), είναι η ενέργεια που χρειάζεται ο κόμβος i για να στείλει στον κόμβο j ένα πακέτο με k bits. Eelec είναι η ενέργεια που καταναλώνει το κύκλωμα εκπομπής-λήψης (transceiver), Eamp είναι η ενέργεια που καταναλώνεται από τον transmit amplifier και dij είναι η απόσταση μεταξύ των 2 κόμβων. (βλ.First Order Radio Model)

Το δένδρο δρομολόγησης (routing tree) κατασκευάζεται με τη χρήση του αλγόριθμου του Prim. Στην Εικόνα11 βλέπουμε μια τέτοια περίπτωση. Σημειωτέον, ότι σε κάθε ενδιάμεσο κόμβο γίνεται και «συνάθροιση» (aggregation) των δεδομένων που μεταδίδουν τα «παιδιά» του κάθε κόμβου στον κόμβο-πατέρα..



Εικ.11: Δίκτυο με πρωτόκολλο δρομολόγησης βασισμένο σε Minimum Spanning Tree. Τα βέλη υποδεικνύουν το μονοπάτι δρομολόγησης και η f(„.), είναι η συνάρτηση «συνάθροισης» των δεδομένων (data aggregation function)

4.6.3 PEDAP-PA (PEDAP Power Aware)

Για να γίνει εξισορρόπηση της ενέργειας που καταναλώνει ο κάθε κόμβος (δηλ. οι κόμβοι να ξοδεύουν παρόμοια ποσά ενέργειας ο καθένας), έχει προταθεί από τους ίδιους συγγραφείς μια παραλλαγή του PEDAP, το PEDAP-PA (Power Aware-PEDAP) [15]. Για να μπορέσουμε να εξισορροπήσουμε το φορτίο ανάμεσα στους κόμβους του δικτύου, πρέπει να ληφθεί υπ' όψιν και η υπολειπόμενη ενέργεια των κόμβων κατά τη διαδικασία της δρομολόγησης αλλά και της «συνάθροισης» των δεδομένων (data aggregating). Αυτό προσπαθεί να το πετύχει η power aware έκδοση του PEDAP, το PEDAP-PA. Αυτό το καταφέρνει, αλλάζοντας το «κόστος» επικοινωνίας μεταξύ 2 κόμβων, το οποίο γίνεται τώρα ECij(k) = Cij(k)/ei, όπου ei είναι η «κανονικοποιημένη» εναπομείνασα ενέργεια (residual energy) του κόμβου. Κανονικοποιημένη σε σχέση με την αρχική ενέργεια του κόμβου. Δηλ. όσο μικρότερο ποσοστό της αρχικής ενέργειας έχει υπόλοιπο ο κόμβος τόσο «βαρύτερη» γίνεται η ακμή (i,j), άρα, για τη δημιουργία του μονοπατιού δρομολόγησης (routing path) «προτιμούνται» οι κόμβοι που έχουν ξοδέψει μικρό ποσοστό από την αρχική τους ενέργεια, και άρα, έχουν ακόμη αποθέματα για αρκετές μεταδόσεις.

Το PEDAP όπως και το PEDAP-PA προϋποθέτει ο Σταθμός Βάσης να γνωρίζει τις θέσεις των υπόλοιπων κόμβων, δηλ. το πρωτόκολλο είναι κεντρικοποιημένο. Η πολυπλοκότητα του πρωτοκόλλου όσο αφορά τη δημιουργία του MST (routing tree) είναι O(n2), όπου n ο αριθμός των κόμβων του δικτύου. Το PEDAP-PA βελτιώνει το χρονικό διάστημα μέχρι να τεθεί εκτός λειτουργίας ο πρώτος κόμβος (first node death) κατά 400%

(4 φορές) σε σχέση με το PEDAP. Στην περίπτωση που ο Σταθμός Βάσης τοποθετείται στο κέντρο του δικτύου, το PEDAP και το PEDAP- PA βελτιώνουν τον χρόνο «θανάτου» του τελευταίου κόμβου (last node death) κατά 2 φορές σε σχέση με το LEACH.

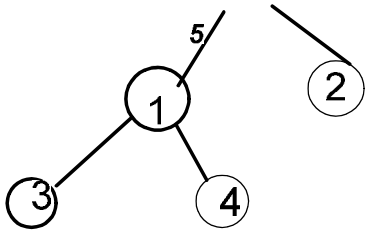
Κεφάλαιο 5ο

Ερευνητική πρόταση

5.1 Εισαγωγή – Αλγόριθμος Prim

Όπως αναφέραμε και στην εισαγωγή της παρούσης εργασίας, σκοπός μας είναι να μεγιστοποιήσουμε τη διάρκεια αξιοποίησης ενός Ασύρματου Δικτύου Αισθητήρων. Θα προσπαθήσουμε να βελτιώσουμε την απόδοση του πρωτοκόλλου PEDAP, κάνοντας κάποια μικρή αλλαγή στον αλγόριθμο με τον οποίο κατασκευάζεται το δένδρο δρομολόγησης (routing tree). Όπως αναφέραμε και στην παράγραφο 4.6.3 το πρωτόκολλο PEDAP χρησιμοποιεί τον αλγόριθμο του Prim για να κατασκευάσει το δένδρο δρομολόγησης.

Ας θεωρήσουμε ένα δίκτυο με 5 κόμβους, κόμβος0, κομβος1,..,κομβος4, με σταθμό βάσης τον κόμβο 0, όπως παριστάνεται στο πιο κάτω σχήμα.



βάσης γίνεται μέσω των ακμών ενός Minimum Spanning Tree το οποίο κατασκευάζεται με τον αλγόριθμο του Prim.

Ο αλγόριθμος του Prim λειτουργεί ως εξής: Σε κάθε βήμα εκτέλεσης του αλγορίθμου υπάρχουν δυο «σύνολα» από κόμβους. Το ένα σύνολο το αποτελούν οι κόμβοι οι οποίοι έχουν ήδη «ενταχθεί» στο MST και το δεύτερο σύνολο, το αποτελούν οι υπόλοιποι κόμβοι, αυτοί που δεν έχουν ακόμη συμπεριληφθεί στο τελικό Ελάχιστο Επικαλύπτον Δένδρο. Σε κάθε βήμα, προσθέτουμε στο Επικαλύπτον Δένδρο (set1) τον κόμβο που δεν ανήκει στο Δένδρο αυτό, και ο οποίος συνδέεται με την ελάχιστου βάρους ακμή (την «ελαφρύτερη») με κάποιον από τους κόμβους που ήδη ανήκουν στο Ελάχιστο Επικαλύπτον Δένδρο.

Ας παρακολουθήσουμε, όμως, την εφαρμογή του αλγορίθμου του Prim (και κατ' επέκταση του πρωτοκόλλου PEDAP) στο δίκτυο του σχ.1

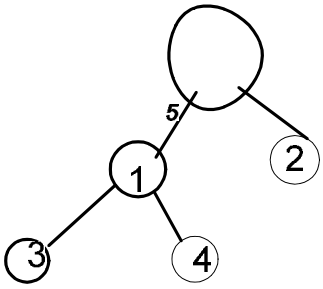
Ας ονομάσουμε set1, το σύνολο των κόμβων που είναι στο τελικό MST και set2 το σύνολο των κόμβων που δεν έχουν ακόμη συμπεριληφθεί στο MST. Αρχικά έχουμε:

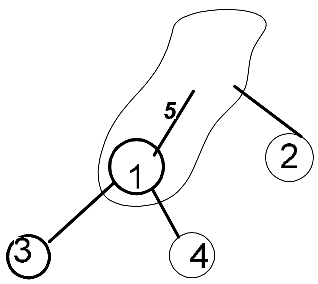
set1={}, set2={κόμβος0, κόμβος1, κόμβος2, κόμβος3, κόμβος4}.(σχ.1)

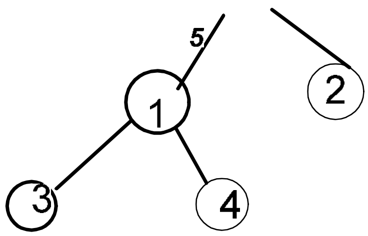
Συμπεριλαμβάνουμε τον κόμβο 0, στο set1, που θα είναι η ρίζα του MST.

Έτσι έχουμε:

set1= {κόμβος0}, set2= {κόμβος1, κόμβος2, κόμβος3, κόμβος4}(σχ2).




5
1
2
3
4


5
1
2
3
4

δηλ Etotal = 2 + (3-1)*1 ⇒

Etotal = 2 + 2*1 ⇒

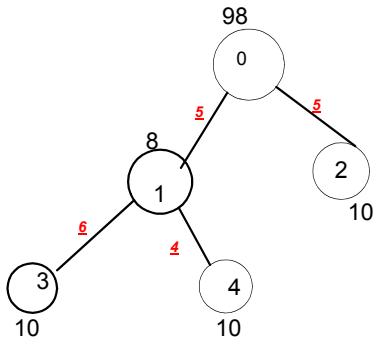
Etotal = 4 μονάδες ενέργειας.

Από την προηγούμενη ανάλυση εύκολα καταλαβαίνουμε ότι ο ρυθμός μείωσης της εναπομείνασας ενέργειας (residual power), στον κόμβο 1, είναι μεγαλύτερος απ' ότι στους άλλους κόμβους. Άρα, ο κόμβος 1, είναι αυτός που είναι «υποψήφιος» να «πεθάνει» πρώτος.

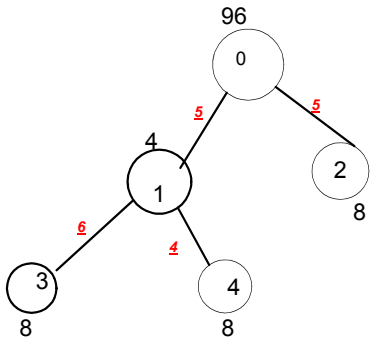
Στον Πίνακα1, φαίνεται η εναπομείνασα ενέργεια για κάθε κόμβο, σε κάθε γύρο συλλογής δεδομένων (data gathering round). Παρατηρούμε ότι το πρόβλημα θα παρουσιαστεί στον κόμβο 1, διότι αυτός έχει τον μεγαλύτερο βαθμό (1 εισερχόμενη και 1 εξερχομένη σύνδεση), άρα ξοδεύει το μεγαλύτερο ποσό ενέργειας σε σχέση με τους υπόλοιπους κόμβους κατά την προώθηση των δεδομένων προς το σταθμό βάσης.

	node				
	0	1	2	3	4
round 0	100	12	12	12	12
round 1	98	8	10	10	10
round 2	96	4	8	8	8

Πινακας1



Σχ8. μετά τον γυρο1



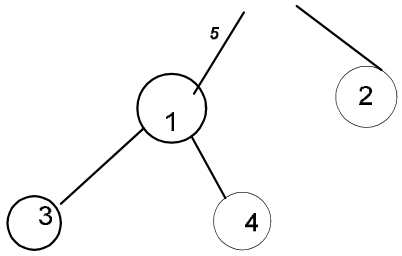
Σχ.9 μετά τον γυρο2

5.2 Τροποποίηση του αλγορίθμου Prim

Στο σημείο αυτό της εκτέλεσης του αλγορίθμου του Prim (δηλ. του PEDAP), θα εισάγουμε την τροποποίησή μας στον αρχικό αλγόριθμο. Όπως είναι το δίκτυο στο σχ.9, δεν μπορεί να συνεχίσει να είναι λειτουργικό, διότι ο κόμβος1 έχει αρκετή ενέργεια για έναν ακόμη γύρο συλλογής-μετάδοσης δεδομένων.

Έτσι, επιλέγουμε να εφαρμόσουμε ένα διαφορετικό κριτήριο για την επιλογή του επόμενου κόμβου που θα συμπεριληφθεί στο Ελάχιστο Επικαλύπτον Δένδρο. Όπως αναφέραμε πιο πάνω, επιλέγαμε από το set2, εκείνον τον κόμβο που συνδέεται με την ελαφρότερη ακμή με κάποιον κόμβο στο set1. Τώρα, θα επιλέξουμε τον κόμβο εκείνο του set2, που έχει τη μεγαλύτερη περίσσεια ενέργειας (max residual power neighbor), από τους κόμβους που συνδέονται με κάποια ακμή με το set1.

Στην περίπτωση, λοιπόν, του σχ.9, ο κομβος4, θα συνδεθεί με τον κομβο2 έστω κι αν η ακμή [1,4] είναι «ελαφρύτερη» από την [2,4]. Σημασία έχει ότι ο κομβος2 έχει 8 μονάδες ενέργειας ενώ ο κομβος1 μόνο 4.



	node				
	0	1	2	3	4
round 0	100	12	12	12	12
round 1	98	8	10	10	10
round 2	96	4	8	8	8
round3	94	1	5	6	6

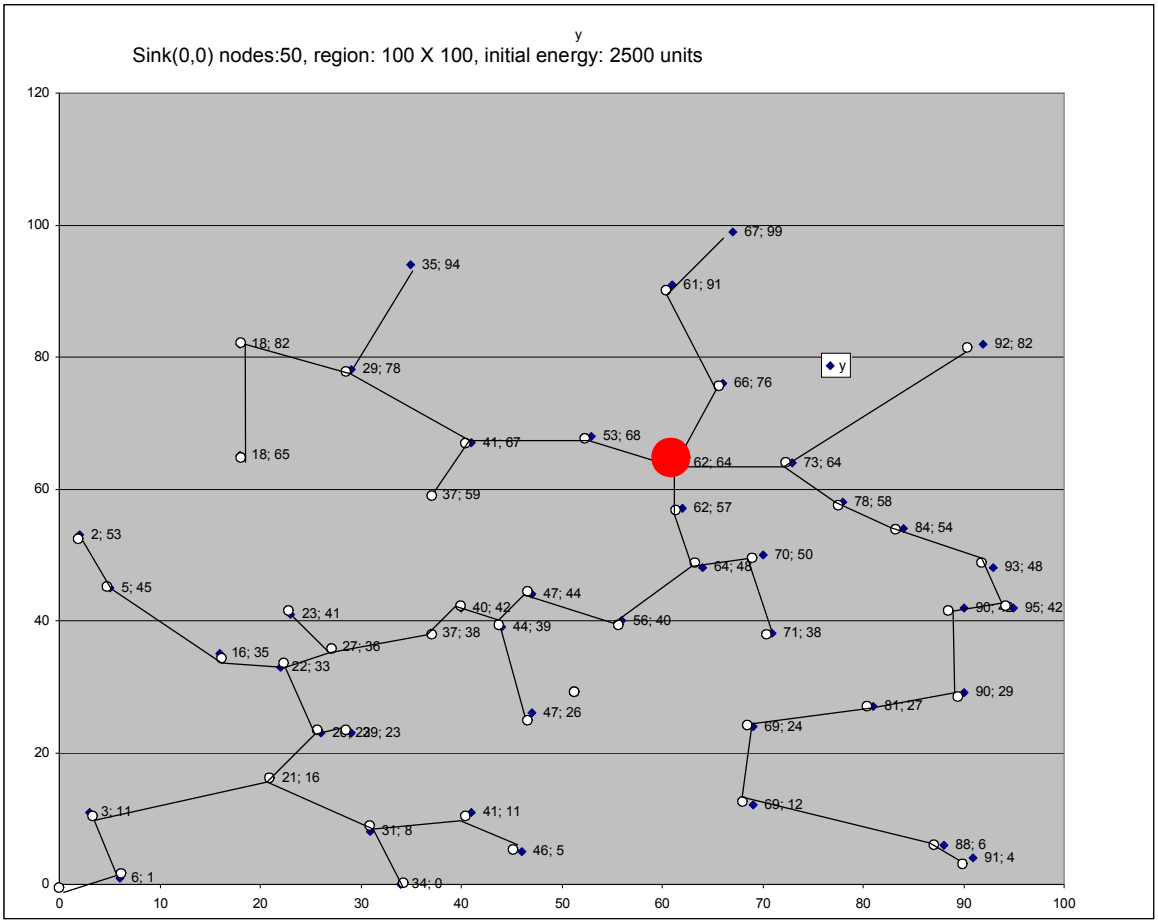
Πινακας2

Όπως βλέπουμε στον Πινακα2 η «ζωή» του δικτύου επεκτείνεται κατά έναν ακόμη γύρο συλλογής δεδομένων.

Μένει λοιπόν, να υλοποιήσουμε τόσο τον αλγόριθμο του Prim, όσο και την τροποποιημένη του μορφή και μέσα από διάφορες προσομοιώσεις να ελέγξουμε την αποδοτικότητα της πρότασής μας.

Πειραματική μελέτη του τροποποιημένου Prim

Όπως αναφέραμε και στην παράγραφο 5.2 προκειμένου να επιτύχουμε αύξηση του χρόνου λειτουργικότητας ενός Ασύρματου Δικτύου Αισθητήρων, θα επιφέρουμε κάποια αλλαγή στον αλγόριθμο του Prim. Ο αλγόριθμος του Prim χρησιμοποιείται από το πρωτόκολλο PEDAP. Το πρωτόκολλο PEDAP δημιουργεί ένα δένδρο δρομολόγησης (routing tree) χρησιμοποιώντας τον αλγόριθμο του Prim.



σχ.11

Στο σχήμα 11, βλέπουμε ένα δίκτυο με 50 κόμβους, σε μια περιοχή 100X100m και ο Σταθμός Βάσης να είναι στο σημείο (0,0), κάτω αριστερή γωνία. Παρατηρούμε πώς ο κόμβος με τον μεγαλύτερο «βαθμό» είναι ο κόμβος με συντεταγμένες (62,64). Ο κόμβος αυτός έχει βαθμό 4, μια εξερχόμενη και 3 εισερχόμενες συνδέσεις. Έτσι είναι πολύ πιθανόν να είναι ο πρώτος κόμβος που θα τεθεί εκτός λειτουργίας. Πράγματι όταν «τρέξουμε» ένα ερώτημα του Σταθμού Βάσης, με τις προδιαγραφές που έχουμε περιγράψει στην παράγραφο για το First Order Radio Model, παρατηρούμε ότι ο κόμβος (62,64) τίθεται εκτός λειτουργίας στον γύρο 959. τον κόμβο αυτό τον έχουμε σημειώσει με κόκκινο χρώμα στο σχ.11.

Για να αυξήσουμε τον χρόνο «ζωής» του δικτύου μας (network lifetime), θα εφαρμόσουμε την παραλλαγή του αλγορίθμου του Prim, για την περιοχή του κόμβου (62,64). Συγκεκριμένα, θα εφαρμόσουμε το κριτήριο του «γείτονα με τη μεγαλύτερη ενέργεια» (max energy neighbor criterion), κατά τη χρονική στιγμή που η εναπομείνασα ενέργεια του συγκεκριμένου κόμβου πέσει κάτω από το μισό της αρχικής ενέργειας.

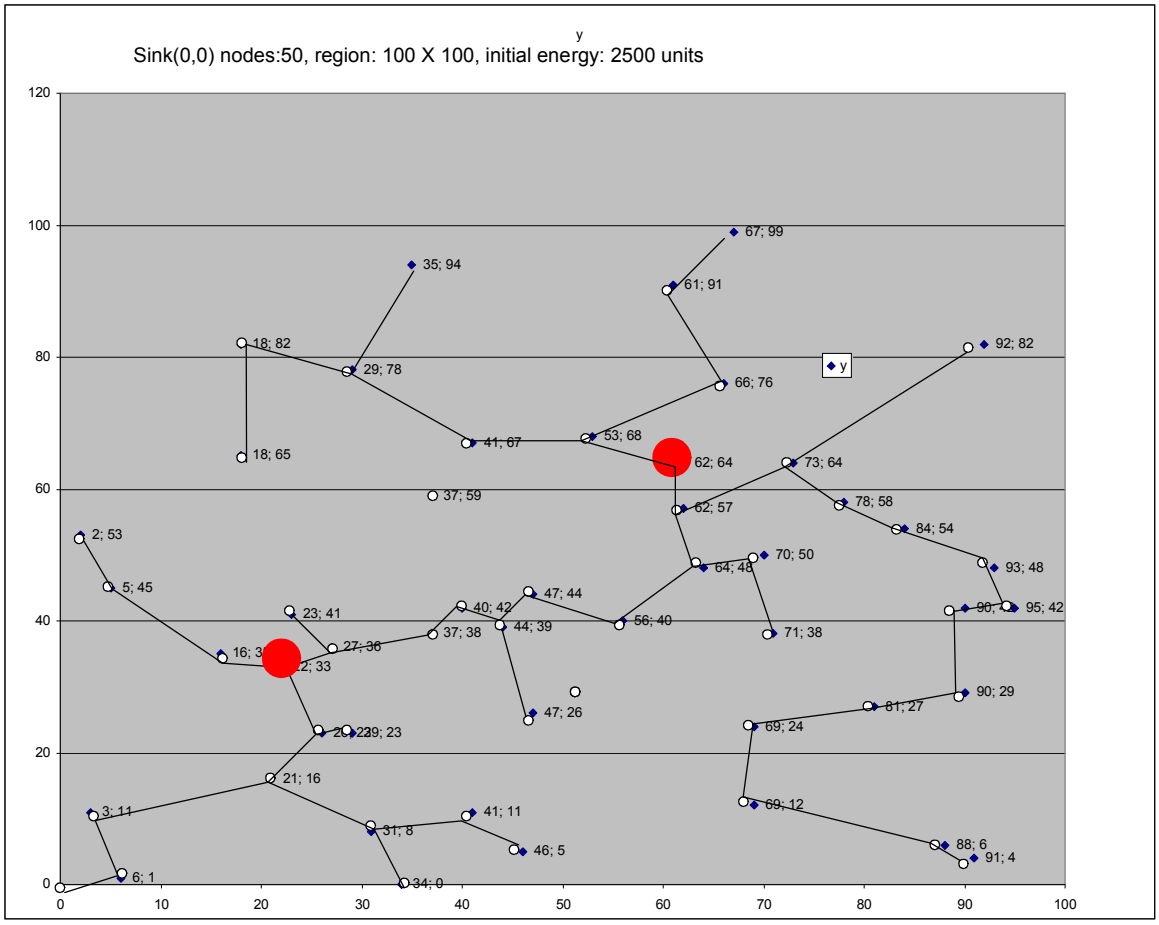
Αρχική ενέργεια=2500 μονάδες ενέργειας, άρα ½*2500 = 1250. Παρατηρούμε (τρέχοντας το πρόγραμμα) ότι στον γύρο 479, η ενέργεια του κόμβου γίνεται 1249 και οι ενέργειες των γειτονικών κόμβων είναι (όπως φαίνεται και στο σχ.11):

Κόμβος(53, 68): 1716 μον.ενέργειας

Κόμβος(73,64): 1459 μον.ενέργειας

Κόμβος(62,57): 1722 μον.ενέργειας

Επιλέγουμε να «καταργήσουμε» τις ακμές (73,64)→ (62,64), και (66,76)→(62,64) (έτσι ο κόμβος (62,64) «ξεφορτώνεται» από 2 εισερχόμενες συνδέσεις), και προσθέτουμε 2 καινούριες ακμές, τις: (73,64)→ (62,57), και (66,76)→(53,68). Προκύπτει το σχ.12



σχ.12

Εφαρμόζοντας το «ερώτημα» (query) του σταθμού βάσης πάνω στο καινούριο δίκτυο παρατηρούμε ότι ο κόμβος που τίθεται εκτός λειτουργίας πρώτος είναι ο (22,33), (σημειώνεται με κόκκινο κύκλο στο σχήμα), αλλά αυτό συμβαίνει στον γύρο 1126, δηλ. έχουμε μια επέκταση της ζωής του δικτύου κατά 167 γύρους, ποσοστό 17% περίπου.

Με βάση το πιο πάνω παράδειγμα πιστεύουμε ότι η επιμήκυνση του χρόνου αξιοποίησης (network lifetime) ενός Ασύρματου Δικτύου Αισθητήρων με τη χρήση της

παραλλαγής του αλγορίθμου του Prim που παρουσιάσαμε, θα είναι δυνατή και σε μεγαλύτερα δίκτυα.

5.3 Συμπεράσματα – συνεισφορά της εργασίας

Στην παρούσα εργασία μελετήσαμε κάποιους αλγορίθμους δρομολόγησης για τα Ασύρματα Δίκτυα Αισθητήρων, και συγκεκριμένα αυτούς που αφορούν τα ιεραρχικά δομημένα δίκτυα και τα δίκτυα τα οποία έχουν δενδρική δομή.

Εισάγαμε μια τροποποίηση στον αλγόριθμο του Prim για τη δημιουργία του δένδρου δρομολόγησης, όσο αφορά την επιλογή του επόμενου κόμβου που θα συμπεριληφθεί στο «τρέχον» Ελάχιστο Επικαλύπτον Δένδρο (Minimum Spanning Tree).

Εφαρμόζοντας αυτόν τον καινούριο αλγόριθμο σε ένα ΑΔΑ με 50 κόμβους σε μια περιοχή 100m X 100m, διαπιστώσαμε «τρέχοντας» το πρόγραμμα που γράψαμε για την υλοποίηση του συγκεκριμένου αλγορίθμου, ότι έχουμε αύξηση του χρόνου αξιοποίησης της λειτουργικότητας του δικτύου, σε όρους «πρώτου κόμβου που θα πεθάνει» (first node death), κατά ένα ποσοστό 17% περίπου.

5.4 Μελλοντική εργασία

Σαν μελλοντική εργασία (future work) σκοπός μας είναι να επεκτείνουμε το πρόγραμμα προσομοίωσης Ασύρματων Δικτύων NS2 [29], με νέες κλάσεις ή να τροποποιήσουμε κάποιες από τις ήδη υπάρχουσες, έτσι ώστε να ελέγξουμε την αποδοτικότητα της ερευνητικής μας πρότασης σε ένα περισσότερο ρεαλιστικό περιβάλλον, όπως το NS2.

Βιβλιογραφία - Αρθρογραφία

[1] S. Gobriel. "Energy-efficient design of ad-hoc and sensor networks", M.Sc, University of Pittsburgh, 2008

[2] Y. Chen and Nasser. “Enabling QoS multipath routing protocol for wireless sensor networks,” in IEEE International Conference, 2008, pp. 2421 – 2425.

[3] T. Zia and A. Zomaya. “Security issues in wireless sensor networks,” in Proceedings of the international Conference on Systems and Networks Communication, 2006. [4] Y. Wang, G. Attebury and B. Ramamurthy. “A survey of security issues in wireless sensor networks,” IEEE communication surveys, Vol.8, No.2, 2006.

[4] M. Ding, X. Cheng and G. Xue, “Aggregation Tree Construction in Sensor Networks,” 2003 IEEE 58th Vehic. Tech. Conf., vol. 4, no. 4, Oct. 2003, pp. 2168–72.

[5] A.al-yasiri and A.sunley. “Data aggregation in wireless sensor networks using the SOAP protocol,” Journal of Physics Conference Series 76, 2007.

[6] A.Khetrapal, “Routing techniques for Mobile Ad Hoc Networks Classification and Qualitative/ Quantitative Analysis,” Department of Computer Engineering, Delhi College of Engineering University.

[7] M. N. Elshakankiri, M. N. Moustafa and Y. H. Dakroury. “Energy Efficient Routing Protocol for Wireless Sensor Networks,” in International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Dec. 2008, pp. 393 – 398.

[9] A. Habib. “Sensor network security issues at network layer,” in 2nd International Conference on Advances in Space Technologies Islamabad, Pakistan, Nov. 2008, pp. 58-63.

[10] A. A. Ahmed, H. Shi and Y. Shang. “A survey on network protocols for wireless sensor networks,” in Proceedings of Information Technology: Research and Education, Aug. 2003, pp. 301- 305.

[11] I. Akyldiz et al., “A Survey on Sensor Networks,” IEEE Commun. Mag., vol. 40, no. 8, Aug. 2002, pp. 102–114

[12]D. Estrin, L. Girod, G. Pottie, M. Srivastava, “Instrumenting the world with wireless sensor networks”, International Conference on Acoustics, Speech, and Signal Processing, Salt Lake City, Utah, May 2001.

[13] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, “Next century challenges: scalable coordination in sensor networks”, ACM MobiCom’99, Washington, USA, 1999.

[14] A. Chandrakasan, R. Amirtharajah, S. Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, A. Wang, “Design considerations for distributed micro-sensor systems”, Proceedings of the IEEE 1999 Custom Integrated Circuits Conference, San Diego, CA, May 1999.

[15] L.E. Larson, “Radio Frequency Integrated Circuit Technology for Low-Power Wireless Communications”, IEEE Personal Communications, 1998.

[16]B.G. Celler et al., “An instrumentation system for the remote monitoring of changes in functional health status of the elderly”, International Conference IEEE-EMBS, New York, 1994.

[17] P. Bonnet, J. Gehrke, P. Seshadri, Querying the physical world, IEEE Personal Communications (October 2000) 10–15.

[18] http://www.alertsystems.org.

[19] N. Bulusu, D. Estrin, L. Girod, J. Heidemann, Scalable coordination for wireless sensor networks: self-configuring localization systems, International Symposium on Communication Theory and Applications (ISCTA 2001), Ambleside, UK, July 2001.

[20] J.M. Kahn, R.H. Katz, K.S.J. Pister, Next century challenges: mobile networking for smart dust, Proceedings of the ACM MobiCom’99, Washington, USA, 1999, pp. 271–278.

[21] N. Noury, T. Herve, V. Rialle, G. Virone, E. Mercier, G. Morey, A. Moro, T. Porcheron, Monitoring behavior in home using a smart fall sensor, IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology, October 2000, pp. 607–610.

[22] D. Estrin, et. al., http://nesl.ee.ucla.edu/tutorials/mobicom02

[23]S. Madden, M. J. Franklin and J.M. Hellerstein and W. Hong, TAG: a tiny aggregation service for ad-hoc sensor networks, to appear in OSDI 2002.

[24] J. Hill, A software architecture to support network sensors, Master’s Thesis, UC Berkeley, 2000.

[25] J. Hill, R. Szewczyk, A. Woo, S. Hollar, and J. Heidemann, System architecture directions for networked sensors, Proc. 9th International Conference on Architectural Support for Programming Languages and Operating Systems, November 2000.

[26] Ramesh Rajagopalan and Pramod K. Varshney, Syracuse University, Data-aggregation Techniques in Sensor Networks: a Survey, 2006

[27] E.M. Petriu, N.D. Georganas, D.C. Petriu, D. Makrakis, V.Z. Groza, Sensor-based information appliances, IEEE Instrumentation and Measurement Magazine (December 2000) 31–35.

[28] Wireless Sensor Networks, A networking perspective, Jun Zheng, Abbas Jamalipour, John Wiley & Sons, 2009

[29] http://www.isi.edu/nsnam/ns/

//Prim_random_net.cpp

```
//runs Prim's algorithm in a weighted graph
//and "stores" the MST in a 2 dim array
//next, runs a query and counts how many rounds do the network stays "alive"
//before 1st node dies.
#include <stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define N 20
#define TRUE 1
#define FALSE 0
FILE *fp;
int NETWORK_X = 100;          // X-size of network
int NETWORK_Y = 100;          // Y-size of network
//create a random network
struct sensor{
      int xLoc;
      int yLoc;
};
int mst2[N][N];
float d[N];
/* d[i] is the distance between node i and the minimum spanning
tree; this is initially infinity (100000); if i is already in
the tree, then d[i] is undefined; this is just a temporary variable.
It's not necessary but speeds up execution considerably (by a factor of n) */
int whoTo[N];
 /* whoTo[i] holds the index of the node i would have to be
                              linked to in order to get a distance of d[i] */
/* updateDistances(int target)
          should be called immediately after target is added to the tree;
```

```
        updates d so that the values are correct (goes through target's
        neighbours making sure that the distances between them and the tree       are indeed minimum)
*/
void Prim(float graph[][N], float mst[][N]);
void updateDistances(int target);
void init_mst(float graph[][N]);
void print_graph(int g[][N]);
void print_energy_level(float energy[]);
void array_to_file(int A[][N],char* file_name);
void array_from_file(int A[][N], char* file_name);
void print_1d_array(float A[N]);

void compute_Receive_energy(float Er[N]);
void compute_edge_costs(float C[][N], float Et[][N], float Er[N]);
//array C[N][N] will be the input to the Prim algorithm
void print_distances(float dist[N][N]);
void print_graph(float Costs[][N]);

int k=1024;//num of bits per data packet
float Et[N][N];//Etransmit, to transmit from i to j
float Er[N];//Ereceive
float C[N][N];//cost of edge (i,j)
float dist[N][N];//distances between nodes
float Eelec = 0.0005;//E tranceiver electronic circuit, 50nJ/bit
float Eamp =  0.000001;//E transmitter amplifier, 100pJ/bit/m^2
float Etotal = 0.0; //initial node energy 1,000,000 nJ
float energy[N];//energy level of each node
float initial_energy[N];
float distances[N][N];
int change;

//****************** main  *************************
```

```
int main(int argc, char *argv[]) {

   struct sensor * network = (struct sensor *) malloc(N * sizeof(struct sensor));
   fp=fopen("network.txt","w");
         float mst[N][N]; //mst[][] is the mst represented as an adjacent matrix

         float costs_mst[N][N];// mst with edges the cost from i to j
         int degree[N];//degree of each node
   int edges=0;//num of edges (out+in) of each node
   //sink energy
   energy[0]=10000000000000000.0;//sink has 100Joules
   //nodes' energy levels

   for (int i=1;i<N;i++){
    energy[i]=2500; //node energy in Joules
    initial_energy[i]=energy[i];
   }
   //create random network;
   network[0].xLoc=0;network[0].yLoc=0;
   for(int i=1; i<N;i++){
      printf("[%d]: ",i);
      network[i].xLoc= rand() % NETWORK_X;printf("xloc= %d ",network[i].xLoc);
      network[i].yLoc= rand() % NETWORK_Y;printf("yloc= %d \n",network[i].yLoc);
   }
   getch();
   //COMPUTE DISTANCES
   for(int i=0;i<N;i++){
      for (int j=0;j<N;j++){
         int dist_x= network[i].xLoc-network[j].xLoc;
         int dist_y= network[i].yLoc-network[j].yLoc;
         distances[i][j] = sqrt(pow(dist_x,2) + pow(dist_y,2));
         printf("Dist[%d][%d]= %.0f\n",i,j,distances[i][j]);
```

```
        }
        printf("\n");
    }
    init_mst(mst);//initialize mst distances with zeros
    init_mst(costs_mst);//initialize edges' costs mst with zeros
    printf("DISTANCES:\n\n");print_graph(distances);
    fprintf(fp,"DISTANCES GRAPH\n");
    for(int i=0; i<N; i++){
                for(int j=0;j<N;j++) fprintf(fp, "%4.0f",distances[i][j]);
    fprintf(fp, "\n");
    }
 printf("\n");
    compute_Receive_energy(Er);
    //RECEIVE ENERGY
    fprintf(fp,"\nEreceive\n");
    for(int i=0;i<N;i++) fprintf(fp, "Er[%d]= %f\n",i,Er[i]);

    for (int i=0;i<N;i++){
      for (int j=0;j<N;j++){
         if (i!=j) Et[i][j]=k*Eelec + k*Eamp*pow(distances[i][j],2);
         printf("Etransmit[%d][%d]= %f\n",i,j,Et[i][j]);
      }
      getch();
    }
    //compute edge costs- C[i][j]= communication cost between node[i] and node[j]
    compute_edge_costs(C,Et,Er);
    //print distances-Etransmit-Edges' costs
    //EDGE COSTS
    fprintf(fp, "EDGE COSTS\n");
    for(int i=0; i<N; i++){
      for(int j=0;j<N;j++){
```

```
      fprintf(fp,"Dist[%d][%d]=  %2.0f  __  Et[%d][%d]=  %.3f  >>  C[%d][%d]=
%.3f\n",i,j,distances[i][j],i,j,Et[i][j],i,j,C[i][j]);
      //printf("Dist[%d][%d]=  %2.0f  __  Et[%d][%d]=  %.3f  >>  C[%d][%d]=
%.3f\n",i,j,distances[i][j],i,j,Et[i][j],i,j,C[i][j]);
    }
    printf("\n");fprintf(fp,"\n");
  }
  getch();
  //runs prim with distances as edges weights
  //Prim(distances, mst);
  //trexei o prim me edge costs to kostos epikoinwnias C[i][j]

  Prim(C, costs_mst);

  printf("\n COSTS MINIMUM SPANNING TREE:\n\n");
  fprintf(fp, "\n COSTS MINIMUM SPANNING TREE:\n\n");

  for(int i=0; i<N; i++){
              for(int j=0;j<N;j++){
             fprintf(fp, "%.3f\n  ",costs_mst[i][j]);
                    printf("%.3f  ",costs_mst[i][j]);
              }
              printf("\n");fprintf(fp,"\n");
}
printf("\n");getch();
  printf("edge costs:\n\n");
  for(int i=0;i<N; i++){
    for(int j=0; j<N; j++)
      if (costs_mst[i][j]!=0.0) {
        printf("C[%d][%d]= %f ",i,j,C[i][j]);
        printf("costs_mst[%d][%d]= %f\n",i,j,costs_mst[i][j]);
      }
```

```
}//end print edge costs
printf("!!!!!!!!!!!!!!!!!!!!!!!!");
getch();
//compute degree of each node in MST tree
for(int i=0; i<N;i++){
        for(int j=0;j<N;j++)
                if (costs_mst[i][j]!=0) edges++;
        degree[i]=edges;
        edges=0;
}
//print degree of each node
fprintf(fp, "DEGREES\n");
for(int i=0;i<N;i++){
  printf("Edges[%d]= %d \n",i,degree[i]);
  fprintf(fp,"%d\n",degree[i]);
}
getch();
float Etransmit[N];//here we have to add some more code to compute Etransmit[i].....


//RUN THE QUERY
int dangerous_node;
change = FALSE;
int dead_node=0;
int node_off=-1;
int round =1;
//run the query
fprintf(fp,"ENERGY LEVELS\n");
while (!dead_node){
  printf("round: %d \n",round);
  fprintf(fp,"round: %d \n",round);
  for(int i=1; i<N; i++){
```

```
    // Ereceive = (incomming links)*Er[i]
    // Etransmit = Er[i] (we have 1 outcoming link)
    energy[i] -= (degree[i]-1)*Er[i] + Etransmit[i];
    if (energy[i]<(initial_energy[i]/2)) {
                                change=TRUE;
                                dangerous_node=i;
                                fprintf(fp,"dangerous state node: %d ",dangerous_node);
                                fprintf(fp,"energy[%d] = %f ",dangerous_node,energy[dangerous_node]);
                                printf("dangerous state node: %d ",dangerous_node);
                                printf("energy[%d] = %f ",dangerous_node,energy[dangerous_node]);
                                //break;
                                }
    if (energy[i]<=0) {dead_node=1;node_off=i;}
    fprintf(fp,"%d %.3f\n",i,energy[i]);
    printf("energy[%d] = %f\n",i,energy[i]);
    }
    fprintf(fp,"\n");
    round++;
  }
  if (change){
     printf("dangerous node: %d ",dangerous_node);
  }
  printf("total rounds: %d, node off: %d \n",round, node_off);
  fprintf(fp, "total rounds: %d, node off: %d \n",round, node_off);
  fclose(fp);
  getch();getch();
return 0;
}
//************* PRIM ****************************
void Prim(float graph[][N], float mst[][N]){

char inTree[N];/* inTree[i] is 1 if the node i is already in the minimum spanning tree; 0 otherwise*/
```

```
 /* Initialise d with infinity */
        for (int i = 0; i < N; ++i)
                d[i] = 100000;
        /* Mark all nodes as NOT beeing in the minimum spanning tree */
        for (int i = 0; i < N; ++i)
                inTree[i] = 0;
int sink=0;
        /* Add the first node=sink to the tree */
        printf("Adding node %d\n",sink);
        inTree[sink] = 1;// sink is in the MST
        updateDistances(sink);


        float total = 0.0;
        int treeSize;//iterator
        int v_last, v_new;
        //main loop of Prim algorithm!!!!!!!!!!!!!
        for (treeSize = 1; treeSize < N; treeSize++) {
    printf("from node %d \n", treeSize);
        /* Find the node with the smallest distance to the tree */
                int min = -1;
                for (int i = 0; i < N; i++)
                        if (!inTree[i])
                                if ((min == -1) || ( d[i] < d[min])) min = i;


      /* And add it */
                printf("Adding edge %d-%d\n", whoTo[min], min);
                inTree[min] = 1;
                total += d[min];
                //update MST //very important step!!!!
                v_last=whoTo[min];
                v_new=min;
```

```
                mst[v_last][v_new] = graph[v_last][v_new];
                printf("costs_mst[%d][%d]= %.0f\n",v_last,v_new,mst[v_last][v_new]);
                mst[v_new][v_last]=mst[v_last][v_new];//mst is symmetric array

                updateDistances(min);
        }
}//end Prim
//*************** PRINT 1D ARRAY *********************
void print_1d_array(float A[N]){
    for(int i=0; i<N;i++)
     printf("Er[%d]= %f\n",i,A[i]);
}
//*************  Compute Edge Costs  ***********************
void compute_edge_costs(float C[][N], float Et[][N], float Er[N]){
    for(int i=0;i<N;i++){
      for (int j=0;j<N;j++)
        if (i==j) C[i][j]=0.0;
        else C[i][j]=Et[i][j] + Er[j];
    }
}
//************ RECEIVE ENERGY OF each node ***************
void compute_Receive_energy(float Er[N]){
    for(int i=0;i<N;i++) Er[i]=k*Eelec;
}

//*************  PRINT DISTANCES  ***********************
void print_distances(float dist[N][N]){
   char answer;

   printf("distances...\n");
        for(int i=0; i<N; i++)
        {
```

```
                for(int j=0;j<N;j++)
                        printf("%6.1f",dist[i][j]);
                printf("\n");
        }
   printf("press any key...");
   scanf("%c",&answer);
}//end print_distances
//----------- PRINT ENERGY LEVELS ----------------
void print_energy_level(float energy[]){
    printf("\n");
   for(int i=0; i<N; i++) printf("%.2f  ",energy[i]);
}
//**********  UPDATE DISTANCES ***************
void updateDistances(int target) {
        int i;
        for (i = 0; i < N; ++i)
                if ((distances[target][i] != 0) && (d[i] > distances[target][i])) {
                        d[i] = distances[target][i];
                        whoTo[i] = target;
                }
}

void print_graph(float graph[][N]){

   for(int i=0; i<N; i++){
                for(int j=0;j<N;j++)
                        printf("%f  ",graph[i][j]);
                printf("\n");
 }
 printf("\n");
}
//************** INIT MST ****************
```

```
void init_mst(float graph[][N]){
for(int i=0;i<N;i++)
                for(int j=0;j<N;j++)
                    graph[i][j]=0.0;
}
//we can store the graph in a file for easier simulation of different cases
//---------------- ARRAY TO FILE ---------------
void array_to_file(float A[][N],char* file_name){

    FILE *fp;

    fp=fopen(file_name,"w");
    if (!fp){
          printf("error opening file..exit!\n");
          getchar();
          exit(1);
          }
    for (int i=0;i<N;i++){
       for(int j=0;j<N;j++){
            fprintf(fp, "%f ",A[i][j]);
            }
       fprintf(fp,"\n");
    }
    fclose(fp);
}//end array to file
//------------------- READ ARRAY FROM FILE -----
void array_from_file(float A[][N], char* file_name){
    FILE *fp;
    char ch;
    fp=fopen(file_name,"r");
    if (!fp){
          printf("error opening file..exit!\n");
```

```
            getchar();
            exit(1);
            }
    for (int i=0;i<N;i++){
       for(int j=0;j<N;j++){
            fscanf(fp, "%f ",&A[i][j]);
            //printf("%d ",A[i][j]);getchar();
            }
    }
    fclose(fp);
}
```

```
//************  my_Prim.cpp ****************************
//runs "modified" Prim's algorithm in a weighted graph
//and "stores" the MST in a 2 dim array
//next, runs a query and counts how many rounds do the network stays "alive"
//before 1st node dies.
#include <stdio.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
#include <assert.h>
#include<math.h>
#include<time.h>
#define N 10

int NETWORK_X = 50;          // X-size of network
int NETWORK_Y = 50;          // Y-size of network
//create a random network
struct sensor{
      int xLoc;
      int yLoc;
};
int inTree[N];
/* inTree[i] is 1 if the node i is already in the minimum
                              spanning tree; 0 otherwise*/
float d[N];
/* d[i] is the distance between node i and the minimum spanning
tree; this is initially infinity (100000); if i is already in
the tree, then d[i] is undefined; this is just a temporary variable.
It's not necessary but speeds up execution considerably (by a factor of n) */
int whoTo[N];
 /* whoTo[i] holds the index of the node i would have to be
                              linked to in order to get a distance of d[i] */
```

```
/* updateDistances(int target)
        should be called immediately after target is added to the tree;
        updates d so that the values are correct (goes through target's
        neighbours making sure that the distances between them and the tree      are indeed minimum)
*/
float mst[N][N]; //mst[][] is the mst represented as an adjacent matrix
int tam[N];
int label[N];
int nodes_degree[N];
bool usedNode[N];
bool usedEdge[N][N];
void unionSet(int a, int b)
{
        if(tam[a] < tam[b])
        {
                tam[b] += tam[a];
                tam[a] = 1;
                label[a] = label[b];
        }
        else
        {
                tam[a] += tam[b];
                tam[b] = 1;
                label[b] = label[a];
        }
}

bool completeProcess()
{
        for(int i=0; i<N; i++)
        {
                if(usedNode[i] == false)
```

```
                {
                        return true;
                }
        }
        return false;
}
int findSet(int x)
{
        if(x != label[x])
        {
                label[x] = findSet(label[x]);
        }
        return (label[x]);
}
int find_max_neighbor(float distances[][N], float energy[], int max_degree)
{
   int max_node = -1;
   float max_energy = 0.0;
        int i;

        //Find already used max-energy node to choose next max-energy node.
        float tmp_max_energy = 0.0;
        int current_node = -1;

        for(i=1; i<N; i++)
        {
                if(usedNode[i] == true)
                        continue;

                if(tmp_max_energy < energy[i] && inTree[i] == 1 && nodes_degree[i] < max_degree)
                {
                        tmp_max_energy = energy[i];
```

```
                current_node = i;
            }
        }

        if(current_node != -1)
        {
            usedNode[current_node] = true;
            printf(">>current node: %d\n", current_node);

            for(i=1;i<N;i++)
            {
                int a = findSet(current_node);
                int b = findSet(i);
                if(a == b)
                {
                    continue;
                }
                //find neighbor with max energy
                printf(">>i:%d energy[%d]:%.2f\n", i, i, energy[i]);
                //getchar();

                if(distances[current_node][i] != 0 && energy[i] > max_energy &&
nodes_degree[i] < max_degree)
                {
                    max_node=i;
                    max_energy = energy[i];
                    printf(">>max node: %d max_energy: %.2f \n",max_node, max_energy);
                    //getchar();
                }
            }

            if(max_node != -1)
```

```
                {
                        mst[current_node][max_node] = 1;
                        mst[max_node][current_node] = 1;//mst is a symmetric matrix
                        //test print added edges
                        printf(">>mst[%d][%d] = %d\n",current_node, max_node, mst[current_node][max_node]);getchar();
                        usedEdge[max_node][current_node] = true;
                        unionSet(current_node, max_node);
                }
        }
        if(max_node != -1)
        {
                nodes_degree[current_node]++;
                nodes_degree[max_node]++;
        }
/*This is another optimization of our heuristic.
If a neighbor node has max energy level, but it has many nodes connected with it,
so the probability to “die” is big, we try to choose another node.
The criterion is the degree of the node. If the degree of the node is bigger than “maxdegree” then we choose another node
In the furure we will work on this different scenario
*/

        //Little modification for WSN correction
        if(max_node == -1)
        {
                float tmp_max = 0.0;
                for(i=1; i<N; i++)
                {
                        if(usedNode[i] == false && inTree[i] == 0)
                        {
                                if(tmp_max < energy[i] && nodes_degree[i] < max_degree)
```

```
                                        {
                                                tmp_max = energy[i];
                                                max_node = i;
                                        }
                                }
                        }
                }
    return max_node;
}
void Prim(float graph[][N], float mst[][N]);
void updateDistances(int target);
void init_mst(float graph[][N]);
void print_graph(int g[][N]);
void print_energy_level(float energy[]);
void array_to_file(int A[][N],char* file_name);
void array_from_file(int A[][N], char* file_name);
void print_1d_array(float A[N]);

void compute_Receive_energy(float Er[N]);
void compute_edge_costs(float C[][N], float Et[][N], float Er[N]);
//array C[N][N] will be the input to the Prim algorithm
void print_distances(float dist[N][N]);
void print_graph(float Costs[][N]);

int k=1024;//num of bits per data packet
float Et[N][N];//Etransmit, to transmit from i to j
float Etransmit[N];//Etransmit to transmit through costs_mst edges
float Er[N];//Ereceive
float C[N][N];//cost of edge (i,j)
float Eelec = 0.00000005;//E tranceiver electronic circuit, 50nJ/bit
float Eamp =  0.0000000001;//E transmitter amplifier, 100pJ/bit/m^2
float Etotal = 0.0; //initial node energy 1,000,000 nJ
```

```
float energy[N];//energy level of each node
float distances[N][N];

#define TRACE(x)
#define WATCH(x) TRACE( cout << #x " = " << x << "\n" )
#define PRINT(x) TRACE( printf (x) )
#define REP(i, x) for (int i = 0; i < x; ++i)
#define FOR(i, x, y) for (int i = x; i < y; ++i)
//Union-Find algorithm for Cycle Detection

//****************** main  **************************
int main(int argc, char *argv[]) {
   int sink=0;
   struct sensor * network = (struct sensor *) malloc(N * sizeof(struct sensor));



        float costs_mst[N][N];// mst with edges the cost from i to j
        int degree[N];//degree of each node
   int edges=0;//num of edges (out+in) of each node
   //sink energy
   energy[0]=10000000000000000.0;//sink has 100Joules
   //nodes' energy levels
   for (int i=1;i<N;i++)
     energy[i]=1.0; //0.25 Joules
   //create random network;
   network[0].xLoc=0;network[0].yLoc=0;
   //srand(time(NULL));
   for(int i=1; i<N;i++){
      printf("[%d]: ",i);
      network[i].xLoc= rand() % NETWORK_X;printf("xloc= %d ",network[i].xLoc);
      network[i].yLoc= rand() % NETWORK_Y;printf("yloc= %d \n",network[i].yLoc);
```

```
    }
    getch();
    //compute distances
    for(int i=0;i<N;i++){
        for (int j=0;j<N;j++){
            int dist_x= network[i].xLoc-network[j].xLoc;
            int dist_y= network[i].yLoc-network[j].yLoc;
            distances[i][j] = sqrt(pow(dist_x,2) + pow(dist_y,2));
            printf("Dist[%d][%d]= %.0f\n",i,j,distances[i][j]);
        }
        printf("\n");
    }
    getch();
    init_mst(mst);//initialize mst distances with zeros
    init_mst(costs_mst);//initialize edges' costs mst with zeros
    printf("DISTANCES:\n\n");print_graph(distances);
    compute_Receive_energy(Er);print_1d_array(Er);
    for (int i=0;i<N;i++){
       for (int j=0;j<N;j++){
           if (i!=j) Et[i][j]=k*Eelec + k*Eamp*pow(distances[i][j],2);
           printf("Etransmit[%d][%d]= %f\n",i,j,Et[i][j]);
       }
    }
    //compute edge costs- C[i][j]= communication cost between node[i] and node[j]
    compute_edge_costs(C,Et,Er);
    //print distances-Etransmit-Edges' costs
    for(int i=0; i<N; i++){
      for(int j=0;j<N;j++){
        printf("Dist[%d][%d]= %.0f __ Et[%d][%d]= %f >> C[%d][%d]=
%f\n",i,j,distances[i][j],i,j,Et[i][j],i,j,C[i][j]);
      }
      printf("\n");
```

```
}
getch();

        printf("Adding node %d\n", sink);
        inTree[sink] = 1;
int total = 0;
        int treeSize;
        int v_last, v_new;
        int next_node;
        int neighbor_index = 0;
        memset(usedNode, 0, sizeof(usedNode));
        memset(inTree, 0, sizeof(inTree));
        //memset(usedEdge, 0, sizeof(usedEdge));

        for(int i=0; i<N; i++)
        {
                label[i] = i;
                tam[i] = 0;
                nodes_degree[i] = 0;
        }
        for(int node=0; node<1; node++)
        {
   if(node == 0)
                {
                        printf("current node: %d\n",node);
                }
                else
                {
                        printf("\n\ncurrent node: %d\n",node);
                }
                assert(node < N);
```

```
                if(node == 0)
                {
                        for(int kk=0; kk<N; kk++)
                        {
                                if(distances[node][kk] != 0 && inTree[kk] == 0)
                                {
                                        printf("neighbor index: %d\n", kk);
                                        inTree[kk]=1;
                                        neighbor_index++;
                                        //For Degree Heuristic
                                        nodes_degree[kk]++;
                                        mst[node][kk] = 1;
                                        mst[kk][node]=1;//mst is a symmetric  matrix
                                        printf("mst[%d][%d]          =          %d\n",node,          kk,
mst[node][kk]);getchar();
                                        unionSet(node, kk);
                                        //usedEdge[node][kk] = usedEdge[kk][node] = true;
                                }
                        }
                }
                //end
        }
        //examine all neighbors of node "node"
        //Worst case -> O( N^2 lgN )
        usedNode[sink] = true;
   while(completeProcess())
        {
                //Important: >> Maximum Degree used here 4
                int max_degree = 4;
                next_node = find_max_neighbor(distances, energy, max_degree);
                if(next_node == -1)
                        break;
```

```
                printf(">>next node: %d\n", next_node);
                getchar();
                inTree[next_node]=1;
                neighbor_index++;
        }
printf("distances MINIMUM SPANNING TREE:\n\n");print_graph(mst);
printf("edge costs:\n\n");
for(int i=0;i<N; i++){
  for(int j=0; j<N; j++)
    if (costs_mst[i][j]!=0.0) {
        printf("C[%d][%d]= %f ",i,j,C[i][j]);
        printf("costs_mst[%d][%d]= %f\n",i,j,costs_mst[i][j]);
    }
}//end print edge costs
printf("!!!!!!!!!!!!!!!!!!!!!!!");getch();
print_graph(costs_mst);
getch();
//compute degree of each node in MST tree
for(int i=0; i<N;i++){
        for(int j=0;j<N;j++)
                if (costs_mst[i][j]!=0) edges++;
        degree[i]=edges;
        edges=0;
}
//ptint degree of each node
for(int i=0;i<N;i++)
  printf("Edges[%d]= %d \n",i,degree[i]);

getch();
//run a query for 1 round
float Etotal=0.0;
int dead_node=0;
```

```
    int node_off=-1;
    int round =1;
    //run the query
    while (!dead_node){
      printf("round: %d \n",round);
      for(int i=1; i<N; i++){
      // Ereceive = (incomming links)*Er[i]
      // Etransmit = Er[i] (we have 1 outcoming link)
      energy[i] -= (degree[i]-1)*Er[i] + Etransmit[i];
      if (energy[i]<=0) {dead_node=1;node_off=i;}
      //printf("energy[%d] = %f\n",i,energy[i]);
      Etotal += (degree[i]-1)*Er[i] + Etransmit[i];
      }
      round++;
    }
    printf("total rounds: %d, node off: %d \n",round, node_off);
    printf("Etotal= %f\n",Etotal);
    getch();getch();
return 0;
}
//************* PRIM ****************************
void Prim(float graph[][N], float mst[][N]){

char inTree[N];/* inTree[i] is 1 if the node i is already in the minimum spanning tree; 0 otherwise*/


     /* Initialise d with infinity */
            for (int i = 0; i < N; ++i)
                    d[i] = 100000;
            /* Mark all nodes as NOT beeing in the minimum spanning tree */
            for (int i = 0; i < N; ++i)
                    inTree[i] = 0;
    int sink=0;
```

```
        /* Add the first node=sink to the tree */
        printf("Adding node %d\n",sink);
        inTree[sink] = 1;// sink is in the MST
        updateDistances(sink);

        float total = 0.0;
        int treeSize;//iterator
        int v_last, v_new;
        //main loop of Prim algorithm!!!!!!!!!!!!!
        for (treeSize = 1; treeSize < N; treeSize++) {
    printf("from node %d \n", treeSize);
        /* Find the node with the smallest distance to the tree */
                int min = -1;
                for (int i = 0; i < N; i++)
                        if (!inTree[i])
                                if ((min == -1) || ( d[i] < d[min])) min = i;

     /* And add it */
                printf("Adding edge %d-%d\n", whoTo[min], min);
                inTree[min] = 1;
                total += d[min];
                //update MST //very important step!!!!
                v_last=whoTo[min];
                v_new=min;
                mst[v_last][v_new] = graph[v_last][v_new];
                printf("costs_mst[%d][%d]= %.0f\n",v_last,v_new,mst[v_last][v_new]);
                mst[v_new][v_last]=mst[v_last][v_new];//mst is symmetric array

                updateDistances(min);
        }
}//end Prim
//*************** PRINT 1D ARRAY *********************
```

```
void print_1d_array(float A[N]){
    for(int i=0; i<N;i++)
     printf("Er[%d]= %f\n",i,A[i]);
}
//*************  Compute Edge Costs  ***********************
void compute_edge_costs(float C[][N], float Et[][N], float Er[N]){
    for(int i=0;i<N;i++){
      for (int j=0;j<N;j++)
        if (i==j) C[i][j]=0.0;
        else C[i][j]=Et[i][j] + Er[j];
    }
}
//************ RECEIVE ENERGY OF each node ***************
void compute_Receive_energy(float Er[N]){
    for(int i=0;i<N;i++) Er[i]=k*Eelec;
}

//*************  PRINT DISTANCES  ***********************
void print_distances(float dist[N][N]){
   char answer;

   printf("distances...\n");
          for(int i=0; i<N; i++)
          {
                for(int j=0;j<N;j++)
                      printf("%6.1f",dist[i][j]);
                printf("\n");
          }
   printf("press any key...");
   scanf("%c",&answer);
}//end print_distances
//------------ PRINT ENERGY LEVELS ----------------
```

```
void print_energy_level(float energy[]){
    printf("\n");
   for(int i=0; i<N; i++) printf("%.2f  ",energy[i]);
}
//**********  UPDATE DISTANCES ***************
void updateDistances(int target) {
        int i;
        for (i = 0; i < N; ++i)
                if ((distances[target][i] != 0) && (d[i] > distances[target][i])) {
                        d[i] = distances[target][i];
                        whoTo[i] = target;
                }
}

void print_graph(float graph[][N]){
   for(int i=0; i<N; i++){
                for(int j=0;j<N;j++)
                    printf("%5.1f  ",graph[i][j]);
                printf("\n");
 }
 printf("\n");
}
//************** INIT MST *****************
void init_mst(float graph[][N]){
for(int i=0;i<N;i++)
                for(int j=0;j<N;j++)
                    graph[i][j]=0.0;
}
//we can store the graph in a file for easier simulation of different cases
//---------------- ARRAY TO FILE ---------------
void array_to_file(float A[][N],char* file_name){
```

```
    FILE *fp;

    fp=fopen(file_name,"w");
    if (!fp){
          printf("error opening file..exit!\n");
          getchar();
          exit(1);
          }
    for (int i=0;i<N;i++){
       for(int j=0;j<N;j++){
             fprintf(fp, "%f ",A[i][j]);
             }
       fprintf(fp,"\n");
    }
    fclose(fp);
}//end array to file
//------------------- READ ARRAY FROM FILE -----
void array_from_file(float A[][N], char* file_name){
    FILE *fp;
    char ch;
    fp=fopen(file_name,"r");
    if (!fp){
          printf("error opening file..exit!\n");
          getchar();
          exit(1);
          }
    for (int i=0;i<N;i++){
       for(int j=0;j<N;j++){
             fscanf(fp, "%f ",&A[i][j]);
             //printf("%d ",A[i][j]);getchar();
             }
    }
```

```
    fclose(fp);
}
```