ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ευφυής, Πολυπαραμετρική Διαχείριση Πληροφορίας σε Ασύρματα Δίκτυα Αισθητήρων

Αθανασούλης Γ. Μανούσος – Γαβριήλ

Αλαγιάννης Χ. Ιωάννης

Επιβλέπων: Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

Σεπτέμβριος 2005

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ευφυής, Πολυπαραμετρική Διαχείριση Πληροφορίας σε Ασύρματα Δίκτυα Αισθητήρων

Αθανασούλης Γ. Μανούσος – Γαβριήλ

Αλαγιάννης Χ. Ιωάννης

Επιβλέπων: Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

Σεπτέμβριος 2005

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ευφυής, Πολυπαραμετρική Διαχείριση Πληροφορίας σε Ασύρματα Δίκτυα Αισθητήτων

Αθανασούλης Γ. Μανούσος - Γαβριήλ

A.M.:1115200100002

Αλαγιάννης Χ. Ιωάννης

A.M.:1115200100011

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

Πρόλογος

Στην πτυχιακή εργασία αυτή αναπτύσσεται μια αρχιτεκτονική ευφυούς, πολυπαραμετρικής διαχείρισης πληροφορίας σε δίκτυα αισθητήρων. Το κείμενο συνοδεύεται από κώδικα που βρίσκεται στον αντίστοιχο ψηφιακό δίσκο αλλά και παρατίθεται σε παράρτημα.

Το παρόν κείμενο έχει οργανωθεί σε δύο μέρη. Το Μέρος Α: «Πλατφόρμες, περιβάλλοντα προγραμματισμού και προγενέστερη δουλειά», όπου συνοψίζεται η τρέχουσα κατάσταση στο πεδίο των ασυρμάτων δικτύων αισθητήρων και το Μέρος Β: «Αρχιτεκτονική διαχείρισης πληροφορίας με γνώση ενέργειας και σφάλματος» στο οποίο παρουσιάζεται η αρχιτεκτονική που προτείνεται σε αυτήν την εργασίας.

Το Μέρος Α, έχει τρία κεφάλαια. Στο κεφάλαιο 1, γίνεται μια εισαγωγή στη λογική των δικτύων αισθητήρων, παρουσιάζονται οι κύριοι τομείς εφαρμογών των δικτύων αυτών καθώς και τα κυριότερα προβλήματα που αντιμετωπίζονται σχεδιάζοντας δίκτυα αισθητήρων και πρωτόκολλα δρομολόγησης. Στο κεφάλαιο 2, γίνεται μια κατηγοριοποίηση και παρουσίαση των κύριων πρωτοκόλλων για δίκτυα αισθητήρων. Παρουσιάζονται συγκριτικά αποτελέσματα και οι βασικές ιδέες για το κάθε πρωτόκολλο. Το τελευταίο κεφάλαιο του πρώτου μέρους, το κεφάλαιο 3, περιλαμβάνει μια περιγραφή της κυριότερης πλατφόρμας λογισμικού που χρησιμοποιείται στα δίκτυα αισθητήρων, το TinyOS μαζί με τη nesC και τον προσομοιωτή TOSSIM καθώς και μια περιγραφή των κόμβων σε επίπεδο υλικού, προκειμένου να γίνει μια σύνδεση με το χρησιμοποιούμενο λογισμικό.

Το Μέρος Β χωρίζεται σε τέσσερα κεφάλαια. Στο κεφάλαιο 4 γίνεται μια θεωρητική παρουσίαση της προτεινόμενη αρχιτεκτονικής και των διαφόρων σημείων που παίζουν σημαντικό ρόλο στην απόδοση της. Στο επόμενο κεφάλαιο, αναλύεται η υλοποίηση που πραγματοποιήθηκε μαζί με τις βασικές σχεδιαστικές επιλογές. Στο κεφάλαιο 6 παρουσιάζονται τα αποτελέσματα των προσμοιώσεων της υλοποίησης και στο κεφάλαιο 7 συνοψίζονται κάποια συμπεράσματα μαζί με μερικες προτάσεις για μελλοντική δουλειά. Στα παραρτήματα που ακολουθούν παρατίθεται ένα γλωσσάρι με μερικές σημαντικές ορολογίες για τα δίκτυα αισθητήρων, μια σύνοψη για τα δεδομένα που χρησιμοποιούνται για τις προσεγγίσεις της ενεργειακής συμπεριφοράς, ένα παράδειγμα εκτέλεσης της εφαρμογής και ο κώδικας που γράφτηκε.

Τέλος, πρέπει να σημειωθεί ότι στα σημεία που παρατίθενται τα ονόματά μας, η παράθεση γίνεται με βάση την αλφαβητική τους σειρά.

Ευχαριστίες

Πριν ξεκινήσουμε την παρουσίαση της εργασίας θα ήταν μεγάλη παράλειψη εκ μέρους μας να μην ευχαριστήσουμε όλους όσους μας βοήθησαν και μας στήριξαν τον τελευταίο χρόνο κατά την εκπόνηση αυτής της πτυχιακής εργασίας.

Πιο συγκεκριμένα, θέλουμε να εκφράσουμε τις βαθύτατες ευχαριστίες μας στον επιβλέποντα καθηγητή κ. Χατζηευθυμιάδη, για τη συνεργασία, την καθοδήγηση, τις χρήσιμες συμβουλές και την ενθάρρυνση του, που μας βοήθησαν να πραγματοποίησουμε τα πρώτα μας βήματα στην έρευνα. Επιστέγασμα των παραπάνω δεν είναι μόνο η παρούσα εργασία αλλά και η – πρώτη μας – δημοσίευση σε επιστημονικό συνέδριο στον κλάδο των ασύρματων δικτύων αισθητήρων, η οποία χωρίς τη βοήθειά του δε θα είχε πραγματοποιηθεί.

Σε αυτό το σημείο θα θέλαμε να ευχαριστήσουμε την Ελένη Αθανασούλη για τη συμβολή της στη γραμματική και συντακτική επιμέλεια της εργασίας αυτής.

Ακόμη, θέλουμε να ευχαριστήσουμε τις οικογένειές μας, οι οποίες, κατά τη διάρκεια του τελευταίου έτους στάθηκαν στο πλευρό μας, μας στήριξαν και μας ανέχτηκαν στην προσπάθειά μας να ολοκληρώσουμε τις προπτυχιακές μας σπουδές παραμελώντας, ίσως, άλλες υποχρεώσεις μας.

Τέλος, θέλουμε να ευχαριστήσουμε τους φίλους μας που κατανόησαν τη σημασία της προσπάθειάς μας, μας στήριξαν, έδειξαν ανοχή στην απουσία μας και μας περίμεναν μέχρι να ολοκληρωθεί το έργο αυτό.

Σεπτέμβρης 2005,

Αθανασούλης Μανούσος-Γαβριήλ

Αλαγιάννης Ιωάννης

Περιεχόμενα

Μέρος Α: Πλατφόρμες, περιβάλλοντα προγραμματισμού και προγενέστερη δουλειά.

ΚΕΦΑΛΑΙΟ 1: *ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ*

1.1 Εισαγωγή σελ. 8
1.2 Τομείς εφαρμογών των δικτύων αισθητήρων σελ. 10
1.3 Παράγοντες που επηρεάζουν τη σχεδίαση δικτύων αισθητήρων και πρωτοκόλλων δρομολόγησης σελ. 14

ΚΕΦΑΛΑΙΟ 2: *ΠΡΩΤΟΚΟΛΛΑ ΑΣΥΡΜΑΤΗΣ ΕΠΙΚΟΙΝΩΝΙΑΣ ΣΕ Α.Δ.Α.*

2.1 Πρωτόκολλα βασισμένα στη δομή του δικτύου σελ. 22
2.1.1 Επίπεδη δρομολόγηση (Flat Routing) σελ. 22
2.1.2 Ιεραρχική δρομολόγηση (Hierarchical Routing) σελ. 28
2.1.3 Πρωτόκολλα βασισμένα στην τοποθεσία σελ. 32
2.2 Πρωτόκολλα βασισμένα στη λειτουργία του δικτύου σελ. 32
2.2.1 Πρωτόκολλα δρομολόγησης πολλαπλών μονοπατιών (Multipath routing protocols) σελ. 32
2.2.2 Δρομολόγηση βασισμένη σε επερώτηση (Query based routing) σελ. 33
2.2.3 Δρομολόγηση βασισμένη σε διαπραγμάτευση (Negotiation based routing protocols) σελ. 34
2.2.4 Δρομολόγηση βασισμένη στην ποιότητα υπηρεσιών (QoS-based routing) σελ. 34
2.2.5 Coherent and non-coherent protocols σελ. 34

ΚΕΦΑΛΑΙΟ 3: *ΠΛΑΤΦΟΡΜΕΣ ΑΣΥΡΜΑΤΩΝ ΔΙΚΤΥΩΝ ΑΙΣΘΗΤΗΡΩΝ*

3.1 Εισαγωγή σελ. 37
3.2 Αισθητήριοι Κόμβοι σε επίπεδο υλικού σελ. 38
3.2.1 Berkeley Motes σελ. 39
3.3 Πλατφόρμες λογισμικού σε επίπεδο κόμβου σελ. 40
3.3.1 TinyOS σελ. 41
3.3.2 nesC σελ. 50
3.4 Προσομοιωτές σε επίπεδο κόμβου σελ. 52
3.4.1 Γενικά σελ. 52
3.4.2 TOSSIM σελ. 55

Μέρος Β: Αρχιτεκτονική διαχείρισης πληροφορίας με γνώση ενέργειας και σφάλματος

ΚΕΦΑΛΑΙΟ 4: *ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ*

4.1 Εισαγωγή σελ. 60
4.2 Χρησιμότητα της αρχιτεκτονικής σελ. 60
4.3 Ιεραρχία δικτύου σελ. 61
4.4 Σχεδιασμός συνάρτησης αξιολόγησης (utility function) σελ. 65
4.5 Μηχανισμός αποστολής δεδομένων σελ. 67
4.6 Μηχανισμός αναπλήρωσης δεδομένων σελ. 68
4.7 Μοντέλο αποφόρτισης και επαναφόρτισης σελ. 68

ΚΕΦΑΛΑΙΟ 5: *ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΜΟΝΤΕΛΟΥ ΣΕ TinyOS, NesC, TOSSIM*

5.1 Components που χρησιμοποιούνται στην υλοποίηση σελ. 70
5.2 Σχόλια για το component Mcmfa σελ. 70
5.2.1 Σχεδιαστικές επιλογές σελ. 71
5.2.2 Κυριότερες συναρτήσεις σελ. 73
5.2.3 Υπόλοιπες συναρτήσεις σελ. 76

ΚΕΦΑΛΑΙΟ 6: *ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΕΩΝ*

6.1 Εισαγωγή σελ. 78
6.2 Παράδειγμα που περιλαμβάνει χρόνο σε κατάσταση αδράνειας σελ. 78
6.3 Παράδειγμα που περιλαμβάνει μόνο χρόνο σε κατάσταση λειτουργίας ή σε κατάσταση εξοικονόμησης ενέργειας σελ. 87

ΚΕΦΑΛΑΙΟ 7: ΣΥΜΠΕΡΑΣΜΑΤΑ - ΠΡΟΤΑΣΕΙΣ

7.1 Συμπεράσματα σελ. 97
7.2 Προτάσεις σελ. 97

Παράρτημα Α: Ορισμοί δικτύων αισθητήρων
Παράρτημα Β: Ενεργειακή συμπεριφορά κόμβων
Παράρτημα Γ: Παράδειγμα Εκτέλεσης
Παράρτημα Δ: Κώδικας εφαρμογής
Αναφορές

Μέρος Α

Πλατφόρμες, περιβάλλοντα προγραμματισμού και προγενέστερη δουλειά.

ΚΕΦΑΛΑΙΟ 1

ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

1.1 Εισαγωγή

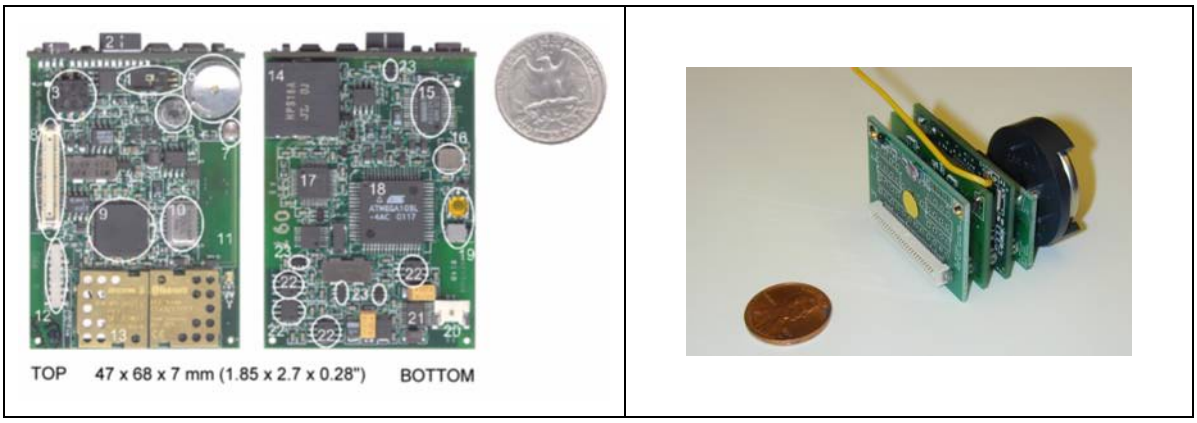
Η πρόσφατη πρόοδος στα μικροηλεκτρονικά μηχανικά συστήματα (MEMS - micro-electro-mechanical systems), στα ασύρματα δίκτυα και στις επικοινωνίες (wireless networks-communications) και στα ψηφιακά ηλεκτρονικά (ολοκληρωμένα και κατανεμημένα) αποτέλεσαν το έναυσμα για τη δημιουργία πολυχρηστικών, χαμηλού κόστους και κατανάλωσης, αισθητήριων κόμβων. Οι κόμβοι αυτοί, έχουν μικρό μέγεθος και τη δυνατότητα να επικοινωνούν σε μικρές αποστάσεις. Η ύπαρξη μικροσκοπικών κόμβων που μπορούν να πραγματοποιούν μετρήσεις, επεξεργασία δεδομένων και επικοινωνία, ώθησε στην ιδέα της δημιουργίας Ασύρματων Δικτύων Αισθητήρων (Α.Δ.Α.), τα οποία θα βασίζονται στη συνεργασία μεγάλου αριθμού αισθητήριων κόμβων. Τα δίκτυα αισθητήρων, αποτέλεσμα της σημαντικής προόδου που έχει συντελεστεί στους παραδοσιακούς αισθητήρες , σχεδιάζονται με τους δύο παρακάτω τρόπους:

- Οι αισθητήρες μπορούν να τοποθετηθούν μακρυά από το φαινόμενο που πρόκειται να μετρηθεί. Σε αυτή την προσέγγιση, είναι απαραίτητη η χρήση περίπλοκων τεχνικών ώστε να ξεχωρίσουμε τη μέτρηση από τον περιβάλλοντα θόρυβο.
- Χρησιμοποιούνται αισθητήρες οι οποίοι πραγματοποιούν μόνο μετρήσεις. Τόσο οι θέσεις των αισθητήρων όσο και η τοπολογία επικοινωνίας σχεδιάζονται προσεκτικά. Οι αισθητήρες μεταδίδουν τις μετρήσεις στους κεντρικούς κόμβους οι οποίοι πραγματοποιούν τους υπολογισμούς και συναθροίζουν τα δεδομένα

Ένα δίκτυο αισθητήρων συντίθεται από ένα μεγάλο αριθμό αισθητήριων κόμβων, οι οποίοι τοποθετούνται είτε μέσα είτε πολύ κοντά στο φαινόμενο που πρόκειται να παρατηρηθεί. Επιπλέον, οι θέσεις τους δεν είναι προσχεδιασμένες, ώστε να είναι δυνατή η τυχαία διασπορά τους π.χ. σε μη προσβάσιμες περιοχές. Έτσι, δημιουργείται η ανάγκη για πρωτόκολλα και αλγορίθμους οι οποίοι θα μπορούν να οργανώνουν δυναμικά το δίκτυο. Ένα επιπλέον μοναδικό χαρακτηριστικό των δικτύων αισθητήρων είναι η συνεργασία των αισθητήρων. Οι αισθητήρες διαθέτουν ενσωματωμένο επεξεργαστή και αντί να στέλνουν ροές δεδομένων στους κόμβους που είναι υπεύθυνοι για τη συγχώνευση, χρησιμοποιούν τον επεξεργαστή τους ώστε να φέρουν σε πέρας απλούς υπολογισμούς και να μεταδώσουν μόνο τα απαραίτητα και μερικώς επεξεργασμένα δεδομένα.

Όλα τα χαρακτηριστικά που αναφέρθηκαν παραπάνω επιβεβαιώνουν το μεγάλο εύρος από τομείς όπου μπορούν να εφαρμοστούν τα δίκτυα αισθητήρων όπως, η υγεία, η ασφάλεια και οι ένοπλες δυνάμεις. Για παράδειγμα, ένας γιατρός μπορεί να παρακολουθεί τα εργομετρικά δεδομένα ενός ασθενή του από μακρυά. Μια τέτοια περίπτωση είναι πιο βολική για τον ασθενή, αλλά επιπλέον επιτρέπει στο γιατρό να έχει περισσότερα στοιχεία για την παρούσα κατάσταση του ασθενή του. Ακόμα, τα δίκτυα αισθητήρων μπορούν να χρησιμοποιηθούν για τον εντοπισμό επικίνδυνων χημικών ουσιών στον αέρα και το νερό, ώστε να βοηθήσουν στην αναγνώριση του τύπου, της συγκέντρωσης και της τοποθεσίας αυτών των ουσιών. Στο μέλλον τα ασύρματα δίκτυα αισθητήρων θα αποτελούν αναπόσπαστο κομμάτι της ζωής μας όπως και οι υπολογιστές σήμερα. Αυτή η τεχνολογία θα αλλάξει επαναστατικά τον τρόπο που ζούμε, εργαζόμαστε και αλληλεπιδρούμε με το περιβάλλον. Στο εγγύς μέλλον, μικροί και

φθηνοί αισθητήρες θα βρίσκονται παντού, στους δρόμους, σε κτήρια, σε τοίχους, στα ρούχα, δημιουργώντας έναν ψηφιακό ιστό που θα παρέχει πληροφορία για ό,τι επιθυμούμε.



Εικόνα 1.1: Αισθητήριοι κόμβοι

Οι εφαρμογές των δικτύων αισθητήρων καθιστούν αναγκαία την ύπαρξη ασύρματων ad hoc τεχνικών δικτύωσης. Παρόλο, που έχουν προταθεί αρκετά πρωτόκολλα και αλγόριθμοι για τα παραδοσιακά ασύρματα ad hoc δίκτυα, αυτά δεν είναι ικανοποιητικά για τα μοναδικά χαρακτηριστικά και τις απαιτήσεις των δικτύων αισθητήρων. Εδώ, είναι σημαντικό να διαχωρίσουμε την έννοια των δικτύων αισθητήρων και των ad hoc δικτύων όπως φαίνεται παρακάτω:

- Ο αριθμός των αισθητήριων κόμβων σε ένα δίκτυο αισθητήρων είναι αρκετές τάξεις μεγέθους μεγαλύτερος από αυτόν των ad hoc δικτύων.
- Οι αισθητήριοι κόμβοι σχηματίζουν πυκνά δίκτυα.
- Οι αισθητήριοι κόμβοι είναι επιρρεπείς στα σφάλματα.
- Η τοπολογία ενός δικτύου αισθητήρων αλλάζει πολύ συχνά.
- Οι αισθητήριοι κόμβοι χρησιμοποιούν, συνήθως, καθολική επικοινωνία (broadcast), ενώ τα ad hoc δίκτυα σημείο-προς-σημείο (point-to-point) επικοινωνία.
- Οι αισθητήριοι κόμβοι έχουν περιορισμένη μνήμη, ενέργεια και υπολογιστική ισχύ.
- Οι αισθητήριοι κόμβοι μπορεί να μη διαθέτουν καθολική αναγνώριση (ID) εξαιτίας της μεγάλης σπατάλης και του μεγάλου αριθμού αισθητήρων.

Δεδομένου ότι μεγάλος αριθμός αισθητήριων κόμβων είναι πυκνά κατανεμημένοι στα δίκτυα αισθητήρων, οι γειτονικοί κόμβοι θα βρίσκονται πολύ κοντά και έτσι η multihop επικοινωνία αναμένεται να καταναλώνει λιγότερη ενέργεια (π.χ. για μεταδόσεις) από ότι η παραδοσιακή single hop επικοινωνία. Η Multihop επικοινωνία μπορεί, επιπλέον, να ξεπεράσει κάποιες από τις παρενέργειες της διάδοσης σήματος που εμφανίζονται σε επικοινωνίες μεγάλων αποστάσεων.

Ένας από τους πιο μεγάλους περιορισμούς στους αισθητήριους κόμβους είναι αυτός της μικρής κατανάλωσης ενέργειας. Οι αισθητήριοι κόμβοι διαθέτουν περιορισμένης ισχύος, και στην πλειοψηφία των περιπτώσεων, μη αντικαταστάσιμες πηγές ενέργειας. Αυτός είναι και ο λόγος όπου, ενώ τα παραδοσιακά δίκτυα αποσκοπούν στην παροχή υψηλής ποιότητας υπηρεσιών, τα πρωτόκολλα για δίκτυα αισθητήρων έχουν ως πρώτο

τους μέλημα την οικονομία στην ενέργεια. Τα δίκτυα αυτά πρέπει να έχουν μηχανισμούς που να επιτρέπουν στο χρήστη να επιλέξει στην παράταση της ζωής του δικτύου με κόστος μικρότερο throughput ή μεγαλύτερη καθυστέρηση μετάδοσης στο δίκτυο.

Οι αλλαγές που επιτελούνται στον τρόπο που αντιλαμβανόμαστε τα δίκτυα, δημιουργούν νέες προκλήσεις σχετικά με την επεξεργασία πληροφορίας. Απαιτούνται, πλέον, νέες υπολογιστικές αναπαραστάσεις, αλγόριθμοι, πρωτόκολλα, μέθοδοι σχεδίασης και εργαλεία για να υποστηρίζουν κατανεμημένη επεξεργασία σήματος, αποθήκευση και διαχείριση πληροφορίας, επικοινωνία και ανάπτυξη εφαρμογών.

Στη συνέχεια του κεφαλαίου θα παρουσιαστούν οι τομείς εφαρμογών των ασύρματων δικτύων αισθητήρων καθώς και οι παράγοντες που επηρεάζουν τη σχεδίαση δικτύων αισθητήρων και πρωτοκόλλων δρομολόγησης.

1.2 Τομείς εφαρμογών των δικτύων αισθητήρων

Ένα δίκτυο αισθητήρων μπορεί να αποτελείται από πολλούς, διαφορετικού τύπου αισθητήρες όπως σεισμικούς, θερμικούς, υπέρυθρους, οπτικούς, ακουστικούς κ.ά. με τους οποίους μπορεί να παρακολουθείται ένας μεγάλος αριθμός περιβαλλοντολογικών καταστάσεων όπως:

- Θερμοκρασία
- Υγρασία
- Φωτεινότητα
- Πίεση
- Επίπεδα θορύβου
- Κίνηση οχημάτων
- Την παρουσία ή όχι συγκεκριμένων αντικειμένων
- Χαρακτηριστικά, όπως ταχύτητα, κατεύθυνση, μέγεθος ενός αντικειμένου

Οι τομείς εφαρμογών των αισθητήριων κόμβων κατηγοριοποιούνται σε στρατιωτικούς, περιβαλλοντολογικούς, υγείας, οικιακούς και εμπορικούς. Όπως είναι λογικό, υπάρχουν και άλλες κατηγορίες όπως και διαφορετική κατηγοριοποίηση.

Στρατιωτικές Εφαρμογές

Τα ασύρματα δίκτυα αισθητήρων μπορούν να αποτελέσουν αναπόσπαστο κομμάτι σε στρατιωτικά συστημάτα ελέγχου, επικοινωνίας, υπολογισμών, παρακολούθησης, στόχευσης κ.ά. Τα δίκτυα αισθητήρων αποτελούνται από αναλώσιμους, χαμηλού κόστους αισθητήρες οπότε η καταστροφή μερικών από αυτούς από εχθρικές ενέργειες δεν επηρεάζει την εξέλιξη μιας στρατιωτικής επιχείρησης. Έτσι, τα δίκτυα αισθητήρων είναι σε θέση να προσφέρουν καλύτερη εποπτεία ενός πεδίου μάχης. Στη συνέχεια αναλύονται συγκεκριμένες στρατιωτικές εφαρμογές.

Έλεγχος φιλικών δυνάμεων, εξοπλισμών και πολεμοφοδίων: Η χρήση δικτύων αισθητήρων προσφέρει τη δυνατότητα για ενδελεχή παρατήρηση της θέσης των δυνάμεων, της κατάστασης του εξοπλισμού και της διαθεσιμότητας πολεμοφοδίων στο πεδίο της μάχης. Κάθε ομάδα στρατιωτών, κάθε όχημα και ο ζωτικός εξοπλισμός μπορεί να μεταφέρει ένα μικρό αισθητήρα με τον οποίο να αναφέρει κάθε στιγμή τη θέση αλλά και την κατάσταση του. Οι πληροφορίες από όλους τους αισθητήρες μιας

μονάδας συλλέγονται σε κόμβους “δεξαμενές” και αποστέλλονται στη διοίκηση του στρατεύματος.

Παρακολούθηση πεδίου μάχης: Με χρήση αισθητήριων κόμβων μπορούν να παρακολουθούνται σημαντικές περιοχές-εδάφη, διαδρομές, μονοπάτια για τις κινήσεις των εχθρικών δυνάμεων. Έτσι, όσο οι στρατιωτικές επιχειρήσεις βρίσκονται σε εξέλιξη τα εκάστοτε σχέδια μπορούν να τροποποιηθούν. Ακόμα, νέα δίκτυα αισθητήρων μπορούν να αναπτυχθούν στο πεδίο της μάχης.

Αναγνώριση εχθρικών δυνάμεων και περιοχών: Τα δίκτυα αισθητήρων μπορούν να αναπτυχθούν σε αποφασιστικής σημασίας περιοχές και να συγκεντρώσουν πολύτιμες πληροφορίες σε χρονικό διάστημα λίγων λεπτών χωρίς να προλάβουν να εμποδιστούν από τις εχθρικές δυνάμεις.

Στόχευση: Τα δίκτυα αισθητήρων μπορούν να ενσωματωθούν στα συστήματα κατεύθυνσης ευφυών οπλικών συστημάτων.

Αποτίμηση καταστροφής στη μάχη: Ένα δίκτυο αισθητήρων μπορεί να αναπτυχθεί στο πεδίο της μάχης πριν ή μετά από μια επίθεση ώστε να συλλέξει πληροφορίες για την έκταση των καταστροφών.

Αναγνώριση και εντοπισμός πυρηνικών, βιολογικών και χημικών επιθέσεων: Στις περιπτώσεις βιολογικών και χημικών επιθέσεων είναι πολύ σημαντικό να βρίσκεσαι στην καρδιά της επίθεσης, για γρήγορο και ακριβή εντοπισμό του μέσου που χρησιμοποιήθηκε. Δίκτυα αισθητήρων μπορούν να χρησιμοποιηθούν ως ένα σύστημα προειδοποίησης σε τέτοιου είδους επιθέσεις, ώστε να υπάρχει γρήγορη αντίδραση και να μειωθούν όσο το δυνατόν οι ανθρώπινες απώλειες. Ακόμα, ένα τέτοιο δίκτυο μπορεί να χρησιμοποιηθεί μετά από μια πυρηνική επίθεση για λεπτομερή καταγραφή του μεγέθους της καταστροφής χωρίς να χρειάζεται να εκτεθούν άνθρωποι στη ραδιενέργεια.

Περιβαλλοντολογικές Εφαρμογές

Μερικές περιβαλλοντολογικές εφαρμογές των δικτύων αισθητήρων περιλαμβάνουν την ανίχνευση των κινήσεων πτηνών, ζώων και εντόμων, την παρακολούθηση των περιβαλλοντολογικών συνθηκών που επηρεάζουν τις σοδειές, τον έλεγχο των συστημάτων άρδευσης, τον εντοπισμό ρυπογόνων ουσιών στο έδαφος, τη γη και τον αέρα, μετεωρολογικές και γεωλογικές έρευνες, τον έλεγχο της βιοποικιλότητας κ.ά. Στη συνέχεια, θα αναφερθούμε αναλυτικά σε κάποιες από αυτές δίνοντας περισσότερα στοιχεία.

Εντοπισμός πυρκαγιών στα δάση: Αισθητήρες μπορούν να τοποθετηθούν στα πιο στρατηγικά σημεία σχηματίζοντας ένα πυκνό δίκτυο που να καλύπτει όλο το δάσος και να μεταδώσουν χρησιμοποιώντας ραδιοσυχνότητες τα όρια της φωτιάς πριν λάβει ανεξέλεγκτες διαστάσεις. Οι αισθητήρες συνεργάζονται ώστε να παρέχουν κατανεμημένα δεδομένα και να μπορούν να ανταπεξέλθουν σε εμπόδια όπως δέντρα, βράχοι, που εμποδίζουν τις μετρήσεις ή τη μεταφορά δεδομένων. Οι αισθητήρες αυτοί μπορούν να εφοδιαστούν με συστήματα επαναφόρτισης, όπως ηλιακούς συλλέκτες, αφού για μεγάλο χρονικό διάστημα μπορεί να μείνουν χωρίς φροντίδα.

Χαρτογράφηση της βιοποικιλότητας ενός περιβάλλοντος: Η χαρτογράφηση της βιοποικιλότητας μιας περιοχής απαιτεί πολύπλοκες προσεγγίσεις οι οποίες θα είναι σε θέση να λαμβάνουν υπόψη τους τις συνεχείς αλλαγές του χώρου. Η αυτοματοποιημένη συλλογή δεδομένων από απόσταση έκανε δυνατή τη βαθύτερη φασματική, χωρική και χρονική ανάλυση με μειωμένο κόστος. Ακόμα, οι αισθητήρες έχουν την ικανότητα να

συνδέονται με το Internet επιτρέποντας τον έλεγχο και την παρακολούθηση της βιοποικιλότητας ενός περιβάλλοντος από απόσταση.

Αν και η χρήση εναέριων μεθόδων παρατήρησης (π.χ. δορυφόρων και αεροπλάνων) καθιστά εφικτή την παρακολούθηση των οργανισμών που κυριαρχούν σε κάποια περιοχή, κάτι τέτοιο, όμως, δεν είναι αποτελεσματικό σε περιπτώσεις όπου θέλουμε να εξετάσουμε μικρού εύρους βιοποικιλότητες, που είναι και η πιο συνηθισμένη περίπτωση. Για αυτό, έχει δημιουργηθεί η ανάγκη για ανάπτυξη ασύρματων δικτύων αισθητήρων στο έδαφος. Ένα παράδειγμα χαρτογράφησης της βιοποικιλότητας είναι αυτό που πραγματοποιείται στο James Reserve στην Νότια Καλιφόρνια. Εκεί, έχουν σχεδιαστεί τρία πλέγματα παρατήρησης όπου το καθένα αποτελείται από 25 έως 100 αισθητήρες και προσφέρουν πάγια εποπτεία.

Ανίχνευση πλημμυρών: Ένα παράδειγμα ανίχνευσης πλημμυρών είναι το σύστημα ALERT το οποίο έχει αναπτυχθεί στις Η.Π.Α. Το σύστημα αυτό αποτελείται από διαφορετικά είδη αισθητήρων όπως, μέτρησης του ύψους της βροχόπτωσης, του ύψους του νερού και του καιρού. Οι πληροφορίες που παρέχουν αυτοί οι αισθητήρες συλλέγονται στη βάση δεδομένων του κεντρικού συστήματος με κάποια προκαθορισμένη μέθοδο. Ερευνητικά προγράμματα όπως, το COUGAR Device Database Project του Cornell και το Dataspace Project του Rutgers, μελετούν κατανεμημένες προσεγγίσεις αλληλεπίδρασης με τους αισθητήρες στο πεδίο δράσης των αισθητήρων, ώστε να είναι δυνατή η λήψη προληπτικών μέτρων σε σύντομο αλλά και μακροπρόθεσμο διάστημα.

Γεωργία: Μερικά από τα οφέλη είναι η παρακολούθηση, σε πραγματικό χρόνο, του επιπέδου εμφάνισης φυτοφαρμάκων στο πόσιμο νερό, του επιπέδου διάβρωσης του εδαφους και του επιπέδου μόλυνσης του αέρα.

Εφαρμογές σε Τομείς της Υγείας

Σημαντικές εφαρμογές των δικτύων αισθητήρων στον τομέα της υγείας αφορούν τη σχεδίαση διεπαφών για τα άτομα με ειδικές ανάγκες. Επιπλέον, εφαρμογές είναι η ολοκληρωμένη παρακολούθηση ασθενών, η αρωγή στις ιατρικές γνωματεύσεις, ο έλεγχος των φαρμακευτικών αγωγών στα νοσοκομεία, η δυνατότητα τηλε-παρακολούθησης των βιομετρικών δεδομένων των ασθενών, ο εντοπισμός γιατρών και ασθενών στους χώρους του νοσοκομείου κ.ά.

Τηλεπαρακολούθηση των ανθρώπινων βιομετρικών δεδομένων: Τα βιομετρικά δεδομένα που συλλέγονται από τους αισθητήρες μπορούν να αποθηκευτούν για μεγάλο χρονικό διάστημα και να χρησιμοποιηθούν για ερευνητικούς σκοπούς. Ένα δίκτυο αισθητήρων μπορεί, ακόμα, να παρακολουθεί και να διακρίνει χαρακτηριστικά στη συμπεριφορά ηλικιωμένων ατόμων όπως στην περίπτωση κάποιου ατυχήματος. Οι αισθητήρες προσφέρουν στα υπο-παρακολούθηση άτομα μεγαλύτερη ελευθερία κινήσεων και στους γιατρούς τη δυνατότητα να εντοπίζουν γνωστά συμπτώματα ασθενειών σε πρώιμα στάδια εξέλιξης. Έτσι, οι ασθενείς απολαμβάνουν καλύτερη ποιότητα ζωής από ότι στις περιπτώσεις που παρακολουθούνται σε ιατρικά κέντρα. Ένα τέτοιο σύστημα παρακολούθησης ασθενών είναι το “Health Smart Home” που εκπονείται στο Faculty of Medicine στη Grenoble της Γαλλίας με σκοπό να εκτιμηθεί πόσο εφικτή είναι η χρήση ενός τέτοιου συστήματος.

Εντοπισμός και παρακολούθηση ασθενών και γιατρών στο χώρο του νοσοκομείου: Κάθε ασθενής έχει πάνω του μικρού σε μέγεθους και βάρους αισθητήρες, όπου στον καθένα έχει ανατεθεί ένα διαφορετικό έργο. Για παράδειγμα, ένας αισθητήρας μπορεί να ελέγχει τον καρδιακό ρυθμό ενώ ένας άλλος την πίεση του ασθενούς. Οι γιατροί μπορούν να

έχουν πάνω τους ένα αισθητήρα με τον οποίο να μπορούν να εντοπιστούν γρήγορα από τους συναδέλφους τους στο χώρο του νοσοκομείου.

Διαχείριση φαρμακευτικών αγωγών στα νοσοκομεία: Η χρήση συστημάτων μηχανογράφησης και επεξεργασίας των δεδομένων που συλλέγονται από αισθητήρες κατά τη χορήγηση φαρμακευτικής αγωγής μπορεί να μειώσει την πιθανότητα επιπλοκών λόγω κάποιας αλλεργίας ή λόγω χορήγησης λάθος φαρμάκου.

Οικιακές Εφαρμογές

Οικιακοί αυτοματισμοί: Η πρόοδος της τεχνολογίας έχει κάνει δυνατή την ενσωμάτωση έξυπνων αισθητήρων σε οικιακές συσκευές όπως η ηλεκτρική σκούπα, ο φούρνος μικροκυμάτων, ο καταψύκτης και το video. Οι αισθητήρες αλληλεπιδρούν και μπορούν να επικοινωνούν με ένα εξωτερικό δίκτυο μέσω Internet ή δορυφόρων. Κάτι τέτοιο, επιτρέπει στους χρήστες να χειρίζονται τις συσκευές εύκολα είτε από κοντά είτε από απόσταση.

Δημιουργία "έξυπνου" περιβάλλοντος: Η σχεδίαση ενός έξυπνου περιβάλλοντος έχει δύο όψεις, την ανθρωποκεντρική και εκείνη που περιστρέφεται γύρω από την τεχνολογία. Για την ανθρωποκεντρική πλευρά το περιβάλλον πρέπει να είναι σε θέση να προσαρμόζεται στις ανάγκες των χρηστών αλλά και στις ικανότητες αλληλεπίδρασης τους. Από την πλευρά της τεχνολογίας πρέπει να αναπτυχθούν νέες τεχνολογίες σε επίπεδο υλικού, λύσεις για τη διασύνδεση αλλά και ενδιάμεσες υπηρεσίες.

Στη συνέχεια, περιγράφεται ένα παράδειγμα για το πως οι αισθητήρες μπορούν να σχηματίσουν ένα έξυπνο περιβάλλον. Οι αισθητήρες μπορούν να είναι ενσωματωμένοι στα έπιπλα και στις οικιακές συσκευές και να επικοινωνούν μεταξύ τους και με τον server του δωματίου στο οποίο βρίσκονται. Ο server αυτός μπορεί να επικοινωνεί με server από άλλα δωμάτια και να ενημερώνεται για τις υπηρεσίες που είναι σε θέση αυτοί να προσφέρουν π.χ. εκτύπωση, σάρωση, τηλετυπία (fax). Οι server και οι αισθητήρες μπορούν να ενοποιηθούν και με τις υπάρχουσες συσκευές να δημιουργήσουν ένα σύστημα που να μπορεί να οργανώνεται, να ρυθμίζεται και προσαρμόζεται αυτόματα. Ένα άλλο παράδειγμα "έξυπνου" περιβάλλοντος είναι το "Residential Laboratory" του Georgia Institute of Technology. Τόσο η ικανότητα υπολογισμών όσο και μετρήσεων σε τέτοια συστήματα απαιτείται να είναι αξιόπιστη και συνεχής.

Άλλες Εμπορικές Εφαρμογές

Άλλες εμπορικές εφαρμογές είναι οι εξής: έλεγχος καταπόνησης υλικών, έλεγχος περιμέτρου εγκαταστάσεων, ποιοτικός έλεγχος προϊόντων, έλεγχος περιβάλλοντος σε γραφεία, έλεγχος και καθοδήγηση ρομπότ σε συστήματα αυτόματης παραγωγής, interactive παιχνίδια και οδηγοί μουσείων, παρακολούθηση περιοχών που έχουν υποστεί καταστροφή, διάγνωση προβλημάτων, εντοπισμός κλεφτών αυτοκινήτων, έλεγχος κίνησης αυτοκινήτων, κ.ά.

Έλεγχος περιβάλλοντος σε γραφεία κτηρίων: Η θέρμανση και η ψύξη στα περισσότερα κτήρια ελέγχεται από ένα κεντρικό σύστημα. Αυτός είναι και ο λόγος που η θερμοκρασία μπορεί να διαφέρει από τη μια άκρη του δωματίου στην άλλη αφού υπάρχει ένας μηχανισμός ελέγχου της θερμοκρασίας και η ροή του αέρα δεν είναι ομοιόμορφα κατανεμημένη. Σε τέτοιες περιπτώσεις μπορεί να χρησιμοποιηθεί ένα κατανεμημένο ασύρματο δίκτυο αισθητήρων όπου να ελέγχει τη θερμοκρασία και τη ροή του αέρα σε διαφορετικά σημεία του δωματίου. Έχει εκτιμηθεί πως ένα τέτοιο σύστημα είναι σε θέση να μειώσει την ενεργειακή κατανάλωση σε τέτοιο βαθμό ώστε να επιτευχθεί

εξοικονόμηση 55$ δις και μείωση κατά 35 εκατομμύρια κυβικούς τόνους των εκπομπών του άνθρακα.

Interactive μουσεία: Στο μέλλον, τα παιδιά θα είναι σε θέση να αλληλεπιδρούν με τα εκθέματα ενός μουσείου και έτσι να μαθαίνουν περισσότερα. Τα αντικείμενα θα μπορούν να αντιδρούν στο άγγιγμα και στην ομιλία και τα παιδιά να συμμετέχουν σε πραγματικό χρόνο. Ακόμα, ένα δίκτυο αισθητήρων θα μπορεί να προσφέρει ένα σύστημα εντοπισμού μέσα στο μουσείο.

Ανίχνευση και έλεγχος κλοπών αυτοκινήτων: Ένα δίκτυο αισθητήρων μπορεί να αναπτυχθεί ώστε να εντοπίζει και να αναγνωρίζει πιθανές απειλές και να τις αναφέρει στους ιδιοκτήτες μέσω Internet.

Έλεγχος απογραφών: Κάθε αντικείμενο σε μια αποθήκη μπορεί να έχει έναν αισθητήρα πάνω του. Έτσι, ο χρήστης είναι σε θέση να ξέρει κάθε στιγμή τη θέση ενός αντικειμένου αλλά και πόσα όμοια του υπάρχουν. Για κάθε εισαγωγή αντικειμένου αρκεί να συνδεθεί σε αυτό ο κατάλληλος αισθητήρας πριν μπει στην αποθήκη.

1.3 Παράγοντες που επηρεάζουν τη σχεδίαση δικτύων αισθητήρων και πρωτοκόλλων δρομολόγησης

Παρά τις αναρίθμητες εφαρμογές των Α.Δ.Α. αυτά τα δίκτυα έχουν αρκετούς περιορισμούς, όπως περιορισμένο ενεργειακό απόθεμα, περιορισμένη υπολογιστική ισχύ και περιορισμένο εύρος ζώνης των ασυρμάτων συνδέσεων που ενώνουν ασύρματους κόμβους. Ένας απο τους κυριότερους στόχους, όσον αφορά το σχεδιασμό Α.Δ.Α., είναι η επίτευξη της επικοινωνίας δεδομένων και η, παράλληλη, πρσπάθεια επιμήκυνσης του χρόνου λειτουργικότητας του δικτύου χωρίς να προκληθεί εκφυλισμός στη συνδεσιμότητα χρησιμοποιώντας αποτελεσματικές (aggressive) τεχνικές διαχείρισης της ενέργειας. Ο σχεδιασμός πρωτοκόλλων δρομολόγησης σε Α.Δ.Α. επηρεάζεται από ενδιαφέροντες (challenging) παράγοντες. Αυτές οι δυσκολίες πρέπει να ξεπεραστούν πριν επιτευχθεί αποδοτική επικοινωνία μέσω Α.Δ.Α. Στη συνέχεια συνοψίζονται κάποιες από τις δυσκολίες δρομολόγησης και κάποια από τα θέματα σχεδιασμού που επηρεάζουν τη διαδικασία δρομολόγησης σε Α.Δ.Α.

Κόστος Παραγωγής (Production Cost)

Δεδομένου ότι τα δίκτυα αισθητήρων αποτελούνται από ένα μεγάλο αριθμό αισθητήρων είναι λογικό το κόστος ενός απλού κόμβου να είναι σημαντικό για την εκτίμηση του συνολικού κόστους ενός δικτύου αισθητήρων. Έτσι, είναι πολύ σημαντικό για την επιλογή δημιουργίας ενός δικτύου αισθητήρων το κόστος κάθε μονάδας-κόμβου να είναι χαμηλό. Το κόστος ενός κόμβου πρέπει να είναι κάτω από 1$ ώστε η δημιουργία του δικτύου να είναι εφικτή. Ακόμα, ένας κόμβος πρέπει να έχει κάποιες επιπλέον μονάδες όπως αυτή για την μέτρηση κάποιου φαινομένου, μονάδες για την επεξεργασία δεδομένων, σύστημα εντοπισμού, σύστημα κίνησης και μονάδα παραγωγής ενέργειας ανάλογα με την εκάστοτε εφαρμογή. Από τα παραπάνω είναι εμφανές πως η κατασκευή ενός κόμβου με πολλές λειτουργικότητες και κόστος μικρότερο από 1$ είναι μια πρόκληση.

Περιορισμοί υλικού (Hardware Constraints)

Ένας αισθητήριος κόμβος αποτελείται από τέσσερα βασικά εξαρτήματα: την αισθητήρια μονάδα (που πραγματοποιεί τις μετρήσεις), τη μονάδα επεξεργασίας, τον πομποδέκτη, και τη μονάδα ενέργειας. Επιπλέον, ανάλογα με το είδος της εφαρμογής στην οποία

χρησιμοποιούνται μπορούν να είναι εφοδιασμένοι και με εξαρτήματα όπως σύστημα εντοπισμού, γεννήτρια ενέργειας και μηχανισμό κίνησης.

Οι αισθητήριες μονάδες συνήθως αποτελούνται από δύο υπομονάδες, τους αισθητήρες και τους μετατροπείς από αναλογικό σε ψηφιακό (Analog to Digital Converters – ADCs). Το αναλογικό σήμα το οποίο προέρχεται από τον αισθητήρα, ανάλογα με το φαινόμενο το οποίο παρατηρείται, μετατρέπεται σε ψηφιακό σήμα με χρήση του ADC και στη συνέχεια προωθείται στη μονάδα επεξεργασίας. Η μονάδα επεξεργασίας, η οποία συνήθως διαθέτει μικρές μονάδες αποθήκευσης, αναλαμβάνει τις διαδικασίες με τις οποίες ο κόμβος συνεργάζεται με άλλους κόμβους για να φέρει σε πέρας το έργο που του έχει ανατεθεί. Ο πομποδέκτης είναι αυτός που συνδέει τον κόμβο με το υπόλοιπο δίκτυο. Τέλος, ένα από τα πιο σημαντικά εξαρτήματα ενός κόμβου είναι η μονάδα ενέργειας. Πολλές φορές οι μονάδες ενέργειας ενισχύονται με βοηθητικές μονάδες όπως ηλιακούς συσσωρευτές.

Ανάλογα με την εφαρμογή μπορεί να χρησιμοποιηθούν και άλλες υπομονάδες. Για παράδειγμα, οι μετρήσεις και οι περισσότερες τεχνικές δρομολόγησης για δίκτυα αισθητήρων χρειάζονται τον ακριβή προσδιορισμό της θέσης. Γι' αυτό, και οι κόμβοι εφοδιάζονται με συστήματα εντοπισμού. Ακόμα, κάποιος μηχανισμός κίνησης μπορεί να χρειάζεται ώστε να μετακινηθεί ο αισθητήρας για να εκπληρώσει το έργο που του έχει ανατεθεί.

Όλες οι υπομονάδες που αναφέραμε παραπάνω είναι πιθανόν να χρειάζεται να καταλάβουν χώρο όσο ένα σπιρτόκουτο ή και μικρότερο π.χ. ένα κυβικό εκατοστό μέγεθος το οποίο πρέπει να αντιστοιχεί σε τέτοιο βάρος ώστε να μην παρασύρεται από τον αέρα. Εκτός από το μέγεθος υπάρχουν και κάποιοι άλλοι αυστηροί περιορισμοί που πρέπει να λάβουμε υπόψη μας. Οι κόμβοι πρέπει:

- Να καταναλώνουν πολύ λίγη ενέργεια
- Να λειτουργούν σε υψηλή ογκομετρική πυκνότητα
- Να έχουν χαμηλό κόστος παραγωγής
- Να είναι αυτόνομοι και να μπορούν να λειτουργούν χωρίς επίβλεψη
- Να προσαρμόζονται στο περιβάλλον

Δεδομένου ότι οι κόμβοι συχνά δεν είναι προσπελάσιμοι, η διάρκεια ζωής ενός δικτύου αισθητήρων εξαρτάται άμεσα από τους ενεργειακούς πόρους που διαθέτουν οι κόμβοι. Η ενέργεια, όμως, είναι ένας "σπάνιος" πόρος εξαιτίας των περιορισμών στο μέγεθος του κόμβου. Για παράδειγμα, η συνολική ενέργεια που μπορεί να αποθηκευτεί σε ένα *smart dust mote* είναι 1J. Για ασύρματους ολοκληρωμένους δικτυακούς αισθητήρες (Wireless Integrated Network Sensors – WINS), το συνολικό κατά μέσο όρο ρεύμα που προμηθεύει το σύστημα πρέπει να είναι λιγότερο από 1μA ώστε να έχουμε μεγάλη περίοδο λειτουργίας. Οι κόμβοι τύπου WINS τροφοδοτούνται από μπαταρίες λιθίου (coin cell μπαταρίες με διαστάσεις 2.5 cm διάμετρο και 1 cm πάχος). Η διάρκεια ζωής ενός κόμβου μπορεί να αυξηθεί αν εκμεταλλευτούμε την ενέργεια που παρέχει το περιβάλλον χρησιμοποιώντας για παράδειγμα ηλιακούς συλλέκτες.

Ο πομποδέκτης του κόμβου μπορεί να είναι είτε παθητική είτε ενεργητική οπτική συσκευή όπως συμβαίνει στους *smart dust motes* είτε συσκευή που χρησιμοποιεί ραδιοσυχνότητες (Radio Frequency – RF). Οι επικοινωνίες τύπου RF χρειάζονται διαμόρφωση, εύρος λειτουργίας, φίλτρα, αποδιαμόρφωση και πολυπλεγμένα κυκλώματα με αποτέλεσμα να είναι πολύπλοκες αλλά και να κοστίζουν πολύ. Επιπλέον, η απώλεια για διαβιβαζόμενα σήματα σε μονοπάτια ανάμεσα σε δύο κόμβους μπορεί να είναι έως και τετάρτου βαθμού εκθετική σε σχέση με την απόσταση των κόμβων αφού οι κεραίες τους βρίσκονται πολύ κοντά στο έδαφος. Παρ' όλα αυτά, οι επικοινωνίες τύπου

RF προτιμούνται στα περισσότερα ερευνητικά εγχειρήματα που αφορούν δίκτυα αισθητήρων γιατί τα πακέτα που διαβιβάζονται είναι μικρά, η συχνότητα ροής δεδομένων είναι μικρή (γενικά μικρότερη από 1 Hz) και η δυνατότητα για επαναχρησιμοποίηση της συχνότητας είναι υψηλή λόγω των μικρών αποστάσεων επικοινωνίας. Τα παραπάνω χαρακτηριστικά κάνουν, ακόμα, δυνατή τη χρήση ασύρματων ηλεκτρονικών με μικρό κύκλο εργασίας. Η σχεδίαση ενεργειακά αποδοτικών και με χαμηλό κύκλο εργασίας ασύρματων κυκλωμάτων είναι τεχνικά ακόμα μια πρόκληση. Οι εμπορικές τεχνολογίες που χρησιμοποιούνται σήμερα όπως τα Bluetooth δεν είναι αρκετά αποδοτικές για να χρησιμοποιηθούν σε δίκτυα αισθητήρων γιατί η ενεργοποίηση και απενεργοποίηση τους (on-off) καταναλώνει πολύ ενέργεια.

Παρ'όλο που υψηλής υπολογιστικής ισχύος επεξεργαστές είναι διαθέσιμοι σε όλο και μικρότερο μέγεθος, οι μονάδες επεξεργασίας και μνήμης για αισθητήριους κόμβους είναι ακόμα περιορισμένοι πόροι. Για παράδειγμα, η μονάδα επεξεργασίας του πρωτότυπου *smart dust mote* είναι ένας 4 MHz Atmel AVR 8535 μικρό controller με 8 KB flash μνήμη εντολών, 512 bytes RAM και 512 bytes ηλεκτρικής διαγράψιμης προγραμματιζόμενης μνήμης μόνο για ανάγνωση (Electrically Erasable Programmable Read-only Memory - EEPROM). Το λειτουργικό σύστημα TinyOS χρησιμοποιήθηκε σε αυτό τον επεξεργαστή, ο οποίος έχει 3500 bytes χώρο για κώδικα του λειτουργικού συστήματος και 4500 bytes διαθέσιμο χώρο για κώδικα. Η μονάδα επεξεργασίας ενός άλλου πρωτότυπου κόμβου, που ονομάζεται μAMPS ασύρματος αισθητήριος κόμβος, έχει ένα 59-206 MHz SA-110 μικροεπεξεργαστή. Ένα πολυνηματικό μ-OS λειτουργικό σύστημα εκτελείται στους μAMPS ασύρματους αισθητήριους κόμβους.

Οι περισσότερες από τις μετρήσεις που πραγματοποιούνται με αισθητήρες χρειάζονται και την πληροφορία που αφορά τη θέση της μέτρησης. Δεδομένου ότι οι κόμβοι αναπτύσσονται τυχαία και τρέχουν χωρίς κάποια επιτήρηση, είναι ανάγκη να συνεργάζονται με ένα σύστημα εντοπισμού. Σύστημα εντοπισμού χρειάζεται από πολλά πρωτόκολλα δρομολόγησης που χρησιμοποιούνται στα ασύρματα δίκτυα αισθητήρων. Στο μέλλον, προβλέπεται πως κάθε κόμβος θα διαθέτει μονάδα καθολικού συστήματος εντοπισμού (Global Positioning System – GPS) το οποίο θα έχει το λιγότερο 5 μέτρα ακρίβεια. Πάντως, εκφράζονται αμφιβολίες αν θα είναι εφαρμόσιμο να διαθέτουν GPS όλοι οι κόμβοι και, έτσι, έχει προταθεί μια διαφορετική προσέγγιση όπου ένας περιορισμένος αριθμός από κόμβους θα διαθέτει GPS και θα βοηθά και τους υπόλοιπους να προσδιορίζουν τη θέση τους.

Περιβάλλον (Environment)

Οι κόμβοι αναπτύσσονται πυκνά είτε πολύ κοντά μεταξύ τους, είτε μέσα στο φαινόμενο, το οποίο παρατηρούν. Συνήθως λειτουργούν χωρίς επίβλεψη σε απομακρυσμένες γεωγραφικές περιοχές. Μπορεί να είναι εγκατεστημένοι και να λειτουργούν:

- Σε πολυάσχολες διασταυρώσεις
- Στο εσωτερικό ενός μεγάλου μηχανισμού
- Στον πυθμένα κάποιου ωκεανού
- Στο εσωτερικό ενός ανεμοστρόβιλου
- Στην επιφάνεια ενός ωκεανού κατα τη διάρκεια καταιγίδας
- Σε κάποια βιολογικά ή χημικά μολυσμένη περιοχή
- Στο πεδίο της μάχης ανάμεσα στις εχθρικές γραμμές
- Σε κάποιο σπίτι ή σε κάποιο μεγάλο κτήριο

- Σε μια μεγάλη αποθήκη
- Προσκολλημένοι σε κάποιο ζώο
- Προσκολλημένοι σε κάποιο κινούμενο όχημα
- Σε κάποιο σωλήνα ή σε κάποιο ποτάμι παρασυρόμενοι από το ρεύμα.

Η παραπάνω λίστα μας προδιαθέτει για τις συνθήκες που πρόκειται να συναντήσουν οι κόμβοι στο περιβάλλον εργασίας τους. Λειτουργούν σε υψηλή πίεση στο βυθό ενός πελάγους, σε σκληρές συνθήκες όπως σε χαλάσματα ή στο πεδίο της μάχης, σε υψηλή ζέστη ή κρύο όπως μέσα στη μηχανή ενός αεροσκάφους ή στις αρκτικές περιοχές και σε πολύ θορυβώδη περιβάλλοντα.

Παράταξη κόμβων (Node deployment)

Η παράταξη κόμβων σε Α.Δ.Α εξαρτάται από την εκάστοτε εφαρμογή και επηρεάζει την απόδοση του πρωτοκόλλου δρομολόγησης. Η παράταξη μπορεί να είναι είτε ντετερμινιστική είτε τυχαία. Εάν είναι ντετερμινιστική οι αισθητήρες τοποθετούνται από το χρήστη ή με χρήση ρομπότ και τα δεδομένα δρομολογούνται μέσω προαποφασισμένων μονοπατιών. Όμως, εάν η παράταξη είναι τυχαία (οι αισθητήρες έχουν αναπτυχθεί π.χ. ρίχνοντας τους από αεροπλάνο, με χρήση κάποιου βλήματος ή πυραύλου, με εκτόξευση από κάποιο καταπέλτη), οι αισθητήρες διασκορπίζονται (scattered) τυχαία δημιουργώντας μια υποδομή με έναν ad hoc τρόπο. Εάν η κατανομή που προέκυψε δεν είναι ομοιόμορφη τότε το optimal clustering γίνεται απαραίτητο για να επιτευχθεί συνδεσιμότητα και να είναι εφικτή η αποδοτική ως προς την ενέργεια λειτουργία του δικτύου. Επικοινωνία μεταξύ κόμβων γίνεται, υπό κανονικές συνθήκες, με μικρή εμβέλεια μετάδοσης λόγω περιορισμών ενέργειας και εύρους ζώνης. Άρα, είναι πιθανό μια διαδρομή πάνω στο δίκτυο να αποτελείται από πολλαπλά ασύρματα “βήματα” (hops). Σκοπός της παράταξης κόμβων σε πρώτο στάδιο πρέπει να είναι η μείωση όσο το δυνατόν του κόστους εγκατάστασης των αισθητήρων, η εξάλειψη της ανάγκης για οποιαδήποτε από πριν οργάνωση και σχεδιασμό, η αύξηση της ευελιξίας της διάταξης, η προώθηση της αυτο-οργάνωσης και η αύξηση της ανεκτικότητας σε σφάλματα. Μετά την παράταξή τους οι αισθητήρες συμπεριφέρονται ανάλογα με τις συνθήκες που συναντούν αλλά και το είδος της εργασίας που πρέπει να διεκπεραιώσουν. Ακόμα, πρέπει να έχουμε υπόψην πως είναι πιθανό να “χάσουμε” αισθητήρες από τη διάταξη λόγω καταστροφής ή μείωσης των διαθέσιμων πόρων τους.

Κατανάλωση ενέργειας χωρίς να χαθεί η ακρίβεια

Οι αισθητήρες μπορούν να καταναλώσουν το περιορισμένο ενεργειακό τους απόθεμα επιτελώντας υπολογισμούς και μεταδόσεις πληροφορίας σε ένα ασύρματο περιβάλλον. Η κύρια αποστολή τους είναι ο εντοπισμός γεγονότων, η γρήγορη τοπική επεξεργασία των δεδομένων και στη συνέχεια η μετάδοση αυτών (*sensing, data processing και communication*). Η ενέργεια που απαιτείται για κάποια μέτρηση εξαρτάται από το είδος της εφαρμογής αλλά και από τη φύση του δικτύου (στατικό ή κινητό). Σποραδικές μετρήσεις μπορεί να καταναλώνουν λιγότερη ενέργεια από τη σταθερή παρακολούθηση φαινομένων. Ακόμα, η πολυπλοκότητα εντοπισμού φαινομένων μπορεί να οδηγήσει σε μεγαλύτερη ενεργειακή δαπάνη. Συνεπώς, μέθοδοι εκτέλεσης υπολογισμών και επικοινωνίας που εξοικονομούν ενέργεια είναι θεμελιώδεις. Ο χρόνος λειτουργικότητας ενός αισθητήρα εξαρτάται σε μεγάλο βαθμό από το χρόνο ζωής της μπαταρίας. Σε ένα multihop Α.Δ.Α. κάθε κόμβος παίζει ένα διπλό ρόλο, ως αποστολέας δεδομένων αλλά και δρομολογητής δεδομένων. Η κακή λειτουργία μερικών αισθητήρων λόγω ανεπάρκειας ενέργειας μπορεί να προκαλέσει σημαντικές αλλαγές στην τοπολογία του

Α.Δ.Α. και να δημιουργήσει την ανάγκη επαναδρομολόγησης πακέτων και επανοργάνωσης του δικτύου.

Μοντέλο διάδοσης δεδομένων

Η καταμέτρηση και η διάδοση δεδομένων σε ένα Α.Δ.Α. εξαρτώνται από την εφαρμογή και από την εξάρτηση της διάδοσης των δεδομένων από το χρόνο. Η διάδοση των δεδομένων μπορεί να κατηγοριοποιηθεί ως βασισμένη στο χρόνο (συνεχής), βασισμένη σε γεγονότα, βασισμένη σε ερώτηση και υβριδική. Το βασισμένο στο χρόνο μοντέλο είναι κατάλληλο για εφαρμογές που απαιτούν περιοδική παρακολούθηση των δεδομένων. Σε μια τέτοια εφαρμογή οι κόμβοι θα ανοίγουν περιοδικά τους αισθητήρες τους, θα μετράνε το αντίστοιχο μέγεθος και θα μεταδίδουν τα δεδομένα ανά σταθερά χρονικά διαστήματα. Στα μοντέλα διάδοσης που είναι βασισμένα σε γεγονότα ή σε ερώτηση, οι κόμβοι αντιδρούν άμεσα σε απότομες και δραστικές αλλαγές της τιμής του υπο-μέτρηση μεγέθους λόγω της εμφάνισης ενός αντίστοιχου γεγονότος ή ερωτήματος παραχθέντος από τον κόμβο-βάση. Συνεπώς, αυτά τα μοντέλα είναι κατάλληλα για εφαρμογές που ο χρόνος είναι ένα κρίσιμο μέγεθος. Επίσης, ένας συνδυασμός των ανωτέρω μοντέλων είναι εφικτός. Το πρωτόκολλο δρομολόγησης επηρεάζεται πολύ από το μοντέλο διάδοσης δεδομένων όσον αφορά την κατανάλωση ενέργειας και τη σταθερότητα της διαδρομής.

Ανομοιογένεια κόμβων/συνδέσεων

Μια υπόθεση που έχει γίνει σε πολλές μελέτες είναι πως οι κόμβοι είναι ομοιογενείς, δηλαδή, έχουν ίδιες δυνατότητες όσον αφορά τους υπολογισμούς, την επικοινωνία και την ενέργεια. Όμως, ανάλογα με την εφαρμογή ένας κόμβος μπορεί να έχει διαφορετικό ρόλο ή δυνατότητες. Η ύπαρξη ετερογενών συνόλων κόμβων εγείρει πολλά τεχνικά ερωτήματα σχετικά με τη δρομολόγηση των δεδομένων. Για παράδειγμα, κάποιες εφαρμογές ενδέχεται να απαιτούν ένα συνδυασμό από διαφορετικούς κόμβους για να παρακολουθεί τη θερμοκρασία, την πίεση και την υγρασία του περιβάλλοντος, για να εντοπίζει κίνηση μέσω ηχητικών σημάτων και να αποθηκεύει φωτογραφικά και μαγνητοσκοπικά ίχνη κινούμενων αντικειμένων. Οι προαναφερθέντες ειδικοί κόμβοι μπορεί να παραταχθούν ατομικά ή να συγκεντρώνονται όλες οι διαφορετικές αυτές λειτουργίες σε κάθε έναν από αυτούς. Επίσης, μέτρηση και διάδοση δεδομένων μπορεί να πραγματοποιηθεί από αυτούς τους κόμβους, αλλά σε διαφορετικούς ρυθμούς, που υπόκεινται στους περιορισμούς της ποιότητας υπηρεσιών που επιβάλλονται και ενδέχεται να ακολουθήσουν πολλαπλά μοντέλα διάδοσης δεδομένων. Για παράδειγμα, τα ιεραρχικά πρωτόκολλα προσδιορίζουν έναν κόμβο επικεφαλής διαφορετικό από τους κανονικούς κόμβους. Οι κόμβοι επικεφαλής μπορεί να επιλεχθούν από τους παρατεταγμένους κόμβους ή μπορεί να είναι πιο ισχυροί από τους υπόλοιπους ως προς το ενεργειακό απόθεμα, το εύρος ζώνης λειτουργίας και την τοπική μνήμη. Συνεπώς, η ευθύνη της μετάδοσης στον κόμβο-βάση μεταφέρεται στους κόμβους-επικεφαλής.

Ανοχή σφάλματος

Η λειτουργία κάποιων κόμβων μπορεί να τερματιστεί ή να εμποδιστεί λόγω έλλειψης ενέργειας, ζημιάς στο υλικό ή παρεμβολής από το περιβάλλον. Η αστοχία κόμβων δεν πρέπει, όσο το δυνατόν, να επηρεάσει τη συνολική λειτουργία του δικτύου. Η ανοχή σφάλματος είναι η ικανότητα του Α.Δ.Α να διατηρεί τις λειτουργικότητες του, χωρίς καμιά διακοπή, σε περιπτώσεις αποτυχίας κάποιων κόμβων. Εάν αποτύχει η λειτουργία

πολλών κόμβων τότε τα πρωτόκολλα MAC (Media Access Control) και δρομολόγησης πρέπει να παρέχουν δυνατότητα σχηματισμού νέων συνδέσεων και μονοπατιών προς τον κόμβο-βάση που συλλέγει τα δεδομένα. Αυτό μπορεί να απαιτεί δραστική επαναπροσαρμογή των ρυθμών μετάδοσης χρησιμοποιώντας τις υπάρχουσες συνδέσεις με σκοπό τη μείωση της κατανάλωσης ενέργειας ή επαναδρομολόγηση πακέτων από περιοχές του δικτύου που η ενεργειακή κατάσταση των κόμβων είναι καλύτερη. Επομένως, σε ένα δίκτυο με περιορισμένη ανοχή σφάλματος χρειάζεται πλεονασμός πληροφορίας σε πολλά επίπεδα. Τόσο τα πρωτόκολλα όσο και οι αλγόριθμοι σχεδιάζονται ώστε να διεκπεραιώνουν το επίπεδο ανοχής σφάλματος που απαιτείται από το δίκτυο δίνοντας, πάντα, βαρύτητα στην εφαρμογή από την οποία θα χρησιμοποιηθούν. Για παράδειγμα, κόμβοι που αναπτύσσονται σε κάποιο σπίτι για να μετρούν τα επίπεδα υγρασίας και θερμοκρασίας έχουν μικρές απαιτήσεις ως προς την ανοχή σφάλματος, αφού ένα τέτοιο δίκτυο είναι δύσκολο να καταστραφεί ή να δεχθεί παρεμβολές από θόρυβο του περιβάλλοντος. Από την άλλη, όμως, αν κόμβοι αναπτυχθούν σε κάποιο πεδίο μάχης (π.χ. για την παρακολούθηση εχθρικών δυνάμεων) τότε οι απαιτήσεις ως προς την ανοχή σφάλματος είναι υψηλές γιατί τα δεδομένα είναι πολύ σημαντικά αλλά και οι πιθανότητες να καταστραφούν κόμβοι από εχθρικές ενέργειες είναι μεγάλες.

Επεκτασιμότητα

Ο αριθμός των κόμβων που αποτελούν ένα ασύρματο δίκτυο αισθητήρων μπορεί να είναι τάξη μεγέθους εκατοντάδες, χιλιάδες ή και περισσότερο. Κάθε πρωτόκολλο δρομολόγησης πρέπει να μπορεί να είναι λειτουργικό με τόσο μεγάλους αριθμούς από κόμβους. Επιπλέον τα πρωτόκολλα αυτά πρέπει να είναι αρκετά επεκτάσιμα ώστε να μπορούν να ανταποκριθούν σε εξωτερικά γεγονότα. Η πλειοψηφία των κόμβων πρέπει να παραμένει σε sleep mode έως ότου ένα γεγονός λάβει χώρα και τα δεδομένα των υπολοίπων κόμβων να παρέχουν μια αδρή ποιότητα. Το πλήθος των κόμβων σε μια περιοχή μπορεί να δείξει την πυκνότητά τους. Η πυκνότητα κόμβων εξαρτάται από την εφαρμογή στην οποία θα χρησιμοποιηθούν. Για παράδειγμα, σε εφαρμογές που αφορούν την παρακολούθηση της λειτουργίας κάποιας μηχανής η πυκνότητα κόμβων είναι περίπου 300 αισθητήριοι κόμβοι σε μια περιοχή 5 x 5 m^2 και η πυκνότητα σε μια εφαρμογή για εντοπισμό οχημάτων περίπου 10 κόμβοι ανά περιοχή.

Δυναμική δικτύου

Οι περισσότερες από τις αρχιτεκτονικές δικτύων υποθέτουν ότι οι αισθητήριοι κόμβοι είναι στατικοί. Πάρα ταύτα η δυνατότητα κίνησης των κόμβων-βάσεων αλλά και των υπόλοιπων κόμβων είναι απαραίτητη για πολλές εφαρμογές. Μηνύματα δρομολόγησης από και προς κινούμενους κόμβους είναι ένα πολύ ενδιαφέρον θέμα, αφού η ευστάθεια του μονοπατιού γίνεται ένας πολύ σημαντικός παράγοντας πέραν του ενεργειακού αποθέματος, του εύρους ζώνης κτλ. Επιπλέον το υπό παρατήρηση φαινόμενο μπορεί να είναι είτε δυναμικό είτε στατικό ανάλογα με την εφαρμογή, λ.χ. είναι δυναμικό σε μια εφαρμογή εύρεσης και παρακολούθησης στόχου, ενώ είναι στατικό σε μια εφαρμογή παρακολούθησης δασών για έγκαιρη ενημέρωση σε περίπτωση πυρκαϊάς. Η παρακολούθηση στατικών γεγονότων επιτρέπει στο δίκτυο μια αιτιοκρατική λειτουργία απλά παράγοντας μηνύματα και κίνηση όταν χρειάζεται να μεταδοθεί πληροφορία. Δυναμικά γεγονότα στις περισσότερες εφαρμογές απαιτούν περιοδική διάδοση και συνεχή παραγωγή σημαντικής κίνησης η οποία να δρομολογείται προς τον κόμβο-βάση.

Μέσον μετάδοσης

Σε ένα multihop δίκτυο αισθητήρων, οι κόμβοι που πραγματοποιούν επικοινωνία συνδέονται με ένα ασύρματο μέσο. Η εγκαθίδρυση αυτών των συνδέσεων μπορεί να πραγματοποιηθεί με χρήση ραδιοπομπών, υπέρυθρων ή άλλων οπτικών μέσων. Για να είναι λειτουργικά ανά την υφήλιο τα Α.Δ.Α πρέπει το μέσο μετάδοσης να είναι διαθέσιμο παντού. Μια επιλογή για τις ασύρματες συνδέσεις είναι να χρησιμοποιήσουν τη βιομηχανική, επιστημονική και ιατρική (Industrial, Scientific and Medical - ISM) ζώνη συχνοτήτων, η οποία είναι ελεύθερη προς χρήση στις περισσότερες χώρες. Τα παραδοσιακά προβλήματα που σχετίζονται με ένα ασύρματο κανάλι (όπως fading και υψηλό ρυθμό σφαλμάτων) ενδέχεται να επηρεάσουν τη λειτουργία του δικτύου αισθητήρων. Σε γενικές γραμμές, το απαιτούμενο εύρος ζώνης για δεδομένα που διακινούνται σε ένα τέτοιο δίκτυο θα είναι μικρό, της τάξης των 1-100 kb/s. Ο σχεδιασμός του MAC σχετίζεται με το μέσον μετάδοσης. Μια προσέγγιση του σχεδιασμού του MAC για δίκτυα αισθητήρων είναι η χρήση πρωτοκόλλων βασισμένων σε TDMA (Time Division Multiple Access), τα οποία εξοικονομούν περισσότερη ενέργεια από τα πρωτόκολλα που είναι βασισμένα σε ανταγωνισμό (contention based) όπως το CSMA (π.χ. IEEE 802.11). Η τεχνολογία Bluetooth μπορεί επίσης να χρησιμοποιηθεί.

Συνδεσιμότητα

Η υψηλή πυκνότητα κόμβων σε δίκτυα αισθητήρων αποκλείει την πιθανότητα κάποιος κόμβος να βρεθεί εξ ολοκλήρου απομονωμένος από τους υπόλοιπους. Άρα, οι κόμβοι αναμένεται να έχουν υψηλή συνδεσιμότητα. Αυτό, όμως, δεν αποτρέπει τις μεταβολές της τοπολογίας του δικτύου καθώς και τη συρρίκνωση του δικτύου που οφείλεται σε αστοχίες κόμβων. Επι πλέον, η συνδεσιμότητα εξαρτάται από την, πιθανώς τυχαία, κατονομή των κόμβων.

Κάλυψη

Στα Α.Δ.Α., κάθε κόμβος μπορεί να αποτυπώσει μόνο μια *όψη* από το περιβάλλον. Μια συγκεκριμένη όψη του περιβάλλοντος έχει περιορισμούς τόσο στο εύρος όσο και στην ακρίβεια. Μπορεί να καλύψει μόνο μια περιορισμένη περιοχή. Έτσι, και η κάλυψη είναι μια σημαντική παράμετρος στα Α.Δ.Α.

Συνάθροιση δεδομένων (Data Aggregation)

Η παραγωγή δεδομένων με μεγάλο ποσοστό πλεονασμού από κόμβους ενός Α.Δ.Α. καθώς και παρόμοια μηνύματα από πολλαπλούς κόμβους, μπορούν να συναθροιστούν ώστε να μειωθούν οι μεταδόσεις. Συνάθροιση δεδομένων είναι ο συνδυασμός δεδομένων διαφορετικών πηγών σύμφωνα με μια συγκεκριμένη συνάρτηση συνάθροισης, όπως, κατάργηση διπλοτύπου (duplicate suppression), ελάχιστο, μέγιστο και μέσος όρος. Αυτή η τεχνική έχει χρησιμοποιηθεί για να επιτευχθεί εξοικονόμηση ενέργειας και βελτιστοποίηση της μετάδοσης δεδομένων σε μια πλειάδα πρωτοκόλλων δρομολόγησης. Τεχνικές επεξεργασίας σήματος μπορούν, επίσης, να χρησιμοποιηθούν για συνάθροιση δεδομένων. Σε αυτήν την περίπτωση, η οποία αναφέρεται ως σύντηξη δεδομένων (data fusion), κάθε κόμβος είναι ικανός να παράγει ένα πιο ακριβές σήμα εξόδου χρησιμοποιώντας τεχνικές όπως beamforming για να συνδυάσει το νεοαφιχθέν σήμα και να μειώσει το θόρυβο στα σήματα αυτά.

Ποιότητα υπηρεσιών

Σε κάποιες εφαρμογές είναι σημαντικό τα δεδομένα να παραδίδονται μέσα σε ένα καθορισμένο χρονικό διάστημα από τη στιγμή της μέτρησης αφού πιθανή καθυστέρηση θα τα καταστήσει άχρηστα. Μια, ακόμη, προϋπόθεση για εφαρμογές με χρονικούς περιορισμούς είναι, λοιπόν, η περιορισμένη καθυστέρηση της παράδοσης δεδομένων. Από την άλλη μεριά, σε πολλά δίκτυα η εξοικονόμηση ενέργειας, η οποία σχετίζεται άμεσα με το χρόνο για τον οποίο το δίκτυο μπορεί να είναι λειτουργικό, θεωρείται πολύ πιο σημαντική από την ποιότητα των δεδομένων που μεταδίδονται. Όσο το ενεργειακό απόθεμα του δικτύου μειώνεται το δίκτυο μπορεί να μειώσει την ποιότητα των αποτελεσμάτων με σκοπό να μειώσει την απώλεια ενέργειας στους κόμβους και άρα να επιμηκύνει το συνολικό χρόνο λειτουργικότητας του δικτύου. Άρα πρωτόκολλα δρομολόγησης με επίγνωση του ενεργειακού αποθέματος πρέπει να συνυπολογίζουν την παράμετρο της ποιότητας υπηρεσιών.

ΚΕΦΑΛΑΙΟ 2

ΠΡΩΤΟΚΟΛΛΑ ΔΡΟΜΟΛΟΓΗΣΗΣ ΣΕ Α.Δ.Α.

Σε αυτήν την ενότητα θα πραγματοποιηθεί μια αναδρομή στα state-of-the-art πρωτόκολλα του κλάδου. Γενικά, η δρομολόγηση σε Α.Δ.Α. μπορούν να χωριστεί σε *flat-based* δρομολόγηση, *hierarchical-based* δρομολόγηση και *location-based* δρομολόγηση ανάλογα με τη δομή του δικτύου. Στη flat-based δρομολόγηση όλοι οι κόμβοι έχουν, γενικά, ίσους ρόλους και λειτουργικότητα. Στη hierarchical-based δρομολόγηση, όμως, οι κόμβοι παίζουν διαφορετικό ρόλο μέσα στο δίκτυο. Στη location-based δρομολόγηση οι θέσεις των κόμβων χρησιμοποιούνται για να δρομολογηθούν τα δεδομένα μέσα από το δίκτυο. Ένα πρωτόκολλο δρομολόγησης θεωρείται προσαρμοστικό (adaptive) εάν κάποιες παράμετροι του συστήματος μπορούν να μεταβληθούν ώστε να προσαρμοστεί στις τρέχουσες συνθήκες του δικτύου αλλά και στα διαθέσιμα ενεργειακά επίπεδα. Επιπλέον, τα πρωτόκολλα μπορούν να ταξινομηθούν σε multipath-based, query-based, negotiation-based, QoS-based ή coherent-based τεχνικές δρομολόγησης, ανάλογα με τη λειτουργία του πρωτοκόλλου. Μια ακόμα διαφοροποίηση μπορεί να γίνει σύμφωνα με το πώς η πηγή βρίσκει ένα μονοπάτι προς τον προορισμό. Σε αυτήν την περίπτωση το πρωτόκολλα διακρίνονται σε proactive, reactive και hybrid πρωτόκολλα. Στα proactive πρωτόκολλα τα μονοπάτια υπολογίζονται πριν να είναι πραγματικά αναγκαία ενώ στα reactive υπολογίζονται μόνο μετά από αίτηση. Τα υβριδικά πρωτόκολλα είναι ένας συνδυασμός των δύο ιδεών. Όταν οι κόμβοι είναι στατικοί είναι προτιμότερο να χρησιμοποιείται πρωτόκολλο δρομολόγησης βασισμένο σε πίνακα (table-driven) παρά να χρησιμοποιείται κάποιο reactive πρωτόκολλο. Ένα σημαντικό ποσό ενέργειας χρησιμοποιείται στον υπολογισμό του μονοπατιού και στη λειτουργία του reactive πρωτοκόλλου. Μια άλλη κλάση πρωτοκόλλων δρομολόγησης είναι τα συνεργατικά (cooperative) πρωτόκολλα δρομολόγησης. Στη συνεργατική δρομολόγηση οι κόμβοι στέλνουν δεδομένα σε έναν κεντρικό κόμβο όπου συναθροίζονται και πιθανόν υπόκεινται σε επιπλέον επεξεργασία μειώνοντας, έτσι, την κατανάλωση ενέργειας. Άλλα πρωτόκολλα βασίζονται σε πληροφορίες χρόνου και θέσης.

2.1 Πρωτόκολλα βασισμένα στη δομή του δικτύου

2.1.1 Επίπεδη δρομολόγηση (Flat Routing)

Η πρώτη κατηγορία πρωτοκόλλων δρομολόγησης είναι τα επίπεδα πρωτόκολλα δρομολόγησης πολλών βημάτων. Αυτά τα πρωτόκολλα παρουσιάζουν μεγάλο ενδιαφέρον και ευκολία στην υλοποίηση. Γενικά, δεν απαιτούν γεωγραφικά δεδομένα αλλά δεν μπορούν να εγγυηθούν τη βέλτιστη λύση και ορισμένες φορές μπορεί να προκληθεί καθυστέρηση στην παράδοση των δεδομένων λόγω των πολλών βημάτων (multihop). Τα πιο σημαντικά πρωτόκολλα μαζί με μια περιγραφή της βασικής τους λειτουργίας αναπτύσσονται παρακάτω:

- *Sensor Protocols for Information via Negotiation (SPIN)*

  Ο Heinzelman πρότεινε μια οικογένεια προτοκόλλων που ονομάζεται SPIN (πρωτόκολλα αισθητήρων για πληροφορία μέσω διαπραγμάτευσης), τα οποία διανέμουν την πληροφορία σε κάθε κόμβο του δικτύου υποθέτοντας ότι όλοι οι κόμβοι είναι δυνάμει κόμβοι-βάσεις. Αυτό επιτρέπει σε κάθε χρήστη να ρωτήσει οποιονδήποτε κόμβο και να λάβει την πληροφορία αμέσως. Αυτά τα πρωτόκολλα

χρησιμοποιούν την ιδιότητα ότι κόμβοι σε κοντινή απόσταση έχουν παρόμοια δεδομένα και άρα χρειάζεται να διανεμηθούν στο δίκτυο μόνο τα δεδομένα που κάποιοι κόμβοι δεν κατέχουν. Η οικογένεια πρωτοκόλλων SPIN χρησιμοποιεί αλγόριθμους διαπραγμάτευσης δεδομένων που προσαρμόζονται στους πόρους του δικτύου. Κόμβοι που τρέχουν SPIN αναθέτουν ένα όνομα υψηλού επιπέδου για να περιγράψουν πλήρως τα δεδομένα που έχουν συγκεντρώσει (που ονομάζεται μετα-δεδομένο) και πραγματοποιούν διαπραγματεύσεις μεταδεδομένων πριν από την αποστολή οποιουδήποτε δεδομένου. Αυτό διασφαλίζει ότι δε στέλνονται περιττά δεδομένα μέσα στο δίκτυο. Η σημασία της μορφής των μεταδεδομένων δεν καθορίζεται από τα πρωτόκολλα και εξαρτάται από την εφαρμογή. Για παράδειγμα οι αισθητήριοι κόμβοι μπορεί να χρησιμοποιούν τα μοναδικά IDs για να στείλουν μεταδεδομένα όταν καλύπτουν μια προαποφασισμένη περιοχή. Επιπλέον, η οικογένεια πρωτοκόλλων SPIN, έχει πρόσβαση στο τρέχον ενεργειακό επίπεδο του κόμβου και χρησιμοποιεί το κατάλληλο πρωτόκολλο ανάλογα με την ενέργεια που απομένει. Αυτά τα πρωτόκολλα λειτουργούν με βάση το χρόνο και διανέμουν την πληροφορία σε όλο το δίκτυο ανεξάρτητα από το αν κάποιος χρήστης ζήτησε να λάβει δεδομένα. Ένα από τα πλεονεκτήματα των πρωτοκόλλων SPIN είναι ότι αλλαγές στην τοπολογία του δικτύου παραμένουν σε τοπικό μόνο επίπεδο, αφού κάθε κόμβος χρειάζεται να γνωρίζει μόνο τους κόμβους με τους οποίους μπορεί να επικοινωνήσει άμεσα, χωρίς άλλον ενδιάμεσο κόμβο. Τα πρωτόκολλα SPIN προσφέρουν ίδια εξοικονόμηση ενέργειας με την τεχνική υπερχείλισης και η διαπραγμάτευση μεταδεδομένων μειώνει στο μισό τα πλεονάζοντα δεδομένα. Παρ'όλα αυτά η δημοσιοποίηση των δεδομένων που γίνεται από το SPIN δεν μπορεί να εγγυηθεί την παράδοση των δεδομένων.

- *Directed Diffusion*

  Η κατευθυνόμενη διάχυση (Directed Diffusion) είναι μια δεδομενο-κεντρική (ΔΚ) μέθοδος δρομολόγησης που έχει επίγνωση της εκάστοτε εφαρμογής με την έννοια ότι όλα τα δεδομένα που παράγονται από κόμβους ονοματίζονται από ζευγάρια γνώρισμα-τιμή. Η βασική ιδέα της ΔΚ μεθόδου είναι ο συνδυασμός δεδομένων που έρχονται από διαφορετικές πηγές, από το εσωτερικό του δικτύου (ενδοδικτυακή συνάθροιση δεδομένων), ώστε να μειώνεται ο πλεονασμός και να ελαχιστοποιείται το πλήθος των μεταδόσεων. Έτσι, επιτυγχάνεται εξοικονόμηση ενέργειας και επιμήκυνση του χρόνου λειτουργικότητας του δικτύου. Αντίθετα με την παραδοσιακή end-to-end δρομολόγηση, η ΔΚ δρομολόγηση βρίσκει μονοπάτια από πολλαπλές πηγές σε έναν προορισμό που επιτρέπουν ενδοδικτυακή συνένωση (σύνθλιψη) των πλεοναζόντων δεδομένων. Οι βασικές έννοιες της κατευθυνόμενης διάχυσης είναι οι μεταβολές (gradients), τα ενδιαφέροντα (interests), ο κόμβος-βάση (Base Station) και φυσικά οι κόμβοι που απαρτίζουν το δίκτυο. Οι κόμβοι, λοιπόν, μετρούν γεγονότα και δημιουργούν στους αντίστοιχους γειτονικούς κόμβους μεταβολές στην πληροφορία. Ο κόμβος-βάση κάνει αιτήσεις για δεδομένα μεταδίδοντας καθολικά ενδιαφέροντα. Το κάθε ενδιαφέρον περιγράφει ένα έργο που πρέπει να έρθει σε πέρας από το δίκτυο. Ένα ενδιαφέρον διαχέεται στο δίκτυο από κόμβο σε κόμβο, μεταδιδόμενο καθολικά από κάθε κόμβο στους γείτονές του. Καθώς το ενδιαφέρον διαδίδεται μέσα στο δίκτυο, οι κόμβοι εγκαθιστούν μεταβολές προκειμένου να δημιουργηθούν οι συνθήκες, ώστε να ικανοποιηθεί η ερώτηση που δημιούργησε το διαδιδόμενο ενδιαφέρον. Γενικά, μια μεταβολή ορίζει την τιμή ενός γνωρίσματος και μια κατεύθυνση ορίζει μια μεταβολή προς τους κόμβους από τους οποίους παρέλαβε το ενδιαφέρον. Η ένταση της μεταβολής μπορεί να διαφέρει σε διαφορετικούς γείτονες του ίδιου κόμβου οδηγώντας, έτσι, σε

διαφοροποιημένες ροές πληροφορίας. Συνοπτικά, τα τρία βήματα λειτουργίας της κατευθυνόμενης διάχυσης είναι α) αποστολή ενδιαφερόντων, β) δημιουργία μεταβολών και γ) διασπορά δεδομένων. Όταν τα ενδιαφέροντα ταιριάξουν με τις μεταβολές δημιουργούνται μονοπάτια ροών δεδομένων από πολλαπλά μονοπάτια και τελικά επιλέγονται τα καλύτερα ώστε να παραμείνει μειωμένη η κατανάλωση ενέργειας για την επικοινωνία. Επιπλέον, για τον ίδιο λόγο, τα δεδομένα συναθροίζονται καθώς προχωρούν μέσα στο δίκτυο και ο σκοπός είναι να βρεθεί ένα καλό δέντρο συνάθροισης το οποίο θα οδηγεί τα δεδομένα από τις πηγές στον προορισμό, στον κόμβο-βάση. Από τη στιγμή που θα αρχίσει η λήψη των δεδομένων από τον κόμβο-βάση, αυτός ξαναστέλνει τα ενδιαφέροντά του, αφού αυτά δεν έχουν εγγυημένη μετάδοση μέσα στο δίκτυο. Η κατευθυνόμενη διάχυση διαφέρει από τα πρωτόκολλα της οικογένειας SPIN σε δύο σημεία. Αρχικά, στην κατευθυνόμενη διάχυση μεταδίδονται δεδομένα μετά από αίτηση του κόμβου-βάση, ενώ στα πρωτόκολλα SPIN οι κόμβοι δημοσιοποιούν τη διαθεσιμότητα δεδομένων ώστε οι ενδιαφερόμενοι κόμβοι να τα ζητήσουν. Δεύτερον, όλες οι επικοινωνίες στην κατευθυνόμενη διάχυση είναι μεταξύ γειτόνων κάτι που επιτρέπει συνάθροιση και προσωρινή αποθήκευση δεδομένων. Αντίθετα, με τα πρωτόκολλα SPIN δε χρειάζεται να φυλάσσεται μια καθολική τοπολογία του δικτύου. Το αρνητικό για την κατευθυνόμενη διάχυση είναι ότι δεν μπορεί να χρησιμοποιηθεί σε κάποιες εφαρμογές που θα απαιτούν συνεχή παράδοση δεδομένων στον κόμβο-βάση. Αυτό οφείλεται στο ότι η λογική της κατευθυνόμενης διάχυσης είναι να παρέχονται τα δεδομένα μετά από αίτηση και οι πολλές προωθήσεις που θα απαιτούνται στην περίπτωση αυτή θα προσθέσουν επιπλέον πληροφορίες προς προσωρινή αποθήκευση.

- *Rumor Routing*

  Η δρομολόγηση διάδοσης (Rumor Routing) είναι μια παραλλαγή της κατευθυνόμενης διάχυσης και προορίζεται για εφαρμογές που η γεωγραφική δρομολόγηση δεν είναι εφικτή. Γενικά, η κατευθυνόμενη διάχυση χρησιμοποιεί υπερχείλιση (επαναλαμβανόμενες καθολικές μεταδόσεις) για να μοιράσει την επερώτηση σε όλο το δίκτυο στην περίπτωση που δεν υπάρχουν γεωγραφικά κριτήρια, για να διαμοιραστεί το έργο σε διάφορους κόμβους. Κάποιες φορές, όμως, υπάρχουν μόνο λίγα δεδομένα τα οποία ζητήθηκαν και δεν είναι απαραίτητο να γίνει υπερχείλιση του δικτύου. Μια εναλλακτική προσέγγιση είναι να πραγματοποιηθεί υπερχείλιση στα γεγονότα όταν ο αριθμός των γεγονότων είναι μικρός και ο αριθμός των επερωτήσεων είναι μεγάλος. Η βασική ιδέα είναι να δρομολογηθούν οι επερωτήσεις προς κόμβους που έχουν παρατηρήσει κάποιο γεγονός, παρά να υπερχειλιστεί όλο το δίκτυο, για να ανακτηθεί η πληροφορία για τα γεγονότα που λαμβάνουν χώρα. Για να επιτευχθεί υπερχείλιση γεγονότων μέσα στο δίκτυο ο αλγόριθμος δρομολόγησης διάδοσης χρησιμοποιεί κάποια πακέτα που παραμένουν για μεγάλο χρονικό διάστημα στο δίκτυο τα οποία ονομάζονται πράκτορες (agents). Όταν κάποιος κόμβος εντοπίσει ένα γεγονός το προσθέτει σε έναν τοπικό πίνακα, ο οποίος ονομάζεται πίνακας γεγονότων και παράγει έναν πράκτορα. Οι πράκτορες ταξιδεύουν μέσα στο δίκτυο με σκοπό να διαδόσουν πληροφορία σχετικά με τοπικά γεγονότα σε απομακρυσμένους κόμβους. Όταν ένας κόμβος παράγει μια επερώτηση για ένα γεγονός, ο κόμβος που γνωρίζει τη διαδρομή μπορεί να απαντήσει στην επερώτηση κοιτώντας τον τοπικό πίνακα γεγονότων. Συνεπώς, δε χρειάζεται να υπερχειλιστεί όλο το δίκτυο, κάτι που μειώνει δραστικά το κόστος επικοινωνίας. Από την άλλη μεριά αυτός ο αλγόριθμος δρομολόγησης διατηρεί μόνο ένα μονοπάτι μεταξύ πηγής και προορισμού σε αντίθεση με την κατευθυνόμενη διάχυση που τα δεδομένα μπορεί να δρομολογηθούν μέσω πολλαπλών

μονοπατιών με χαμηλούς ρυθμούς. Αποτελέσματα προσομοίωσης έδειξαν ότι η δρομολόγηση διάδοσης μπορεί να επιτύχει σημαντική εξοικονόμηση ενέργειας σε σχέση με την υπερχείλιση γεγονότων και μπορεί να χειριστεί τις περιπτώσεις που κάποιος κόμβος δεν είναι πια λειτουργικός. Το αρνητικό είναι ότι η επίδοση του αλγορίθμου είναι καλή μόνο όταν ο αριθμός των γεγονότων είναι μικρός. Για μεγάλο αριθμό γεγονότων το κόστος συντήρησης των πρακτόρων και των πινάκων γεγονότων σε κάθε κόμβο καταλήγει να είναι μη πρακτικό εάν δεν υπάρχει ενδιαφέρον για αυτά τα γεγονότα από τον κόμβο-βάση.

- *Minimum Cost Forwarding Algorithm (MCFA)*

  Ο αλγόριθμος αυτός (αλγόριθμος ελαχίστου κόστους προώθησης) εκμεταλλεύται το γεγονός ότι η κατεύθυνση της δρομολόγησης είναι πάντα γνωστή και συγκεκριμένα, προς τον στατικό, εξωτερικό, κόμβο-βάση. Συνεπώς, κάθε κόμβος δε χρειάζεται να έχει ένα μοναδικό χαρακτηριστικό (ID) ούτε να συντηρεί έναν πίνακα δρομολόγησης. Αντί γι' αυτά, κάθε κόμβος συντηρεί την πληροφορία για το μονοπάτι με το μικρότερο κόστος προς τον κόμβο-βάση. Κάθε μήνυμα που πρέπει να προωθηθεί από κάποιον κόμβο στέλνεται καθολικά σε όλους τους γείτονές του, και κάθε κόμβος που λαμβάνει ένα μήνυμα ελέγχει εάν βρίσκεται πάνω στο μονοπάτι ελαχίστου κόστους μεταξύ της πηγής και του προορισμού, και αν ναι, η διαδικασία επαναλαμβάνεται μέχρι να παραδοθεί το μήνυμα στον προορισμό του. Στον αλγόριθμο ελαχίστου κόστους προώθησης κάθε κόμβος πρέπει να ξέρει το μονοπάτι ελαχίστου κόστους από τον εαυτό του στον κόμβο-βάση. Για να επιτευχθεί αυτό ο κόμβος-βάση στέλνει σε όλους ένα μήνυμα με το κόστος ίσο με μηδέν, ενώ όλοι οι κόμβοι έχουν θέσει το ελάχιστο κόστος επικοινωνίας με τον κόμβο-βάση ίσο με άπειρο. Κάθε κόμβος που λαμβάνει το καθολικό μήνυμα από τον κόμβο-βάση ελέγχει αν το άθροισμα της εκτίμησης του κόστους που βρίσκεται στο μήνυμα και του κόστους της επικοινωνίας με τον κόμβο από τον οποίο έλαβε το μήνυμα είναι μικρότερο από την τρέχουσα εκτίμηση και, αν ναι, ανανεώνεται τόσο το κόστος που φυλάσσεται στον κόμβο για το κόστος επικοινωνίας μεταξύ του εαυτού του και του κόμβου-βάση, όσο και η τιμή που βρίσκεται στο καθολικό μήνυμα και επαναστέλνεται σε όλους τους κόμβους. Αυτή η διαδικασία μπορεί να οδηγήσει σε πολλαπλές ενημερώσεις. Γενικά, όσο πιο μακρυά είναι ένας κόμβος από τον κόμβο-βάση τοσο πιο πολλές ενημερώσεις θα πρέπει να κάνει. Για να αποφευχθεί αυτό, χρησιμοποιείται ένας αλγόριθμος αποκοπής κατά τη διαδικασία έναρξης του πρώτου αλγορίθμου. Αυτός ο αλγόριθμος καθορίζει ότι ένας κόμβος δε θα δέχεται ενημερώσεις αν δεν περάσουν τουλάχιστον $a * l_c$ μονάδες χρόνου από την προηγούμενη ενημέρωση, όπου a μια σταθερά και l_c το κόστος επικοινωνίας του τελευταίου μηνύματος που οδήγησε στην ενημέρωση.

- *Gradient-Based Routing (GBR)*

  Μια άλλη παραλλαγή της κατευθυνόμενης διάχυσης είναι η βασισμένη στη μεταβολή δρομολόγηση. Η βασική ιδέα του αλγορίθμου αυτού είναι να απομνημονευθεί ο αριθμός των βημάτων όταν το ενδιαφέρον έχει διαδοθεί σε ολόκληρο το δίκτυο. Με αυτόν τον τρόπο, κάθε κόμβος μπορεί να υπολογίσει μια παράμετρο αποκαλούμενη ύψος του κόμβου, η οποία είναι ο ελάχιστος αριθμός βημάτων που απαιτούνται για να φτάσει στον κόμβο-βάση. Η διαφορά μεταξύ του ύψους ενός κόμβου και του ύψους του γείτονά του θεωρείται ως η μεταβολή της σύνδεσης μεταξύ των δύο κόμβων. Ένα πακέτο διαβιβάζεται σε μια σύνδεση με τη μεγαλύτερη μεταβολή. Ο αλγόριθμος χρησιμοποιεί μερικές βοηθητικές τεχνικές, όπως η συνάθροιση δεδομένων και ο διαμοιρασμός της κίνησης, προκειμένου να διαιρεθεί ομοιόμορφα η κίνηση μέσα στο δίκτυο. Όταν πολλαπλά

μονοπάτια περνούν από τον ίδιο κόμβο, ο οποίος λειτουργεί ως κόμβος επικοινωνίας, ο κόμβος αυτός μπορεί να συνδυάσει τα δεδομένα σύμφωνα με μια καθορισμένη συνάρτηση. Στη βασισμένη στη μεταβολή δρομολόγηση, έχουν συζητηθεί διάφορες τεχνικές διασποράς δεδομένων των οποίων ο κύριος στόχος είναι η επίτευξη μιας ισορροπημένης διανομής της κίνησης στο δίκτυο, αυξάνοντας κατά συνέπεια τη διάρκεια ζωής του δικτύου. Τα αποτελέσματα προσομοίωσης της βασισμένης στην μεταβολή δρομολόγησης έδειξαν ότι έχει καλύτερη απόδοση από την κατευθυνόμενη διάχυση ως προς την συνολική ενέργειας επικοινωνίας.

- *COUGAR*

  Ένα άλλο δεδομενο-κεντρικό πρωτόκολλο είναι το COUGAR το οποίο βλέπει το δίκτυο ως ένα μεγάλο σύστημα κατανεμημένης βάσης δεδομένων. Η κεντρική ιδέα είναι η χρησιμοποίηση δηλωτικών επερωτήσεων έτσι ώστε να ανεξαρτητοποιηθεί η επεξεργασία επερωτήσεων από τις λειτουργίες του επιπέδου δικτύου (network layer functions), όπως η επιλογή κατάλληλων κόμβων. Το πρωτόκολλο COUGAR χρησιμοποιεί ενδοδικτυακή συνάθροιση δεδομένων για να επιτύχει επιπλέον εξοικονόμηση ενέργειας και ενσωματώνει μια αρχιτεκτονική για το σύστημα βάσης δεδομένων των αισθητήρων, κατά την οποία, οι κόμβοι επιλέγουν έναν κόμβο-ηγέτη για να επιτελέσει τη συνάθροιση και την αποστολή των δεδομένων στον κόμβο-βάση. Ο κόμβος-βάση είναι υπεύθυνος για τη δημιουργία ενός σχεδίου επερωτήσεων, το όποιο προσδιορίζει την αναγκαία πληροφορία για τη ροή των δεδομένων και είναι υπεύθυνος για τους ενδοδικτυακούς υπολογισμούς για την εκάστοτε επερώτηση και την αποστολή της στους σχετικούς κόμβους. Στο σχέδιο αυτό περιγράφεται και η διαδικασία επιλογής του κόμβου-ηγέτη για την επερώτηση. Η αρχιτεκτονική παρέχει ενδοδικτυακή υπολογιστική ικανότητα, η οποία μπορεί να βοηθήσει στην εξοικονόμηση ενέργειας, σε περίπτωση που τα παραγόμενα δεδομένα είναι τεράστια. Όμως, το πρωτόκολλο αυτό παρουσιάζει και κάποιες αδυναμίες. Αρχικά, η προσθήκη του επιπέδου επερώτησης («πάνω» από τα άλλα επίπεδα του δικτύου) ενδεχομένως να οδηγεί σε επιπλέον κατανάλωση ενέργειας και δέσμευση μνήμης. Κατά δεύτερον, για να επιτευχθεί επιτυχής ενδοδικτυακή επεξεργασία δεδομένων χρειάζεται συγχρονισμός μεταξύ των κόμβων (δεδομένου ότι δε λαμβάνονται ταυτόχρονα τα δεδομένα από όλες τις πηγές) πριν αποσταλούν τα δεδομένα στον κόμβο-ηγέτη. Τρίτον, οι κόμβοι ηγέτες πρέπει να συντηρούνται δυναμικά ώστε να αποφευχθεί η υπερβολική αύξηση της κίνησης του δικτύου που εξυπηρετούν.

- *ACQUIRE*

  Μια τεχνική για να τίθενται επερωτήσεις σε δίκτυα αισθητήρων είναι η ACQUIRE (ACtive QUery forwarding In sensoR nEtworks). Παρόμοια με το COUGAR, η τεχνική ACQUIRE βλέπει το δίκτυο ως μια κατανεμημένη βάση δεδομένων όπου οι σύνθετες επερωτήσεις μπορούν να διαιρεθούν περαιτέρω σε υπο-επερωτήσεις. Η λειτουργία του ACQUIRE μπορεί να περιγραφεί ως εξής. Ο κόμβος-βάση στέλνει μια επερώτηση, η οποία προωθείται από κάθε κόμβο που τη λαμβάνει. Κατά τη διάρκεια αυτής της διαδικασίας, κάθε κόμβος προσπαθεί να απαντήσει στην επερώτηση μερικώς με τη χρησιμοποίηση προ-αποθηκευμένης πληροφορίας και, στη συνέχεια, προωθεί την απάντηση σε έναν άλλο κόμβο. Εάν η προ-αποθηκευμένη πληροφορία δεν είναι έγκυρη, οι κόμβοι συγκεντρώνουν πληροφορία από τους γείτονές τους σε απόσταση το πολύ *d* βημάτων. Μόλις απαντηθεί πλήρως η επερώτηση, στέλνεται μέσω του αντιστρόφου μονοπατιού είτε του μονοπατιού ελαχίστου κόστους προς τον

κόμβο-βάση το αποτέλεσμα. Ως εκ τούτου, το ACQUIRE μπορεί να ανταπεξέλθει σε σύνθετες επερωτήσεις επιτρέποντας σε πολλούς κόμβους να στέλνουν απαντήσεις. Είναι σημαντικό ότι η κατευθυνόμενη διάχυση δεν είναι τόσο αποτελεσματική για τις σύνθετες επερωτήσεις, ενώ η τεχνική ACQUIRE είναι πολύ αποτελεσματική ενημερώνοντας κατάλληλα την τιμή της παραμέτρου *d*. Όταν το *d* είναι ίσο με τη διάμετρο του δικτύου τότε η συμπεριφορά του ACQUIRE είναι παρόμοια με τη συμπεριφορά της υπερχείλισης δικτύου. Εντούτοις, η επερώτηση πρέπει να κάνει περισσότερα βήματα εάν το *d* είναι πάρα πολύ μικρό. Η βέλτιστη τιμή της παραμέτρου *d* για ένα πλέγμα όπου κάθε κόμβος έχει 4 άμεσους γείτονες έχει υπολογιστεί με χρήση μαθηματική μοντελοποίησης. Όμως, η προσομοίωση δεν επιβεβαίωσε πλήρως τα αποτελέσματα. Προκειμένου να επιλεγεί ο επόμενος κόμβος για την προώθηση της επερώτησης, η τεχνική ACQUIRE είτε επιλέγει τυχαία έναν κόμβο είτε η επιλογή βασίζεται στο ποιος κόμβος έχει τη μεγαλύτερη πιθανότητα να ικανοποιήσει την επερώτηση.

- *Energy Aware Routing*

  Ο στόχος του ενεργειακά ενήμερου πρωτοκόλλου δρομολόγησης, ένα reactive ενεργοποιούμενο από τον προορισμό πρωτόκολλο, είναι να επιμηκυνθεί η λειτουργική περίοδος του δικτύου. Αν και το πρωτόκολλο αυτό είναι παρόμοιο με την κατευθυνόμενη διάχυση, διαφέρει κατά το ότι διατηρεί ένα σύνολο από μονοπάτια για κάθε ζεύξη, αντί για το βέλτιστο μονοπάτι, ως προς το ρυθμό μετάδοσης. Αυτά τα μονοπάτια επιλέγονται και συντηρούνται με χρήση μιας συγκεκριμένης πιθανότητας. Η τιμή αυτής της πιθανότητας εξαρτάται από το πόσο χαμηλή θα είναι η κατανάλωση ενέργειας σε κάθε ένα μονοπάτι. Η ενέργεια ενός μόνο μονοπατιού δεν πρόκειται να μηδενιστεί αφού επιλέγονται μονοπάτια που δεν είναι υποχρεωτικά ίδια σε διαφορετικές χρονικές στιγμές. Αυτό οδηγεί σε μεγαλύτερο χρόνο ζωής του δικτύου μιας και η ενέργεια καταναλώνεται ισομερώς στους κόμβους του δικτύου. Η βασική μετρική του πρωτοκόλλου είναι η βιωσιμότητα του δικτύου. Το πρωτόκολλο υποθέτει ότι κάθε κόμβος μπορεί να προσπελαστεί με χρήση μιας διευθυνσιοδότησης, η οποία είναι βασισμένη σε κλάσεις που ενσωματώνει τη θέση και τον τύπο των κόμβων. Το πρωτόκολλο αρχικοποιεί τη σύνδεση με χρήση τοπικής υπερχείλισης (localized flooding), που βοηθά στην εύρεση όλων των μονοπατιών μεταξύ πηγών και προορισμών καθώς και στο ενεργειακό τους κόστος. Ακολούθως, δημιουργούνται οι πίνακες δρομολόγησης, αφού πρώτα αφαιρεθούν τα μονοπάτια υψηλού κόστους. Στη συνέχεια, χρησιμοποιούνται οι πίνακες δρομολόγησης για να προωθηθούν τα δεδομένα προς τον παραλήπτη, και η πιθανότητα επιλογής του κάθε κόμβου εξαρτάται από το κόστος του. Τα μονοπάτια παραμένουν ζωντανά με επαναλαμβανόμενες τοπικές υπερχειλίσεις. Το πρωτόκολλο αυτό, σε σχέση με την κατευθυνόμενη διάχυση, βελτιώνει την εξοικονόμηση ενέργειας κατά 21.5% και επιμηκύνει το λειτουργικό χρόνο του δικτύου κατά 44%. Παρ'όλα αυτά, η τεχνική αυτή απαιτεί την περισυλλογή πληροφοριών για τη θέση του κάθε κόμβου και τη δημιουργία μηχανισμού διευθυνσιοδότητησης για όλους τους κόμβους. Αυτό περιπλέκει τη δημιουργία μονοπατιών σε σχέση με την κατευθυνόμενη διάχυση.

2.1.2 Ιεραρχική δρομολόγηση (Hierarchical Routing)

Η ιεραρχική ή αλλιώς, η βασισμένη σε ομάδες δρομολόγηση αναφέρεται, γενικά, σε ετερογενή δίκτυα (με ασύρματους και ενσύρματους κόμβους) και πλεονεκτεί ως προς την επεκτασιμότητα και την οικονομική επικοινωνία. Γενικά, τα πρωτόκολλα αυτά χρησιμοποιούνται για να υλοποιηθεί ενεργειακά οικονομική δρομολόγηση σε Α.Δ.Α.. Οι κόμβοι υψηλής ενέργειας χρησιμοποιούνται για επεξεργασία και προώθηση δεδομένων, ενώ οι κόμβοι χαμηλής ενέργειας χρησιμοποιούνται για μετρήσεις στην επιθυμητή τοποθεσία. Έτσι, η ανάθεση ενεργειών σε κόμβους-ηγέτες μπορεί να συμβάλει ουσιωδώς στην επεκτασιμότητα, στο χρόνο ζωής του δικτύου και στην εξοικονόμηση ενέργειας. Η ιεραρχική δρομολόγηση είναι ένας αποτελεσματικός τρόπος να μειωθεί η κατανάλωση ενέργειας σε μια ομάδα κόμβων κάνοντας συνάθροιση δεδομένων. Τα επίπεδα της ιεραρχικής δρομολόγησης είναι δύο, στο ένα επίπεδο επιλέγονται οι κόμβοι-ηγέτες και στο άλλο επιτελείται η δρομολόγηση.

- *LEACH protocol*

  Ένας ιεραρχικός αλγόριθμος ομαδοποίησης για δίκτυα αισθητήρων είναι το πρωτόκολλο LEACH (Low Energy Adaptive Clustering Hierarchy – Προσαρμοστική Ιεραρχία Ομαδοποίησης Χαμηλής Ενέργειας). Το LEACH επιλέγει τυχαία κάποιους κόμβους ως κεφαλές ομαδοποίησης (Cluster Heads – CHs) και εναλλάσει τους ρόλους, ώστε να κατανεμηθεί ισόποσα το ενεργειακό φορτίο στους κόμβους του δικτύου. Η εκάστοτε κεφαλή ομαδοποίησης συμπιέζει τα δεδομένα που λαμβάνει από κάποιον κόμβο που ανήκει στην ομάδα της και στέλνει ένα συναθροισμένο πακέτο στον κόμβο-βάση, έτσι ώστε να μειωθεί η ποσότητα της πληροφορία που πρέπει να μεταδοθεί προς τον κόμβο-βάση. Το LEACH χρησιμοποιεί TDMA/CDMA MAC τις συγκρούσεις πακέτων κατά τη διάρκεια της επικοινωνίας, τόσο στο εσωτερικό μιας ομάδας, όσο και μεταξύ διαφορετικών ομάδων. Ωστόσο, η συλλογή δεδομένων γίνεται κεντρικά και πραγματοποιείται περιοδικά. Επομένως, το πρωτόκολλο είναι πιο κατάλληλο όταν χρειάζεται συνεχής παρακαλούθηση κάποιου φυσικού μεγέθους από το δίκτυο αισθητήρων. Ο χρήστης του δικτύου ενδέχεται να μη χρειάζεται όλα τα δεδομένα την ίδια στιγμή, οπότε οι περιοδικές μεταδόσεις δεδομένων, οι οποίες θα οδηγούσαν σε γρήγορη κατανάλωση του περιορισμένου ενεργειακού αποθέματος του κάθε κόμβου, είναι περιττές. Μετά από ένα δεδομένο χρονικό διάστημα, πραγματοποιείται μια τυχαία εναλλαγή ρόλων των κεφαλών ομαδοποίησης, ώστε να επιτευχθεί ισοκατανεμημένη κατανάλωση ενέργειας διαμέσου του δικτύου. Αποτελέσματα προσομοίωσης των δημιουργών του πρωτοκόλλου δείχνουν ότι μόνο το 5% των κόμβων χρειάζεται να διαδραματίσουν το ρόλο της κεφαλής ομαδοποίησης. Η λειτουργία του πρωτοκόλλου χωρίζεται σε δύο φάσεις, τη φάση εγκατάστασης και τη στατική φάση. Κατά την πρώτη φάση επιλέγονται οι κεφαλές ομαδοποίησης σύμφωνα με κάποιο πιθανοθεωρητικό μοντέλο και κατά τη δεύτερη φάση πραγματοποιείται η μετάδοση δεδομένων προς τον κόμβο-βάση. Πιο συγκεκριμένα, οι κόμβοι που μετρούν κάποιο φυσικό μέγεθος, στέλνουν τις μετρήσεις τους στην κεφαλή της εκάστοτε ομάδας τους και ο κόμβος που διαδραματίζει αυτό το ρόλο συναθροίζει τα δεδομένα και τα αποστέλλει στον κόμβο-βάση. Μετά από ένα χρονικό διάστημα το δίκτυο επανέρχεται στη φάση εγκατάστασης, επιλέγονται νέες κεφαλές ομαδοποίησης και ακολουθεί εκ νέου η στατική φάση. Κάθε ομάδα επικοινωνεί χρησιμοποιώντας διαφορετικούς CDMA κώδικες, ώστε να μειωθεί η παρεμβολή μεταξύ κόμβων που ανήκουν σε διαφορετικές ομάδες. Αν και το LEACH είναι ικανό να επιμηκύνει το χρόνο ζωής του δικτύου υπάρχουν κάποια

ερωτήματα για τις προϋποθέσεις που χρησιμοποιούνται από το πρωτόκολλο. Το LEACH υποθέτει ότι όλοι οι κόμβοι μπορούν να μεταδώσουν μηνύματα με όση ισχύ χρειάζεται για να φτάσουν στον κόμβο-βάση εάν αυτό χρειάζεται, και ότι κάθε κόμβος έχει αρκετή υπολογιστική ισχύ ώστε να μπορεί να υποστηρίζει διάφορα MAC πρωτόκολλα. Συνεπώς, το πρωτόκολλο δεν είναι εφαρμόσιμο σε δίκτυα που έχουν παραταχθεί σε περιοχές εξαιρετικά εκτεταμένες. Η τυχαιότητα της επιλογής των κεφαλών ομαδοποιήσης μπορεί να οδηγήσει στη δημιουργία μεγάλων περιοχών χωρίς μια κεφαλή και, άρα, να υπάρχουν κόμβοι που να μην μπορούν να μεταδώσουν τις μετρήσεις τους. Η δυναμική ομαδοποίηση που θα χρησιμοποιηθεί στην περίπτωση αυτή, απαιτεί επιπλέον έξοδα τα οποία μπορεί να εκμηδενίσουν το ενεργειακό κέρδος που έχει επιτευχθεί από τη δρομολόγηση. Τέλος, το πρωτόκολλο υποθέτει πως όλοι οι κόμβοι ξεκινούν με το ίδιο απόθεμα ενέργειας σε κάθε φάση εγκατάστασης και, άρα, υποθέτει πώς όλοι οι κόμβοι που είναι κεφαλές καταναλώνουν σχεδόν την ίδια ενέργεια. Μια επέκταση του LEACH, το LEACH με διαπραγμάτευση, χρησιμοποιεί μεταδεδομένα για να εξασφαλίσει πως μόνο μηνύματα που προσφέρουν νέα πληροφορία θα μεταδοθούν προς τις κεφαλές ομαδοποίησης και στη συνέχεια προς τον κόμβο-βάση. Στον παρακάτω πίνακα φαίνεται μια σύγκριση μεταξύ των πρωτοκόλλων SPIN, LEACH, και της Κατευθυνόμενης Διάχυσης.

Πίνακας 2.1: Σύγκριση μεταξύ SPIN, LEACH και Κατευθυνόμενης Διάχυσης

	SPIN	**LEACH**	**Directed Diffusion**
Optimal Route	No	No	Yes
Network Lifetime	Good	Very Good	Good
Resource Awareness	Yes	Yes	Yes
Use of Meta-Data	Yes	No	Yes

- *Power-Efficient Gathering in Sensor Information Systems (PEGASIS)*

  Μια επαύξηση του LEACH είναι το πρωτόκολο PEGASIS. Το πρωτόκολλο αυτό είναι ένα σχεδόν βέλτιστο πρωτόκολλο βασισμένο στη λογική της αλυσίδας. Η βασική ιδέα είναι, πως προκειμένου να επιμηκυνθεί ο χρόνος ζωής του δικτύου, οι κόμβοι επικοινωνούν μόνο με τους πιο κοντινούς τους κόμβους και παίρνουν σειρά για την επικοινωνία τους με τον κόμβο-βάση. Όταν επικοινωνήσουν όλοι με τον κόμβο-βάση τότε ένας νέος «γύρος» επικοινωνίας αρχίζει. Αυτή η διαδικασία μειώνει την ενέργεια που απαιτείται για μια μετάδοση ανά γύρο επικοινωνίας αφού η κατανάλωση ενέργειας διαμοιράζεται ομοιόμορφα σε όλους τους κόμβους. Συνεπώς, το PEGASIS έχει δύο κύριους στόχους. Ο πρώτος, είναι η επιμήκυνση της ζωής του κάθε κόμβου με συνεργατικές τεχνικές, κάτι που οδηγεί σε επιμήκυνση της ζωής του δικτύου. Ο δεύτερος, είναι να επιτραπεί μόνο τοπικός συντονισμός μεταξύ των κόμβων, ώστε να μειωθεί το εύρος-ζώνης που χρησιμοποιείται για την επικοινωνία. Αντίθετα με το LEACH, το PEGASIS αποφεύγει τη δημιουργία ομάδων και χρησιμοποιεί μόνο έναν κόμβο ανά αλυσίδα για να μεταδοθεί κάποιο μήνυμα στον κόμβο-βάση αντί για πολλαπλούς κόμβους. Για να εντοπιστεί ο κοντινότερος γείτονας, κάθε κόμβος χρησιμοποιεί την ένταση του σήματος, ώστε, τελικά, να ρυθμιστεί η ένταση τόσο ώστε μόνο

ένας κόμβος να μπορεί να τον ακούσει. Έτσι με έναν άπληστο αλγόριθμο δημιουργούνται μονοπάτια προς τον κόμβο-βάση. Αποτελέσματα προσομοίωσης δείχνουν ότι το πρωτόκολλο PEGASIS μπορεί να διπλασιάσει το χρόνο ζωής του δικτύου σε σχέση με το πρωτόκολλο LEACH. Αυτό επιτυγχάνεται λόγω της απουσίας δυναμικής δημιουργίας ομάδων, καθώς και λόγω της μειώσης των αποστολών και των λήψεων δεδομένων λόγω συνάθροισης δεδομένων. Παρόλο όμως, που αποφεύγεται το κόστος ομαδοποίησης, το πρωτόκολλο PEGASIS απαιτεί δυναμική ενημέρωση τοπολογίας, κάτι που μπορεί να οδηγήγησει σε μεγάλο κόστος σε περίπτωση δικτύου έντονης χρήσης. Επιπλέον, το πρωτόκολλο υποθέτει πως κάθε κόμβος είναι ικανός να επικοινωνήσει με τον κόμβο-βάση, ενώ, πρακτικά, οι περισσότεροι κόμβοι χρησιμοποιούν multihop επικοινωνία.

- *Threshold-sensitive Energy Efficient Protocol (TEEN και APTEEN)*

  Δύο ακόμα ιεραρχικά πρωτόκολλα δρομολόγησης είναι το TEEN (Threshold-sensitive Energy Efficient Protocol) και το APTEEN (Adaptive Periodic Threshold-sensitive Energy Efficient Protocol). Αυτά τα πρωτόκολλα προτείνονται για εφαρμογές που επηρεάζονται σημαντικά από το χρόνο. Στο πρωτόκολλο TEEN οι αισθητήρες μετράνε συνεχώς το αντίστοιχο φυσικό φαινόμενο, αλλά τα δεδομένα μεταδίδονται πιο αραιά. Ένας κόμβος που έχει επιλεγεί ως κεφαλή μιας ομάδας κόμβων στέλνει στα μέλη της ομάδας ένα ισχυρό κατώφλι (hard threshold), που είναι η τιμή κατωφλίου του υπό μέτρηση φαινομένου, και ένα ασθενές κατώφλι (soft threshold), το οποίο είναι η τιμή της επιτρεπόμενης μεταβολής της τιμής του ισχυρού κατωφλίου που ενεργοποιεί το μηχανισμό μετάδοσης του κόμβου και επιχειρεί την αποστολή της πληροφορίας. Συνεπώς, το ισχυρό κατώφλι μειώνει τον αριθμό των μεταδόσεων επιτρέποντας στους κόμβους να μεταδώσουν μόνο όταν η τιμή του υπό μέτρηση φυσικού μεγέθους ανήκει σε ένα σύνολο τιμών που ενδιαφέρει την εφαρμογή. Το ασθενές κατώφλι μειώνει περαιτέρω τις μεταδόσεις, αποτρέποντάς τις, στις περιπτώσεις που υπάρχει μικρή ή καμμία μεταβολή στην νέα μέτρηση. Μελετώντας τη συμπεριφορά του δικτύου για διάφορες τιμές των κατωφλίων, παρατηρείται ότι για μικρές τιμές του ασθενούς κατωφλίου επιτυγχάνουμε μεγαλύτερη ακρίβεια στα δεδομένα, αλλά μικρότερη εξοικονόμηση ενέργειας. Άρα, ο χρήστης μπορεί να ανταλλάξει ακρίβεια με εξοικονόμηση ενέργειας, και ανάποδα, ανάλογα με την εφαρμογή. Και σε αυτό το πρωτόκολλο, μετά από ένα χρονικό διάστημα, αλλάζουν οι κόμβοι που παίζουν το ρόλο των κεφαλών των ομάδων και στέλνονται καθολικά νέες τιμές για τις παραμέτρους. Το βασικό μειονέκτημα αυτής της ιδέας είναι ότι αν αποτύχουν τα μηνύματα με τις τιμές των παραμέτρων, τότε, οι κόμβοι δε θα επικοινωνήσουν ποτέ και ο χρήστης δε θα επιτύχει ποτέ την επιθυμητή επικοινωνία. Στο πρωτόκολλο APTEEN αντί να διανέμονται στο δίκτυο μόνο οι παράμετροι ισχυρό και ασθενές κατώφλι, χρησιμοποιούνται οι εξής παράμετροι: α) ιδιότητες (attributes), που είναι τα υπό μέτρηση φυσικά μεγέθη για τα οποία ο χρήστης συλλέγει πληροφορίες, β) κατώφλια (thresholds), το ισχυρό και ασθενές κατώφλι όπως προαναφέρθηκε, γ) χρονοδιάγραμμα (schedule), το οποίο είναι το χρονοδιάγραμμα για το TDMA το οποίο αναθέτει ένα slot σε κάθε κόμβο και δ) ένα χρονικό διάστημα (count time) το οποίο είναι το μέγιστο χρονικό διάστημα μεταξύ δύο διαδοχικών αποστολών από έναν κόμβο. Η βασική ιδέα εδώ είναι ότι οι κόμβοι, που μετρούν συνεχώς τα διάφορα φυσικά μεγέθη, στέλνουν τις μετρήσεις μόνο εάν η τιμή είναι μεγαλύτερη από το ισχυρό κατώφλι και η διαφορά από την προηγούμενη μέτρηση είναι μεγαλύτερη από ασθενές κατώφλι. Επίσης αν ένας κόμβος δε στείλει κάποια μέτρηση για χρονικό διάστημα μεγαλύτερο από το προκαθορισμένο υποχρεούται

να ξαναμετρήσει και να αποστείλει την επόμενη τιμή. Το μοναδικό ελάττωμα του πρωτοκόλου αυτού είναι το απαιτούμενο υπολογιστικό κόστος για την υλοποίηση τόσο των κατωφλίων όσο και του χρονισμού. Τα δύο πρωτόκολλα που αναφέρθηκαν έχουν καλύτερη απόδοση από το LEACH. Το TEEN είναι λίγο, μόνο καλύτερο σε εξοικονόμηση ενέργειας και επιμήκυνση του χρόνου ζώης του δικτύου από το APTEEN.

Υπάρχουν και άλλα πρωτόκολλα ιεραρχικής δρομολόγησης τα οποία παρουσιάζουν κάποιο ενδιαφέρον. Κάποια από αυτά συνοψίζονται ευθύς αμέσως. Το πρωτόκολλο Small Minimum Energy Communication Network (MECN), υπολογίζει κάθε φορά ένα υποδίκτυο ενεργειακά οικονομικό, μέσα στο οποίο είναι εγγυημένα όλα τα μονοπάτια ελαχίστου κόστους και οι μεταδόσεις μέσα από αυτό είναι σχεδόν πάντα πιο οικονομικές από τυχόν μεταδόσεις με χρήση κόμβων και συνδέσεων που να μην ανήκουν σε αυτό. Το Self Organizing Protocol (SOP) είναι ένα πρωτόκολλο που αυτο-οργανώνεται. Είναι ένα πρωτόκολλο που μπορεί να χρησιμοποιηθεί σε ετερογενή δίκτυα, δηλαδή δίκτυα με διαφορετικά είδη κόμβων. Υπάρχουν κινητοί και στατικοί κόμβοι οι οποίοι αποτελούν τη ραχοκοκκαλιά του δικτύου. Το πρωτόκολλο αυτό έχει ως βασικό ελάττωμα την ανάγκη για σχηματισμό μιας πολύ δυσμετάβλητης ιεραρχίας. Ένα ακόμα πρωτόκολλο είναι το πρωτόκολλο Sensor Aggregates Routing, το οποίο, βασικά, δημιουργεί και συντηρεί συναθροίσεις δεδομένων. Η κεντρική ιδέα είναι να παρακολουθείται αθροιστικά η κίνηση του στόχου στο δεδομένο περιβάλλον, η οποία αναφέρεται σε εφαρμογές παρακολούθησης στόχων. Η δρομολόγηση αρχιτεκτονικής εικονικού πλέγματος (Virtual Grid Architecture routing – VGA) είναι ένα πρωτόκολλο που χρησιμοποιεί συνάθροιση δεδομένων και ενδοδικτυακή επεξεργασία για επιμήκυνση του χρόνου ζωής του δικτύου. Οι κόμβοι του δικτύου διαχωρίζονται σε αισθητήριους κόμβους, σε τοπικούς συναθροιστές και κύριους συναθροιστές. Έτσι, τα δεδομένα συναθροίζονται σταδιακά. Αρχικά, στους τοπικούς συναθροιστές, στη συνέχεια στους κύριους και τελικά αποστέλλονται στον κόμβο-βάση. Στην ιεραρχική ενεργειακά ενήμερη δρομολόγηση Hierarchical Power-aware Routing (HPAR), οι κόμβοι χωρίζονται σε ομάδες ανάλογα με τη γεωγραφική εγγύτητα. Κάθε ομάδα ονομάζεται ζώνη και θεωρείται μια οντότητα. Κάθε ζώνη επιλέγει να δρομολογήσει ιεραρχικά τα πακέτα της από το μονοπάτι ελάχιστης κατανάλωση και μέγιστου αποθέματος (max-min path). Έτσι, αποφεύγει μονοπάτια που ναι μεν έχουν πολλή ενέργεια, όμως, το κόστος επικοινωνίας είναι μεγάλο.

Πίνακας 2.2: Σύγκριση Ιεραρχικής και Επίπεδης Δρομολόγησης

Ιεραρχική δρομολόγηση	**Επίπεδη δρομολόγηση**
Χρονοδρομολόγηση με προτεραιότητα	Χρονοδρομολόγηση ανάλογα με την κίνηση του δικτύου
Αποφυγή συγκρούσεων	΄Υπαρξη συγκρούσεων
Συνάθροιση δεδομένων με χρήση ομάδων κόμβων	Κόμβος σε multihop μονοπάτι συναθροίζει δεδομένα γειτόνων
Απλή αλλά όχι βέλτιστη δρομολόγηση	Εφικτή η βέλτιστη δρομολόγηση με επιπλέον πολυπλοκότητα
Απαίτηση για καθολικό και τοπικό συγχρονισμό	Πραγματοποίηση συνδέσεων κατά την

	εκτέλεση χωρίς ανάγκη συγχρονισμού
Ισοκατανεμημένη κατανάλωση ενέργειας	Κατανάλωση ενέργειας ανάλογα με την κίνηση του δικτύου
Μοιρασμένη χρήση του καναλιού επικοινωνίας	Η δίκαιη χρήση του καναλιού δεν είναι εγγυημένη

2.1.3 Πρωτόκολλα βασισμένα στην τοποθεσία

Μια τρίτη κατηγορία πρωτοκόλλων δρομολόγησης είναι τα πρωτόκολλα που βασίζονται στη θέση. Σ'αυτά τα πρωτόκολλα, οι κόμβοι επικοινωνούν με χρήση των στοιχείων της θέσης τους. Η απόσταση μεταξύ δύο κόμβων μπορεί να υπολογιστεί από την ένταση του σήματος που εκπέμπει ο ένας στον άλλο και οι σχετικές θέσεις μπορούν να υπολογιστούν με ανταλλαγή αντίστοιχων πληροφοριών από διάφορους γειτονικούς κόμβους. Οι πληροφορίες αυτές μπορεί εναλλακτικά να είναι διαθέσιμες μέσω δορυφόρου με χρήση GPS εάν αυτό είναι εφικτό από τον εξοπλισμό των κόμβων. Για εξοικονόμηση ενέργειας υπάρχει και η κατάσταση ύπνωσης των κόμβων στην οποία περιέρχονται εάν δεν υπάρχει καθόλου δραστηριότητα.

Τα πρωτόκολλα αυτά δεν παρουσιάζουν τόσο μεγάλο ενδιαφέρον, όσο τα προηγούμενα, αφού είναι πιο εξειδικευμένα και απαιτούν ειδικό εξοπλισμό. Υπάρχουν πέντε βασικές ομάδες πρωτοκόλλων που βασίζονται στις θέσεις των κόμβων. Ο αλγόριθμος Geographic Adaptive Fidelity (GAF) είναι ένας αλγόριθμος (που σκοπεύει σε εξοικονόμηση ενέργειας) σχεδιασμένος κυρίως για ad hoc κινητά δίκτυα που, όμως, μπορεί να εφαρμοστεί και σε δίκτυα αισθητήρων. Το πρωτόκολλο Geographic and Energy Aware Routing (GEAR) χρησιμοποιεί πληροφορία για την ενέργεια και τις γεωγραφικές θέσεις των κόμβων ώστε να δρομολογηθεί ευριστικά το εκάστοτε πακέτο διαμέσου του δικτύου προς τον προορισμό. Άλλα πρωτόκολλα είναι το MFR (Most Forward within Radius), DIR (Distance Routing), GEDIR (GEographic DIstance Routing), GOAFR (Greedy Other Adaptive Face Routing) και SPAN.

2.2 Πρωτόκολλα βασισμένα στη λειτουργία του δικτύου

2.2.1 Πρωτόκολλα δρομολόγησης πολλαπλών μονοπατιών (Multipath routing protocols)

Σε αυτήν την ενότητα θα αναφερθούμε στα πρωτόκολλα δρομολόγησης που χρησιμοποιούν πολλαπλά μονοπάτια αντί για ένα, με σκοπό να βελτιώσουν την απόδοση του δικτύου. Η ανοχή σφάλματος (προσαρμοστικότητα) ενός πρωτοκόλλου μετριέται από την πιθανότητα να υπάρχει ένα εναλλακτικό μονοπάτι μεταξύ της πηγής και του προορισμού, όταν το πρώτο αποτύχει. Αυτό, μπορεί να αυξηθεί διατηρώντας πολλαπλά μονοπάτια μεταξύ των πηγών και των προορισμών με τίμημα την αύξηση της κατανάλωσης ενέργειας και της αύξησης της κίνησης μέσα στο δίκτυο. Τα μονοπάτια διατηρούνται ζωντανά με χρήση περιοδικών μηνυμάτων. Συνεπώς, η αξιοπιστία του δικτύου μπορεί να αυξηθεί με το αντίστοιχο αυξημένο κόστος της συντήρησης πολλαπλών μονοπατιών.

Στο αρθρο [20] οι συγγραφείς προτείνουν έναν αλγόριθμο ο οποίος δρομολογεί τα δεδομένα διαμέσου μονοπατιών με τη μεγαλύτερη εναπομένουσα ενέργεια. Το μονοπάτι αλλάζει κάθε φορά που ένα καλύτερο μονοπάτι ανακαλύπτεται. Το αρχικό μονοπάτι θα συνεχίσει να χρησιμοποιείται έως ότου η εναπομένουσα ενέργεια μειωθεί τόσο, ώστε να είναι λιγότερη από την εναπομένουσα ενέργεια του νέου μονοπατιού, οπότε και το νέο μονοπάτι αρχίζει να χρησιμοποιείται. Με αυτή τη λογική οι κόμβοι του

αρχικού μονοπατιού δε θα εξανεμίσουν το ενεργειακό τους απόθεμα, χρησιμοποιώντας συνεχώς το ίδιο μονοπάτι, επιμηκύνοντας έτσι το χρόνο ζωής τους. Στο άρθρο αυτό δε συζητείται το κόστος αλλαγής μονοπατιού.

Στο άρθρο [21], προτείνεται η χρήση ενός συνόλου υπο-βέλτιστων μονοπατιών, περιστασιακά, με σκοπό την αύξηση του χρόνου ζωής του δικτύου. Αυτά τα μονοπάτια επιλέγονται με βάση μια πιθανότητα που εξαρτάται από το πόσο χαμηλή είναι η κατανάλωση ενέργειας σε κάθε μονοπάτι.

Συχνά, το μονοπάτι με το μεγαλύτερο απόθεμα ενέργειας είναι και το μονοπάτι με τη μεγαλύτερη κατανάλωση ενέργειας. Άρα, υπάρχει μια ανταλλαγή μεταξύ της μείωσης της συνολικής ενέργειας που καταναλώνεται και της εναπομένουσας ενέργειας του δικτύου. Οι συγγραφείς του [19] προτείνουν έναν αλγόριθμο στον οποίο συμμετέχουν στην απόφαση του βέλτιστου μονοπατιού και τα δύο κριτήρια, δηλαδή, αφ'ενός, το απόθεμα και, αφ'ετέρου, η κατανάλωση ενέργειας.

Στο άρθρο [22] η δρομολόγηση πολλαπλών μονοπατιών χρησιμοποιήθηκε για να βελτιωθεί η αξιοπιστία των Α.Δ.Α.. Η ιδέα που προτείνεται στο άρθρο αυτό είναι χρήσιμη για την παράδοση δεδομένων σε μη αξιόπιστα περιβάλλοντα. Είναι γνωστό ότι η αξιοπιστία ενός δικτύου μπορεί να αυξηθεί παρέχοντας παραπάνω από ένα μονοπάτι από την πηγή προς τον προορισμό και αποστέλλοντας το πακέτο πάνω από όλα αυτά τα μονοπάτια. Με αυτήν την τεχνική όμως θα αυξηθεί σημαντικά και η κίνηση στο δίκτυο. Συνεπώς, υπάρχει μια ανταλλαγή μεταξύ της αξιοπιστίας και της κίνησης του δικτύου. Μια ιδέα, λοιπόν, είναι το σπάσιμο του πακέτου σε υποπακέτα και, στη συνέχεια, η αποστολή του κάθε υποπακέτου με χρήση κάποιων από τα διαθέσιμα μονοπάτια. Έχει βρεθεί ότι ακόμα και αν χαθούν κάποια πακέτα το μήνυμα μπορεί να ανακατασκευαστεί με τη χρήση κάποιας συνάρτησης πλεονασμού. Ένα άλλο σημαντικό αποτέλεσμα είναι ότι για συγκεκριμένη πιθανότητα αποτυχίας ενός κόμβου, η αύξηση του πλήθους των μονοπατιών για την αποστολή των υποπακέτων μπορεί να οδηγήσει σε αύξηση της συνολικής πιθανότητας αποτυχίας.

Η κατευθυνόμενη διάχυση είναι ένας ισχυρός υποψήφιος για στιβαρή δρομολόγηση πολλαπλών μονοπατιών και παράδοση των δεδομένων με χρήση αυτής της δρομολόγησης. Έχει βρεθεί ότι η χρήση δρομολόγησης πολλαπλών μονοπατιών παρέχει μια βιώσιμη εναλλακτική λύση για ενεργειακά συμφέρουσα ανάκαμψη από αποτυχίες μέσα σε ένα Α.Δ.Α. Η ιδέα είναι ότι επειδή τα εναλλακτικά μονοπάτια θα είναι συγγενικά με το βασικό μονοπάτι το κόστος συντήρησής τους θα είναι συγκρίσιμο με αυτό της συντήρησης του βασικού μονοπατιού.

2.2.2 Δρομολόγηση βασισμένη σε επερώτηση (Query based routing)

Σε αυτό το είδος δρομολόγησης οι κόμβοι-προορισμοί διαδίδουν μια επερώτηση για δεδομένα (δηλαδή για μετρήσεις) από κάποιον κόμβο μέσα στο δίκτυο, και ο κόμβος ο οποίος είναι εφοδιασμένος με τα δεδομένα στέλνει όσα από αυτά ταιριάζουν με την επερώτηση στον κόμβο που διέδωσε την επερώτηση προηγουμένως. Συνήθως, αυτές οι επερωτήσεις περιγράφονται σε φυσική γλώσσα ή σε κάποια γλώσσα επερωτήσεων υψηλού επιπέδου. Η κατευθυνόμενη διάχυση, που περιγράφηκε νωρίτερα, είναι ένα τέτοιο είδος δρομολόγησης. Στην κατευθυνόμενη διάχυση ο κόμβος-βάση στέλνει μηνύματα ενδιαφέροντος στους αισθητήριους κόμβους. Καθώς τα ενδιαφέροντα διαδίδονται μέσα στο δίκτυο οι πηγές στέλνουν τα δεδομένα από το μονοπάτι που διαμορφώθηκε από τα ενδιαφέροντα. Με σκοπό τη μείωση της ενεργειακής κατανάλωσης, κατά τη δρομολόγηση επιτελείται συνάθροιση των δεδομένων.

Η δρομολόγηση διάδοσης χρησιμοποιεί ένα σύνολο από πράκτορες που παραμένουν στο δίκτυο για μεγάλο χρονικό διάστημα με σκοπό τη δημιουργία μονοπατιών που

κατευθύνονται προς τα γεγονότα που συμβαίνουν. Όταν ένας πράκτορας βρεθεί σε ένα μονοπάτι που οδηγεί σε ένα γεγονός, το οποίο δεν έχει ακόμη καταγραφεί, τότε δημιουργεί ένα μονοπάτι, το οποίο οδηγεί στο γεγονός αυτό, και όταν ένας πράκτορας βρεθεί σε ένα κοντύτερο και πιο αποτελεσματικό μονοπάτι τότε ενημερώνει τα τρέχοντα μονοπάτια αναλόγως. Κάθε κόμβος διατηρεί έναν πίνακα από γειτονικούς κόμβους και από γεγονότα που ενημερώνεται κάθε φορά που καταγράφονται γεγονότα. Κάθε πράκτορας φέρει από έναν πίνακα γεγονότων που ενημερώνεται από τον αντίστοιχο πίνακα του εκάστοτε κόμβου που επισκέπτεται. Κάθε κόμβος μπορεί να δημιουργήσει έναν πράκτορα με πιθανοθεωρητικές διαδικασίες. Οι πράκτορες έχουν συγκεκριμένο χρόνο ζωής που μετριέται σε ασύρματες μεταδόσεις (single hops) μεταξύ κόμβων μετά από τον οποίο πεθαίνουν. Ένας κόμβος δεν πρόκειται να δημιουργήσει μια επερώτηση παρά μόνον εάν μάθει ένα μονοπάτι προς το γεγονός αυτό. Εάν δεν υπάρχει κάποιο διαθέσιμο μονοπάτι τότε η επερώτηση στέλνεται προς τυχαία κατεύθυνση. Στη συνέχεια, ο κόμβος περιμένει να μάθει εάν η επερώτηση έφτασε στον προορισμό για ένα συγκεκριμένο χρονικό διάστημα, μετά από το οποίο ο κόμβος στέλνει καθολικά την επερώτηση προς όλο το δίκτυο.

2.2.3 Δρομολόγηση βασισμένη σε διαπραγμάτευση (Negotiation based routing protocols)

Τα πρωτόκολλα που υλοποιούν δρομολόγηση βασισμένη σε διαπραγμάτευση χρησιμοποιούν περιγραφείς δεδομένων υψηλού επιπέδου με σκοπό την ελαχιστοποίηση αποστολών πλεονασματικών δεδομένων μέσα από διαπραγμάτευση. Αποφάσεις για την επικοινωνία μεταξύ κόμβων λαμβάνονται, επίσης, με βάση τους διαθέσιμους πόρους. Πρωτόκολλα που κάνουν τέτοιου είδους δρομολόγηση είναι τα πρωτόκολλα της οικογένειας SPIN. Το κίνητρο της δημιουργίας τέτοιων πρωτοκόλλων είναι ότι η υπερχείλιση του δικτύου με σκοπό τη διασπορά των δεδομένων θα οδηγήσει στην παραλαβή πολλαπλών αντιτύπων των δεδομένων στους προορισμούς, κάτι που θα οδηγήσει σε μεγαλύτερη κατανάλωση για μετάδοση και επεξεργασία των δεδομένων σε πολλούς κόμβους. Έτσι, η βασική ιδέα της δρομολόγησης με διαπραγμάτευση είναι η αποφυγή πολλαπλών αποστολών πλεονασματικών δεδομένων με χρήση μια σειράς μηνυμάτων διαπραγμάτευσης πριν από την αποστολή των πραγματικών δεδομένων.

2.2.4 Δρομολόγηση βασισμένη στην ποιότητα υπηρεσιών (QoS-based routing)

Σε πρωτόκολλα δρομολόγησης βασισμένα στην ποιότητα υπηρεσιών το δίκτυο πρέπει να ισορροπήσει μεταξύ κατανάλωσης ενέργειας και ποιότητας των δεδομένων. Πιο συγκεκριμένα το δίκτυο πρέπει να ικανοποιήσει συγκεκριμένες μετρικές ποιότητας υπηρεσιών όπως καθυστέρηση, ενέργεια, εύρος ζώνης και άλλα όταν παραδίδονται δεδομένα στον κόμβο-βάση.

Παράγοντες που συχνά παίζουν ρόλο σε τέτοια πρωτόκολλα είναι το απόθεμα της ενέργειας, η ποιότητα υπηρεσιών σε κάθε μονοπάτι, και η προτεραιότητα του κάθε πακέτου. Συχνά, χρησιμοποιείται η λογική της δρομολόγησης πολλαπλών μονοπατιών, ώστε να αποφευχθεί η γενική αποτυχία λόγω αποτυχίας ενός μόνο κόμβου. Πρωτόκολλα που λειτουργούν προς αυτήν την κατεύθυνση είναι, λόγου χάρη, το πρωτόκολλο SAR (Sequential Assignment Routing) και το πρωτόκολλο SPEED.

2.2.5 Coherent and non-coherent protocols

Η επεξεργασία δεδομένων είναι μια σημαντική συνιστώσα στη λειτουργία των Α.Δ.Α. Συνεπώς, οι τεχνικές δρομολόγησης χρησιμοποιούν διάφορες τεχνικές επεξεργασίας

δεδομένων. Γενικά, οι κόμβοι ενός τέτοιου δικτύου συνεργάζονται μεταξύ τους για να επεξεργαστούν διάφορα δεδομένα που υπερχειλίζουν την περιοχή του δικτύου. Οι τεχνικές επεξεργασίας δεδομένων χωρίζονται σε δύο βασικές κατηγορίες, σε coherent (συμφωνούσα) και non-coherent (μη-συμφωνούσα) δρομολόγηση βασισμένη στην επεξεργασία δεδομένων. Σε non-coherent δρομολόγηση οι κόμβοι επεξεργάζονται τοπικά τα αρχικά δεδομένα πριν τα στείλουν σε άλλους κόμβους για πρόσθετη επεξεργασία. Οι κόμβοι αυτοί που επιτελούν την επιπλέον επεξεργασία καλούνται και συναθροιστές. Σε coherent δρομολόγηση τα δεδομένα προωθούνται προς τους συναθροιστές με τη λιγότερη δυνατή επεξεργασία, όπως time-stamping, απάλειψη διπλοτύπων κ.α. Για να επιτευχθεί ενεργειακά αποτελεσματική δρομολόγηση χρησιμοποιείται συχνά η coherent δρομολόγηση. Οι non-coherent εφαρμογές προκαλούν σχετικά χαμηλή κίνηση στο δίκτυο, ενώ στις coherent εφαρμογές που παράγουν επιμήκεις ροές δεδομένων, η εξοικονόμηση ενέργειας μπορεί να επιτευχθεί με βάση την επιλογή του κατάλληλου μονοπατιού.

Επίλογος

Μια παρατήρηση που απορρέει από τη μελέτη του συνόλου των ανωτέρω πρωτοκόλλων είναι ότι υπάρχουν πολλά υβριδικά πρωτόκολλα. Αυτά τα πρωτόκολλα ταιριάζουν σε πολλές από τις προαναφερθείσες κατηγορίες. Στον Πίνακα 2.3 συνοψίζεται πώς τα διαφορετικά πρωτόκολλα ταιριάζουν με τις διάφορες κατηγοριοποιήσεις και, επιπλέον, αποτυπώνεται μια σύγκριση των πρωτοκόλλων αυτών μεταξύ τους σε σχέση με διάφορες μετρικές.

Πίνακας 2.3: Συνοπτική περιγραφή των ανωτέρω πρωτοκόλλων

	Classification	Mobility	Position Awareness	Power Usage	Negotiation based	Data Aggregation	Localization	QoS	State Complexity	Scalability	Multipath	Query based
SPIN	Flat	Possible	No	Limited	Yes	Yes	No	No	Low	Limited	Yes	Yes
Directed Diffusion	Flat	Limited	No	Limited	Yes	Yes	Yes	No	Low	Limited	Yes	Yes
Rumour Routing	Flat	Very Limited	No	N/A	No	Yes	No	No	Low	Good	No	Yes
GBR	Flat	Limited	No	N/A	No	Yes	No	No	Low	Limited	No	Yes
MCFA	Flat	No	No	Limited	No	No	No	No	Low	Good	No	No
CADR	Flat	No	No	Limited	No	Yes	No	No	Low	Limited	No	No
COUGAR	Flat	No	No	N/A	No	Yes	No	No	Low	Limited	No	Yes
ACQUIRE	Flat	Limited	No	N/A	No	Yes	No	No	Low	Limited	No	Yes
EAR	Flat	Limited	No	Maximum	No	No	Yes	No	Low	Limited	No	Yes
LEACH	Hierarchical	Fixed BS	No	Maximum	No	Yes	Yes	No	CHs	Good	No	No
TEEN & APTEEN	Hierarchical	Fixed BS	No	Maximum	No	Yes	Yes	No	CHs	Good	No	No
PEGASIS	Hierarchical	Fixed BS	No	Maximum	No	No	No	No	Low	Good	No	No
MECN & SMECN	Hierarchical	No	No	N/A	No	No	No	No	Low	Low	No	No
SOR	Hierarchical	No	No	N/A	No	No	No	No	Low	Low	No	No
HPAR	Hierarchical	No	No	N/A	No	No	Yes	No	Low	Good	No	No
VGA	Hierarchical	No	No	N/A	Yes	Yes	No	No	CHs	Good	Yes	No
Sensor aggregate	Hierarchical	Limited	No	Limited	No	Yes	No	No	Low	Good	No	Possible
TTDD	Hierarchical	Yes	Yes	Limited	No	No	No	No	Moderate	Low	Possible	Possible
GAF	Location	Limited	No	Limited	No	No	No	No	Low	Good	No	No
GEAR	Location	Limited	No	Limited	No	No	No	No	Low	Limited	No	No
SPAN	Location	Limited	No	N/A	Yes	No	No	No	Low	Limited	No	No
MFR, GEDIR	Location	No	No	N/A	No	No	No	No	Low	Limited	No	No
GOAFR	Location	No	No	N/A	No	No	No	No	Low	Good	No	No
SAR	QoS	No	No	N/A	Yes	Yes	No	Yes	Moderate	Limited	No	Yes
SPEED	QoS	No	No	N/A	No	No	No	Yes	Moderate	Limited	No	Yes

ΚΕΦΑΛΑΙΟ 3

ΠΛΑΤΦΟΡΜΕΣ ΑΣΥΡΜΑΤΩΝ ΔΙΚΤΥΩΝ ΑΙΣΘΗΤΗΡΩΝ

3.1 Εισαγωγή

Οι παραδοσιακές προγραμματιστικές τεχνολογίες βασίζονται στο λειτουργικό σύστημα το οποίο παρέχει ένα αφηρημένο μοντέλο το οποίο θα είναι υπεύθυνο για την επεξεργασία, την είσοδο και έξοδο (I/O), τη σύνδεση μέσω δικτύου και την αλληλεπίδραση του χρήστη με το υλικό. Οταν ένα τέτοιο μοντέλο πρόκειται να εφαρμοστεί σε προγραμματιζόμενα ενσωματωμένα συστήματα, όπως ένα δίκτυο αισθητήρων, ο προγραμματιστής της εφαρμογής είναι εκείνος που πρέπει να διαχειριστεί το πέρασμα μηνυμάτων, το συγχρονισμό των events, το χειρισμό των διακοπών και τις μετρήσεις. Αυτό έχει ως αποτέλεσμα, μια εφαρμογή να υλοποιείται ως μια μηχανή πεπερασμένων καταστάσεων (Finite State Machine – FSM) η οποία καλύπτει όλες τις περιπτώσεις, όπως την ύπαρξη μη αξιόπιστων καναλιών επικοινωνίας, τις μεγάλες καθυστερήσεις, την ακανόνιστη άφιξη μηνυμάτων, τα ταυτόχρονα γεγονότα κ.ά.

Για ενσωματωμένα συστήματα με περιορισμένους πόρους και απαιτήσεις απόκρισης σε πραγματικό χρόνο, χρησιμοποιούνται διάφοροι μηχανισμοί (MicroKernel τεχνολογίες και scheduler πραγματικού χρόνου) στο λειτουργικό σύστημα, ώστε να μειωθεί το μέγεθος του κώδικα, να βελτιωθεί ο χρόνος απόκρισης και να μειωθεί η κατανάλωση ενέργειας. Η εκτέλεση με βάση events επιτρέπει στο σύστημα να ενεργοποιεί μια κατάσταση ύπνου (sleep mode), χαμηλής κατανάλωσης, όταν δεν απαιτείται επεξεργαστία events.

Τα ενσωματωμένα λειτουργικά συστήματα έχουν την τάση να δίνουν στον προγραμματιστή μεγαλύτερες προσβάσεις σε επίπεδο υλικού, ώστε να μπορεί να χειρίζεται τους οδηγούς συσκευών, τους αλγόριθμους δρομολόγησης και να μπορεί να βελτιώσει τον κώδικα σε επίπεδο assembly. Οι παραπάνω τεχνικές εφαρμόζονται με επιτυχία σε μικρά και “ιδανικά” ενσωματωμένα συστήματα, αλλά αποτυγχάνουν σε περιπτώσεις μεγάλων δικτύων για δύο λόγους. Αρχικά, γιατί τα αποτελέσματα που επιθυμούμε προέρχονται από ένα μεγάλο αριθμό κατανεμημένων κόμβων και οι κατανεμημένοι αλγόριθμοι είναι δύσκολο να υλοποιηθούν εξαιτίας των περιορισμών (μνήμης, ενέργειας, εύρους ζώνης κ.ά.). Επιπλέον, ένας κόμβος πρέπει να μπορεί να αντιδρά σε πολλά ταυτόχρονα ερεθίσματα και σε ταχύτητες όπως αυτές της αλλαγής του φαινομένου που παρακολουθεί.

Υπάρχουν δύο τύποι προγραμματισμού δικτύων αισθητήρων, αυτοί που εκτελούνται από τους τελικούς χρήστες και αυτοί που διεκπεραιώνονται από όσους αναπτύσσουν εφαρμογές. Ένας τελικός χρήστης βλέπει το δίκτυο αισθητήρων ως μια δεξαμενή με δεδομένα και αλληλεπιδρά με το δίκτυο μέσω επερωτήσεων. Μια γλώσσα επερωτήσεων για δίκτυα αισθητήρων πρέπει να είναι εκφραστική, ώστε να κωδικοποιεί τη λογική κάθε εφαρμογής σε υψηλό επίπεδο αφαίρεσης και ταυτόχρονα, να είναι έτσι δομημένη, ώστε να επιτρέπει αποτελεσματική εκτέλεση σε κατανεμημένες πλατφόρμες. Όσοι αναπτύσσουν εφαρμογές πρέπει να προσφέρουν στους τελικούς χρήστες δυνατότητες όπως η παροχή δεδομένων, η επεξεργασία και η αποθήκευση. Το εύρος γνώσεων που εμπλέκεται σε αυτό τον τομέα είναι μεγάλο. Αυτοί που αναπτύσσουν εφαρμογές, είναι συνήθως ειδικοί στην επεξεργασία σήματος και πληροφορίας, αλλά όχι στα λειτουργικά συστήματα και τα δίκτυα, γι’ αυτό και αποτελεί πρόκληση η δημιουργία του κατάλληλου αφηρημένου προγραμματιστικού μοντέλου που θα παρέχει το υπόβαθρο για περαιτέρω ανάπτυξη εφαρμογών.

Πριν αναφερθούμε στις πλατφόρμες λογισμικού σε επίπεδο κόμβου και πιο αναλυτικά στο TinyOS που χρησιμοποιείται, σε συνδυασμό με τον προσομοιωτή του, το TOSSIM, στην εργασία αυτή, είναι σημαντικό να αναφέρουμε κάποια στοιχεία σχετικά με πλαφόρμες υλικού αισθητήρων σε επίπεδο κόμβου. Αρχικά, παρατίθενται οι τρεις κατηγορίες του υλικού αισθητήριων κομβων. Στη συνέχεια, από την πιο δημοφιλή οικογένεια ενσωματωμένων αισθητήριων κόμβων, αυτή του Berkeley, επιλέγεται ένας αντιπρόσωπος, ο MICA, του οποίου παραθέτουμε, εν συνεχεία, τα τεχνικά χαρακτηριστικά.

3.2 Αισθητήριοι κόμβοι σε επίπεδο υλικού

Πριν ασχοληθούμε με θέματα που αφορούν το λογισμικό για δίκτυα αισθητήρων, είναι σημαντικό να αναφερθούμε στις πιο αντιπροσωπευτικές πλατφόρμες υλικού (hardware platforms) για αισθητήριους κόμβους. Το υλικό των αισθητήριων κόμβων μπορεί να χωριστεί σε τρεις κατηγορίες όπου κάθε μία έχει τα δικά της πλεονεκτήματα και μειονεκτήματα.

- *Υπολογιστές γενικού σκοπού με προσθήκες (Augmented general-purpose computers)*: Τέτοια παραδείγματα περιλαμβάνουν PC με χαμηλή κατανάλωση ισχύος, ενσωματωμένα PC (π.χ. PC104), PC σχεδιασμένα κατά παραγγελία (π.χ. Sensoria WINS NG κόμβοι) και διάφορους προσωπικούς ψηφιακούς βοηθούς (Personal Digital Assistant – PDA). Σε αυτούς τους κόμβους χρησιμοποιούνται τυπικά λειτουργικά συστήματα όπως WIN CE, Linux ή λειτουργικά συστήματα πραγματικού χρόνου και χρησιμοποιούν αναγνωρισμένα πρωτόκολλα επικοινωνίας όπως Bluetooth ή IEEE 802.11. Εξαιτίας της σχετικά υψηλής υπολογιστικής ικανότητας τους, είναι σε θέση να εξυπηρετούν μια μεγάλη ποικιλία αισθητήρων μετρήσεων, από απλά μικρόφωνα έως πολύπλοκες βιντεοκάμερες. Σε σχέση με τους αφοσιωμένους αισθητήριους κόμβους (Dedicated Sensor Nodes), οι τύπου PC πλατφόρμες έχουν μεγάλες απαιτήσεις σε ενέργεια. Όταν όμως το θέμα της ενέργειας δεν είναι τόσο σημαντικό, αυτές οι πλατφόρμες πλεονεκτούν, αφού μπορούν να χρησιμοποιήσουν διαθέσιμα πρωτόκολλα δικτύων, δημοφιλείς γλώσσες προγραμματισμού, middleware και άλλο υπάρχον λογισμικό.
- *Αφοσιωμένοι ενσωματωμένοι αισθητήριοι κόμβοι (Dedicated embedded sensor nodes)*: Τέτοια παραδείγματα περιλαμβάνουν την οικογένεια αισθητήρων του Berkeley, την οικογένεια αισθητήρων Medusa του UCLA, τους κόμβους Ember και τους μAMP του MIT. Οι πλατφόρμες αυτές συνήθως χρησιμοποιούν εμπορικής παραγωγής (commercial off-the-shelf - COTS) σύνολα chip τα οποία δίνουν έμφαση στον παράγοντα “μικρό μέγεθος”, στην επεξεργασία και επικοινωνία με μικρή κατανάλωση ισχύος και στην απλή αλληλεπίδραση των κόμβων. Χάρη στις COTS CPU τους, οι πλατφόρμες αυτές είναι σε θέση να υποστηρίζουν τουλάχιστον μια προγραμματιστική γλώσσα όπως η C. Ακόμα, για να διατηρήσουν το χώρο που καταλαμβάνουν τα προγράμματα μικρό, ώστε να χωρά στην περιορισμένη μνήμη τους, επιτρέπουν στους προγραμματιστές αυτών των πλατφορμών πλήρη άδεια πρόσβασης στο υλικό, αλλά ελάχιστη υποστήριξη σε επίπεδο λειτουργικού συστήματος. Ένα κλασικό παράδειγμα τέτοιας πλατφόρμας είναι το TinyOS και η προγραμματιστική γλώσσα nesC, που το συνοδεύει. Το TinyOS είναι αυτό με το οποίο θα ασχοληθούμε στη συνέχεια.
- *System-on-chip (SoC) κόμβοι*: Παραδείγματα SoC υλικού είναι οι smart dust, ο BWRC picoradio κόμβος και ο PASTA κόμβος. Οι σχεδιαστές αυτών των πλατφορμών προσπαθούν να εξωθήσουν το υλικό στα όρια του, επαναπροσδιορίζοντας τα οφέλη και τις ζημιές για τους κόμβους σε επίπεδο

σχεδίασης του chip. Ο σκοπός είναι να βρεθούν νέοι τρόποι ολοκλήρωσης για τεχνολογίες CMOS, MEMS και RF, ώστε να κατασκευαστούν κόμβοι με εξαιρετικά μικρή κατανάλωση ισχύος και μικρές διαστάσεις, που, όμως, να μπορούν να παρέχουν τις ίδιες ή και καλύτερες δυνατότητες από τις ήδη υπάρχουσες. Οι πλατφόρμες υλικού αυτές βρίσκονται ακόμα σε ερευνητικό στάδιο και γι' αυτό δεν υπάρχουν διαθέσιμες πλατφόρμες λογισμικού για αυτές.

Από τις παραπάνω πλατφόρμες υλικού, οι κόμβοι του Berkeley, λόγω του μικρού τους μεγέθους, της open-source ανάπτυξης λογισμικού και της διαθεσιμότητάς τους στο εμπόριο, έχουν γίνει πολύ δημοφιλείς στην ερευνητική κοινότητα.

3.2.1 Berkeley motes

Οι κόμβοι του Berkeley είναι μια οικογένεια ενσωματωμένων αισθητήριων κόμβων που έχουν την ίδια αρχιτεκτονική. Στον Πίνακα 3.1 που ακολουθεί, παρουσιάζεται μια σύγκριση διάφορων τύπων κόμβων του Berkeley.

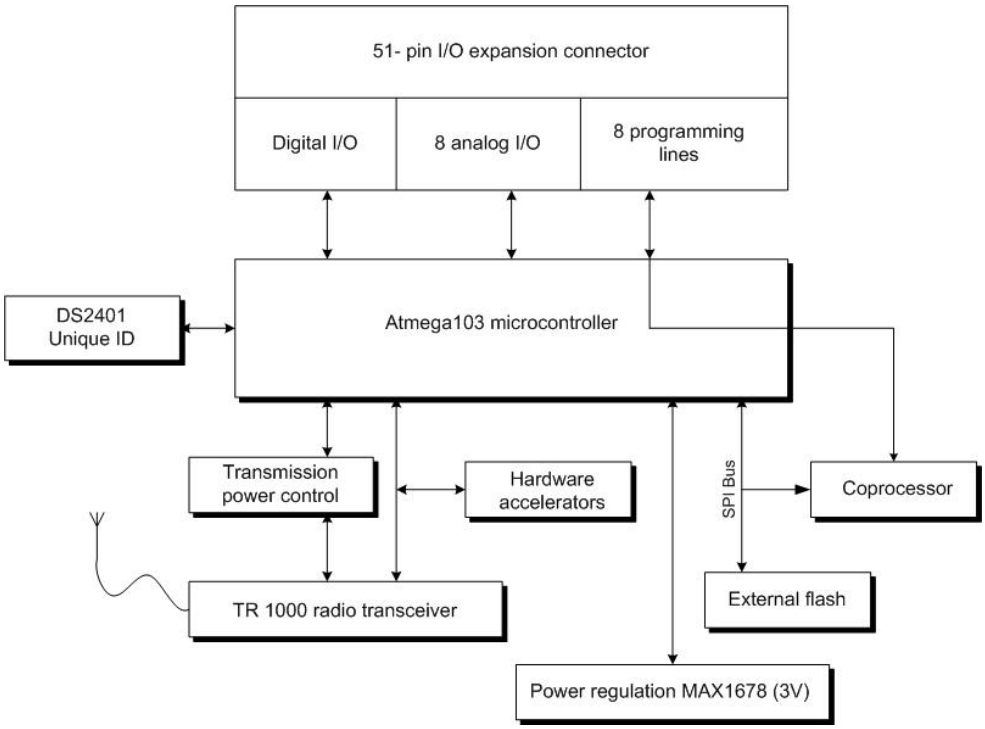
Πίνακας 3.1: Σύγκριση κόμβων του Berkeley.

Mote Type Year	*WeC* 1998	*René* 1999	*René 2* 2000	*Dot* 2000	*Mica* 2001	*Mica2Dot* 2002	*Mica 2* 2002	*Telos* 2004
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			60
RAM (KB)	0.5		1		4			2
Active Power (mW)	15		15		8		33	3
Sleep Power (μW)	45		45		75		75	6
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M24M01S
Connection type	I^2C				SPI			I^2C
Size (KB)	32				512			128
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7			2.7		1.8
Total Active Power (mW)	24				27	44	89	41
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	10-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes

Στη συνέχεια, θα αναφέρουμε κάποια χαρακτηριστικά του κόμβου MICA. Οι κόμβοι MICA διαθέτουν δύο CPU, όπως φαίνεται στο Σχήμα 3.1. Ο κύριος microcontroller (MCU), ένας Atmel Atmega103L, αναλαμβάνει τη συνήθη επεξεργασία. Ένας ξεχωριστός και λιγότερο ικανός επεξεργαστής είναι ενεργός μόνο όταν ο MCU επαναπρογραμματίζεται. Ο Atmega103L MCU διαθέτει 512 KB μνήμης τύπου flash και 4 KB μνήμη δεδομένων. Εξαιτίας του τόσο μικρού μεγέθους μνήμης, το γράψιμο κώδικα για κόμβους αποτελεί μια πρόκληση και απαιτεί πολλές φορές βελτίωση σε επίπεδο assembly.

Εκτός από τη μνήμη στον MCU ο MICA έχει μια επιπλέον μνήμη flash 512 KB για να αποθηκεύονται δεδομένα. Ο MCU συνδέεται με αυτή την εξωτερική μνήμη μέσω ενός χαμηλής ταχύτητας περιφερειακού πρωτοκόλλου αλληλεπίδρασης (Serial Peripheral Interface – SPI) γι' αυτό και στη μνήμη αυτή αποθηκεύονται δεδομένα για μετέπειτα

επεξεργασία και όχι προγράμματα. Η RF επικοινωνία χρησιμοποιεί το TR1000 chip set που λειτουργεί στη ζώνη των 916 MHz. Ο MICA υλοποιεί 40 kbps ρυθμό μετάδοσης και το μεγαλύτερο εύρος μετάδοσης είναι περίπου 300 πόδια σε ανοιχτό χώρο.



Σχήμα 3.1: Αρχιτεκτονική του κόμβου MICA.

Ο MICA διαθέτει ένα 51 pin I/O ζεύκτη, ο οποίος συνδέει μετρητές, μηχανισμούς κίνησης και παράλληλους και σειριακούς I/O boards. Η σειριακή I/O (UART) σύνδεση επιτρέπει την επικοινωνία μεταξύ κόμβου και υπολογιστή σε πραγματικό χρόνο.

Όσον αφορά την κατανάλωση ενέργειας, οι ραδιομεταδόσεις είναι αυτές που καταναλώνουν την μεγαλύτερη ισχύ (Πίνακας 3.2). Κάθε πακέτο (π.χ. 30 bytes) χρειάζεται μόνο 4 ms για να σταλεί, αλλά η λήψη εισερχόμενων μηνυμάτων απαιτεί ο δέκτης να είναι ανοικτός συνέχεια. Η ενέργεια που χρειάζεται για να σταλεί ένα πακέτο ενισχύει το δέκτη μόνο για 27 ms. Ακόμη, πολύ σημαντικές είναι οι διαφορές στην κατανάλωση ενέργειας από την ενεργή, την αδρανή και την περίπτωση αναστολής της λειτουργίας του MCU. Εξοικονόμηση ενέργειας μπορούμε να πετύχουμε με αναστολή της λειτουργίας του MCU και του RF δέκτη, για όσο το δυνατό μεγαλύτερο χρονικό διάστημα.

3.3 Πλατφόρμες λογισμικού σε επίπεδο κόμβου

Οι περισσότερες μεθοδολογίες σχεδίασης για λογισμικό δικτύων αισθητήρων είναι κομβοκεντρικές (node-centric), και σε αυτές, οι προγραμματιστές, πρέπει να σκέφτονται έχοντας υποψη τους πως θα έπρεπε να συμπεριφέρεται ο κόμβος στο περιβάλλον. Μια πλατφόρμα σε επίπεδο κόμβου μπορεί να είναι:

- Ένα κομβοκεντρικό (node-centric) λειτουργικό σύστημα, το οποίο παρέχει στους προγραμματιστές του αφαίρεση για τους κόμβους σε υλικό και σε θέματα που αφορούν το δίκτυο.

- Πλατφόρμα γλώσσας (language platform), η οποία παρέχει στους προγραμματιστές της μια βιβλιοθήκη “εργαλείων”.

Πίνακας 3.2: Κατανάλωση ενέργειας κόμβων MICA

Component	Rate	Startup time	Current consumption
MCU active	4 MHz	N/A	5.5 mA
MCU idle	4 MHz	1 μs	1.6 mA
MCU suspend	32 KHz	4 ms	<20 μA
Radio transmit	40 KHz	30 ms	12 mA
Radio receive	40 KHz	30 ms	1.8 mA
Photoresister	2000 Hz	10 ms	1.235 mA
Accelerometer	100 Hz	10 ms	5 mA/axis
Temperature	2 Hz	500 ms	0.150 mA

Ένα τυπικό λειτουργικό σύστημα διαχωρίζεται από την πλατφόρμα υλικού παρέχοντας ένα σύνολο υπηρεσιών για εφαρμογές, οι οποίες περιλαμβάνουν διαχείριση αρχείων, κατανομή μνήμης, προγραμματισμό εργασιών, οδηγούς για περιφερειακές συσκευές και κάλυψη δικτύου. Τα λειτουργικά συστήματα ενσωματωμένων συστημάτων, εξαιτίας των εξαιρετικά εξειδικευμένων εφαρμογών τους και των περιορισμένων πόρων, είναι υποχρεωμένα να κάνουν διάφορες ανταλλαγές όταν παρέχουν υπηρεσίες. Για παράδειγμα, αν δε χρειάζεται διαχείριση αρχείων, τότε είναι λογικό να μη χρειάζεται να παρέχεται file system. Αν δεν υπάρχει δυναμική κατανομή μνήμης, τότε το σύστημα διαχείρισης μνήμης απλοποιείται σε μεγάλο βαθμό. Αν η προτεραιότητα εκτέλεσης εργασιών είναι σημαντική, τότε μπορεί να προστεθεί ένας πιο εξελιγμένος μηχανισμός προγραμματισμού των προτεραιοτήτων.

Το TinyOS είναι ένα από τα πιο αντιπροσωπευτικά παραδείγματα κομβοκεντρικών εργαλείων προγραμματισμού. Στη συνέχεια θα δοθούν πολλά στοιχεία για τη λειτουργία του και τα χαρακτηριστικά του. Το TinyOS είναι στενά συνδεδεμένο με την προγραμματιστική γλώσσα nesC και αυτό έχει ως αποτέλεσμα πολλά στοιχεία που θα παρατεθούν εν συνεχεία, να αφορούν άμεσα και τη nesC και κυρίως το σχεδιαστικό και προγραμματιστικό της κομμάτι.

3.3.1 TinyOS

Το TinyOS είναι ένα μικρό (λιγότερο από 400 bytes), ευέλικτο λειτουργικό σύστημα το οποίο έχει δημιουργηθεί από ένα σύνολο επαναχρησιμοποιήσιμων components τα οποία συναθροίζονται σε ένα ειδικό ανά εφαρμογή σύστημα. Το TinyOS υποστηρίζει ένα event-driven, ταυτόχρονο μοντέλο το οποίο βασίζεται σε interfaces χωριστών φάσεων, ασύγχρονα γεγονότα και στα tasks τα οποία είναι υπολογισμοί, των οποίων η επεξεργασία μπορεί να αναβληθεί. Το TinyOS έχει υλοποιηθεί στη γλώσσα nesC, η οποία υποστηρίζει τα components και το ταυτόχρονο μοντέλο του TinyOS, όπως και εκτεταμένες cross-component βελτιώσεις και εντοπισμό συνθηκών ανταγωνισμού κατά

τη μεταγλώττιση (compile-time). Το TinyOS αποτελεί καινοτομία τόσο στα συστήματα δικτύων αισθητήρων όσο και σε ένα μεγάλο φάσμα εφαρμογών.

Το TinyOS δεν είναι ένα λειτουργικό σύστημα με την παραδοσιακή έννοια. Είναι ένα προγραμματιστικό πλαίσιο για ενσωματωμένα συστήματα και για ένα σύνολο components το οποίο επιτρέπει να χτίζεται ειδικό λειτουργικό σύστημα σε κάθε εφαρμογή. Μια τυπική εφαρμογή έχει μέγεθος περίπου 15 Kbytes όπου το βασικό λειτουργικό σύστημα είναι τα 400 bytes. Η μεγαλύτερη εφαρμογή, ένα σύστημα επερωτήσεων όπως μια βάση δεδομένων, έχει μέγεθος 64 Kbytes. Το TinyOS έχει ως σκοπό να υποστηρίξει εφαρμογές δικτύων αισθητήρων σε πλατφόρμες με περιορισμένους πόρους, όπως οι κόμβοι τύπου Berkeley.

Οι τέσσερεις βασικές ανάγκες που υποκίνησαν τη δημιουργία του TinyOS είναι οι εξής:

1) Περιορισμένοι πόροι (Limited Resources): Οι κόμβοι έχουν περιορισμένους φυσικούς πόρους για να καλύψουν τις απαιτήσεις του μικρού μεγέθους, του χαμηλού κόστους και της μικρής κατανάλωσης ενέργειας.
2) Αναδραστικός ταυτοχρονισμός (Reactive Concurrency): Ένας τυπικός κόμβος ενός δικτύου αισθητήρων πρέπει να είναι σε θέση να ανταποκρίνεται σε πολλά γεγονότα, όπως διαχείριση ραδιοεπικοινωνιών, σε πραγματικό χρόνο.
3) Ευελιξία (Flexibility): Λόγω της ποικιλίας σε υλικό και σε εφαρμογές απαιτείται ένα ευέλικτο λειτουργικό σύστημα, το οποίο ανάλογα με την εφαρμογή πρέπει να μπορεί να μειώνει το χρόνο και την ισχύ και να ανεξαρτητοποιεί τα όρια ανάμεσα στο υλικό και το λογισμικό.
4) Χαμηλή ισχύς (Low Power): Η χαμηλή κατανάλωση ισχύος αποτελεί σημαντικό στόχο στο σχεδιασμό ενός κόμβου. Το TinyOS πρεπει να λειτουργεί με χαμηλή ισχύ, αλλά και να είναι ευέλικτο στη διαχείριση ενέργειας και στις στρατηγικές κύκλου εργασίας.

Γενικά

Ένα πρόγραμμα σε TinyOS είναι ένας γράφος από components, το καθένα από τα οποία είναι μια ανεξάρτητη υπολογιστική οντότητα, που παρουσιάζει ένα ή περισσότερα interfaces. Τα components διαθέτουν τρεις αφαιρετικούς τρόπους υπολογισμού: τις commands, τα events και τα tasks. Οι commands και τα events είναι μηχανισμοί για επικοινωνία μεταξύ components, ενώ τα tasks χρησιμοποιούνται για να εκφράσουν ταυτοχρονισμό μέσα στα components.

Μια command είναι, τυπικά, μια αίτηση προς ένα component για να διεκπεραιώσει μια υπηρεσία π.χ. την αρχικοποίηση των μετρήσεων για ένα αισθητήρα. Από την άλλη, ένα event σηματοδοτεί την ολοκλήρωση της υπηρεσίας. Τα events μπορεί να σηματοδοτούνται ασύγχρονα, για παράδειγμα, εξαιτίας κάποιου hardware interrupt ή της άφιξης ενός μηνύματος. Από την οπτική ενός παραδοσιακού λειτουργικού συστήματος, οι commands είναι ανάλογες των downcalls και τα events των upcalls. Οι commands και τα events δεν μπορούν να μπλοκαριστούν. Μια αίτηση για μια υπηρεσία είναι χωριστής φάσης (split phase) δηλαδή η αίτηση για την υπηρεσία (η command) και το σήμα ολοκλήρωσης (το αντίστοιχο event) δεν αποτελούν ζευγάρι. Η command επιστρέφει αμέσως, ενώ το event σηματοδοτεί την ολοκλήρωση αργότερα.

Αντί να εκτελείται ένας υπολογισμός αμέσως, οι commands και events handlers μπορούν να “αναρτήσουν” (post) ένα task. Το task είναι μια συνάρτηση η οποία θα εκτελεστεί αργότερα από τον scheduler του TinyOS. Αυτό επιτρέπει στις commands και στα events να ανταποκρίνονται άμεσα και να επιστρέφουν αμέσως, ενώ εκτεταμένοι υπολογισμοί έχουν ανατεθεί στα tasks. Όσο τα tasks εκτελούν σημαντικούς

υπολογισμούς, το βασικό μοντέλο εκτέλεσής τους “τρέχει” συνεχώς μέχρι να ολοκληρωθεί αντί να εκτελείται αόριστα. Αυτό κάνει και τα tasks λιγότερο απαιτητικά σε σχέση με τα νήματα (threads). Τα tasks αναπαριστούν τον εσωτερικό ταυτοχρονισμό μέσα σε ένα component και η δικαιοδοσία τους περιορίζεται στο component αυτό. Ο συνηθισμένος scheduler για tasks του TinyOS χρησιμοποιεί non-preemptive FIFO πολιτική.

Πίνακας 3.3: Βασικά interfaces που παρέχει toTinyOS.

Interface	Description
ADC	Sensor hardware interface
EEPROMRead/Write	Hardware clock
HardwareId	EEPROM read and write
I2C	Hardware ID access
Leds	Red/yellow/green LEDs
MAC	Radio MAC layer
Mic	Microphone interface
Pot	Hardware potentiometer for transmit power
Random	Random number generator
ReceiveMsg	Receive Active Message
SendMsg	Send Active Message
StdControl	Init, start and stop components
Time	Get current time
TinySec	Lightweight encryption/decryption
WatchDog	WatchDog timer control

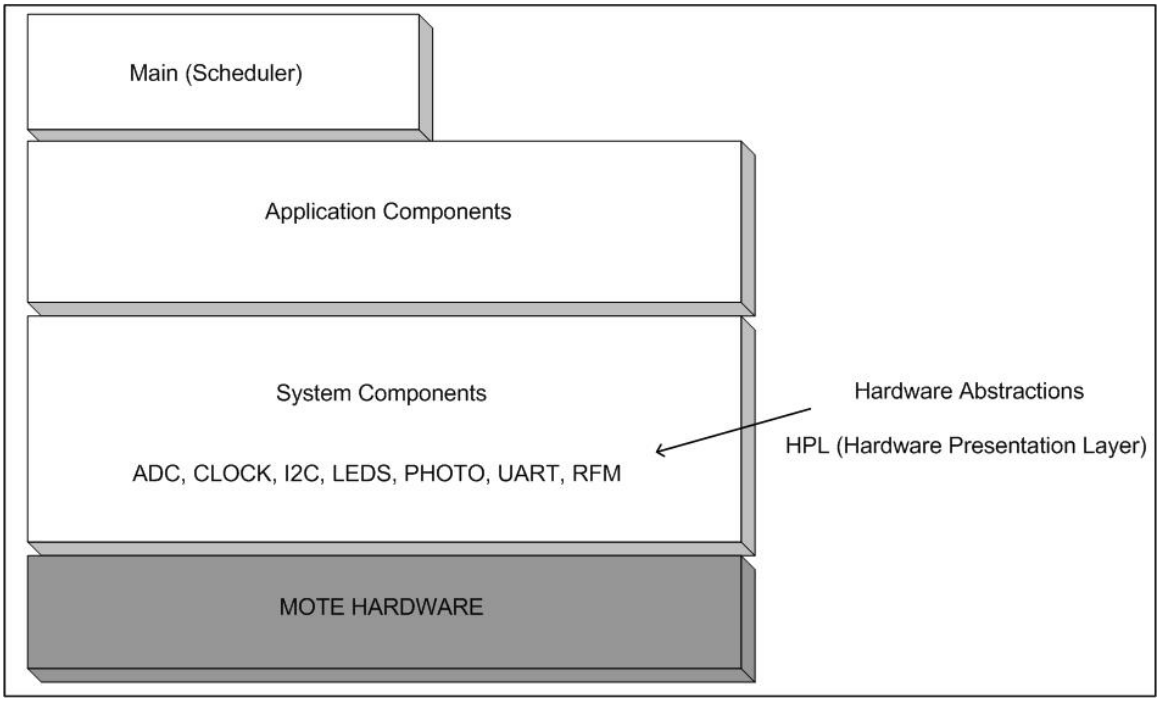
Το TinyOS συμπτύσσει όλους τους υλικούς πόρους σε components. Για παράδειγμα, η κλίση της getData() εντολής σε ένα component θα προκαλέσει αργότερα τη σηματοδότηση του dataReady() event όταν πυροδοτηθεί το hardware interrupt. Καθώς πολλά components βασίζονται εντελώς στο λογισμικό, ο συνδυασμός των λειτουργιών σε χωριστές φάσεις και των tasks καθιστούν αυτό το διαχωρισμό ξεκάθαρο στον προγραμματιστή. Για παράδειγμα, ας θεωρήσουμε ένα component το οποίο αποκρύπτει μια μνήμη δεδομένων. Σε μια υλοποίηση υλικού, η command θα καθοδηγήσει την απόκρυψη σε επίπεδο υλικού για να εκτελέσει την λειτουργία, ενώ μια υλοποίηση λογισμικού θα αναρτήσει ένα task για να αποκρύψει τα δεδομένα στη CPU. Και στις δύο περιπτώσεις ένα event θα σηματοδοτήσει ότι η απόκρυψη έχει ολοκληρωθεί.

Η τωρινή έκδοση του TinyOS παρέχει ένα μεγάλο αριθμό από components στους σχεδιαστές εφαρμογών, συμπεριλαμβανομένων αφαιρέσεων για αισθητήρες, single-hop κάλυψη δικτύου, ad-hoc δρομολόγηση, διαχείριση ενέργειας και σταθερή αποθήκευση. Ένας σχεδιαστής συνθέτει μια εφαρμογή με τη δημιουργία components και “καλωδιώνοντας” (wiring) τα με ήδη υπάρχοντα components του TinyOS που προσφέρουν υλοποιήσεις για απαιτούμενες υπηρεσίες.

Η δομή του TinyOS αναπαρίσταται στο παρακάτω σχήμα 3.2, όπου ξεχωρίζουν τα παρακάτω στρώματα:

- **Main component:** Περιέχει τον κώδικα που σχετίζεται με το σώμα της main(), η οποία είναι αυτή που αρχικοποιεί το υλικό, τον scheduler και ξεκινά την εκτέλεση του προγράμματος.

- **Application components:** Το επίπεδο αυτό περιέχει τα components, που ορίζονται από τις εφαρμογές για να υλοποιήσουν τις λειτουργικότητες τους.
- **System components:** Components τα οποία υλοποιούν τις υπηρεσίες που προσφέρονται στα components των εφαρμογών.
- **HPL:** Components του συστήματος τα οποία περικλείουν το υποκείμενο επίπεδο υλικού. Αυτή η ομάδα από components του συστήματος είναι η πιο εξαρτημένη πλατφόρμα. Πρέπει να υπάρχει ένα component του συστήματος το οποίο να διαχειρίζεται το υλικό αυτό (π.χ. το component CLOCK) ως κάλυμμα υλικού. Για κάθε interrupt υλικού, παράγεται ένα event. Τα events που παράγονται θα επεξεργάζονται από components του πιο πάνω επιπέδου που έχουν καλωδιωθεί με το κάλυμμα υλικού.
- **Mote hardware:** Το TinyOS υποστηρίζει πλατφόρμες και ανάλογα με τον τύπο του κόμβου, είναι οι ακόλουθες: avrmote, mica, mica128, mica2, mica2dot, pc.



Σχήμα 3.2: Η δομή του TinyOS που αποτελείται από ένα scheduler και ένα γράφο από components.

Μοντέλο component

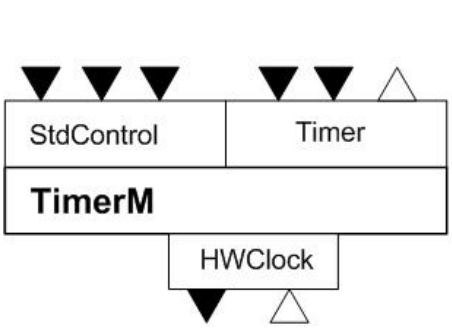
Το προγραμματιστικό μοντέλο του TinyOS, το οποίο παρέχεται από τη γλώσσα προγραμματισμού nesC, επικεντρώνεται γύρω από την έννοια του component. Τα components συμπυκνώνουν ένα συγκεκριμένο σύνολο από υπηρεσίες, που καθορίζονται από τα interfaces. Το TinyOS αποτελείται από ένα σύνολο από επαναχρησιμοποιήσιμα components συστήματος σε συνδυασμό με ένα task scheduler. Μια εφαρμογή συνδέει components χρησιμοποιώντας μια προδιαγραφή καλωδίωσης (wiring specification) η οποία είναι ανεξάρτητη από την υλοποίηση των components. Η προδιαγραφή καλωδίωσης αυτή που καθορίζει το σύνολο των components που χρησιμοποιεί μια εφαρμογή.

Ο compiler ελαχιστοποιεί το μειονέκτημα των μικρών, στοιχειωδών component με ανάλυση και απλοποίηση όλου του προγράμματος (εφαρμογής και λειτουργικού

συστήματος). Μη χρησιμοποιήσιμα components δεν περιλαμβάνονται στο εκτελέσιμο της εφαρμογής.

Ένα component έχει δύο τύπους interface. Αυτά που παρέχουν (provides) και αυτά που χρησιμοποιούν (uses). Αυτά τα interfaces ορίζουν πώς ένα component αλληλεπιδρά με άλλα components. Ένα interface, γενικά, μοντελοποιεί κάποια υπηρεσία (π.χ. αποστολή μηνυμάτων) και καθορίζεται λεπτομερώς από τον τύπο του interface (interface type). Ένα απλό παράδειγμα (Σχήμα 3.3) είναι η μορφή του component *TimerM*, το οποίο αποτελεί τμήμα της υπηρεσίας χρονομέτρησης του TinyOS, και παρέχει τα interfaces *StdControl* και *Timer* και χρησιμοποιεί το *Clock*. Ένα component μπορεί να χρησιμοποιεί και να παρέχει το ίδιο interface πολλές φορές, αρκεί κάθε περίπτωσή του να έχει διαφορετικό όνομα.

Τα interfaces είναι διπλής κατεύθυνσης (bidirectional) και περιέχουν τόσο commands όσο και events. Μια command είναι μια συνάρτηση η οποία υλοποιείται από τους παρόχους ενός interface, ενώ ένα event είναι μια συνάρτηση που υλοποιείται από τους χρήστες. Για παράδειγμα, το interface *Timer* (Σχήμα 3.4) ορίζει τις commands *start* και *stop* και ένα event το *fired*. Παρόλο, που η αλληλεπίδραση μεταξύ μετρητή και του πελάτη του θα μπορούσε να είχε εξασφαλιστεί μέσω δύο διαφορετικών interfaces (ένα για τις commands και ένα για τα events), η ομαδοποίηση τους στο ίδιο interface κάνει τον προσδιορισμό τους πιο καθαρό και συντελεί στην αποφυγή bugs όταν γίνεται η καλωδίωση μεταξύ components.




```
Module TimerM {
        provides {
                interface StdControl;
                interface Timer[uint8_t id];
        }
        uses interface Clock;
}
implementation {
  … a dialect of C …
}
```

Σχήμα 3.3: Προδιαγραφές και γραφική απεικόνιση του component *TimerM*. Τα παρεχόμενα interfaces απεινονίζονται πάνω από το component *TimerM* και τα χρησιμοποιούμενα interfaces από κάτω. Τα βέλη προς τα κάτω απεικονίζουν *commands* και τα βέλη προς τα πάνω *events*.

Η nesC έχει δύο τύπους components τα *modules* και τις *configurations*. Τα modules παρέχουν κώδικα και είναι γραμμένα σε μια διάλεκτο της C με επεκτάσεις για κλίσεις (calling) και υλοποίηση (implementing) commands και events. Ένα module δηλώνει ιδιωτικές (private) μεταβλητές και ενδιάμεση μνήμη δεδομένων (data buffers), στα οποία μπορεί να αναφερθεί. Οι configurations χρησιμοποιούνται για να καλωδιώσουν components μαζί, συνδέοντας interfaces που χρησιμοποιούνται από components σε interfaces που παρέχονται από άλλα components. Η υπηρεσία χρονομέτρησης του TinyOS (Σχήμα 3.5) είναι μία configuration (*TimerC*) η οποία συνδέει το module χρονομέτρησης (*TimerM*) με το component το οποίο υλοποιεί το ρολόι σε επίπεδο υλικού (*HWClock*). Οι configurations επιτρέπουν πολλαπλά components να συναθροίζονται σε ένα απλό “supercomponent” που εκθέτει ένα απλό σύνολο από interfaces. Για παράδειγμα, η στοίβα δικτύου του TinyOS είναι μια configuration που αποτελείται από καλωδίωση 21 διαφορετικών modules και 10 υπο-components.

Κάθε component έχει το δικό του χώρο ονομάτων (namespace) για interfaces, που χρησιμοποιείται για αναφορά στις commands και στα events που χρησιμοποιεί. Όταν

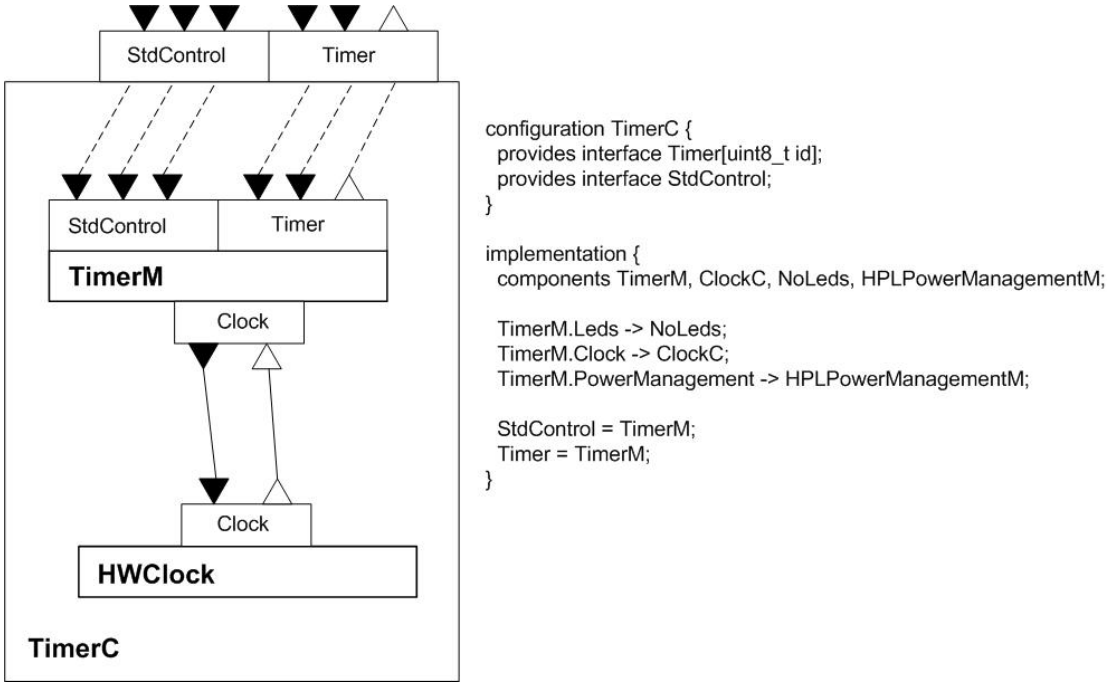
καλωδιώνονται interfaces μαζί, μία configuration δημιουργεί τη σύνδεση ανάμεσα στο τοπικό όνομα (local name) ενός interface που χρησιμοποιείται από ένα component και στο τοπικό όνομα ενός interface που παρέχεται από ένα άλλο. Έτσι, ένα component επιδρά σε ένα interface χωρίς να αναφέρεται ρητά στην υλοποίηση του. Αυτό διευκολύνει την εκτέλεση παρεμβάσεων με την εισαγωγή ενός νέου component, το οποίο χρησιμοποιεί και παρέχει τα ίδια interfaces, στο γράφο των components.

```
interface StdControl  {
  command result_t init ( );
  command result_t start ( );
  command result_t stop ( );
}

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

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

interface SendMsg
{
  command result_t send (uint16_t address,
                         uint8_t length,
                         TOS_MsgPtr msg);
  event result_t sendDone (TOS_MsgPtr msg, result_t success);
}
```

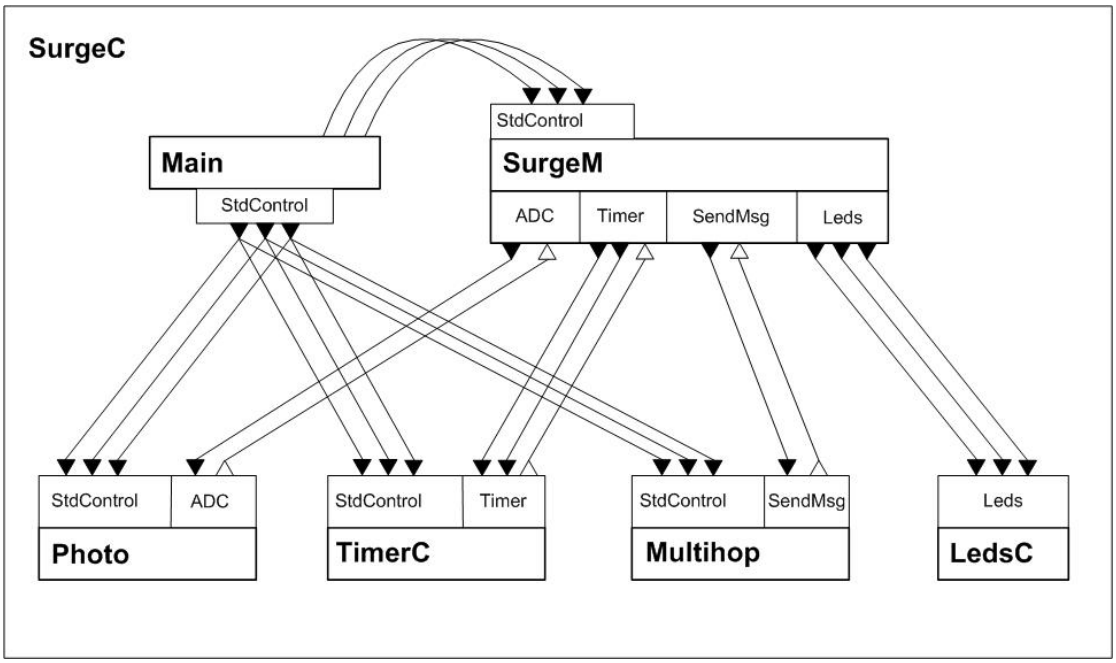
Σχήμα 3.4: Παραδείγματα από interfaces του TinyOS.

Τα interfaces μπορούν να καλωδιωθούν πολλές φορές. Αυτό το fan-out είναι διάφανο γι' αυτόν που πραγματοποιεί την κλήση. Η nesC επιτρέπει fan-out εφόσον ο τύπος που επιστρέφεται διαθέτει μια συνάρτηση για να συνδυάζει τα αποτελέσματα όλων των κλήσεων. Για παράδειγμα, ο επιστρεφόμενος τύπος *result_t* είναι ένα λογικό AND. Ακόμα, το fan-out επιστρέφει αποτυχία αν μια υποκλήση αποτύχει.

Ένα component μπορεί να παρέχει ένα παραμετροποιημένο (parameterized) interface το οποίο εξάγει πολλές περιπτώσεις του ίδιου interface, παραμετροποιημένο από έναν αναγνωριστή (identifier), συνήθως ένα μικρό ακέραιο. Ο αναγνωριστής περνάει στις commands και στα events του interface ως μια επιπλέον παράμετρος. Έτσι, το παραμετροποιημένο interface επιτρέπει σε ένα component να υλοποιεί πολλαπλά χωριστά interface, ένα για κάθε component πελάτη. Ένας πελάτης ενός παραμετροποιημένου interface πρέπει να αναφέρει τον αναγνωριστή, ως σταθερά στην καλωδιωμένη configuration. Για να αποφεύγονται τα μπερδέματα στην τιμή του αναγνωριστή, η nesC παρέχει μια ειδική λέξη κλειδί, την *unique*, η οποία επιλέγει διαφορετική τιμή αναγνωριστή για κάθε πελάτη.



Σχήμα 3.5: Υπηρεσία χρονομέτρησης του TinyOS : η *TimerC* configuration.



Σχήμα 3.6: Η top-level configuration για την εφαρμογή Surge

Κάθε εφαρμογή του TinyOS περιγράφεται από μια configuration υψηλού επιπέδου (top-level), η οποία συνδέει όλα τα component που χρησιμοποιούνται. Στο Σχήμα 3.6 φαίνεται το γράφημα μιας απλής εφαρμογής της SurgeC. Η εφαρμογή αυτή, περιοδικά (TimerC), πραγματοποιεί μετρήσεις φωτός (Photo) και τις προωθεί προς έναν κόμβο-βάση χρησιμοποιώντας multi-hop δρομολόγηση (MultiHop).

Η nesC επιβάλλει κάποιους περιορισμούς στη C για να βελτιώσει την αποτελεσματικότητα και την ευρωστία του κώδικα. Αρχικά, η γλώσσα απαγορεύει δείκτες σε συναρτήσεις, έτσι ώστε ο compiler να γνωρίζει ακριβώς το γράφο κλήσεων του προγράμματος. Αυτό επιτρέπει βελτιώσεις σε περιπτώσεις ίδιων components για ολόκληρα μονοπάτια κλήσεων. Έτσι, απομακρύνεται το κόστος από επιπλέον κλήσεις λόγω διασταυρούμενων module. Ακόμα, η γλώσσα δεν επιτρέπει την εκτεταμένη χρήση δυναμικής δέσμευσης μνήμης. Τα components δηλώνουν στατικά όλη την κατάσταση του προγράμματος, το οποίο εμποδίζει το θρυμματισμό της μνήμης και αποτυχία στη κατανομή μνήμης κατά την εκτέλεση. Οι περιορισμοί αυτοί ακούγονται πιο σημαντικο από ότι είναι στην πραγματικότητα. Η “αφαίρεση” των components ελαχιστοποιεί πολλές από τις απαιτήσεις για δυναμική δέσμευση. Στις σπάνιες περιπτώσεις, όπου πραγματικά χρειάζεται (π.χ. TinyDB), ένα component κοινής μνήμης μοιράζεται από ένα σύνολο συνεργαζόμενων components.

Μοντέλο εκτέλεσης και ταυτοχρονισμός

Το γεγονοκεντρικό πεδίο ορισμού των δικτύων αισθητήρων χρειάζεται αξιόπιστη παραλληλία. Τα events μπορεί να ληφθούν οποιαδήποτε στιγμή και πρέπει να αλληλεπιδρούν, σαφώς, με τους υπό εκτέλεση υπολογισμούς. Αυτό είναι ένα κλασικό πρόβλημα των συστημάτων και έχει δύο γενικές προσεγγίσεις: 1) εισαγωγή στην ουρά ατομικά των εργασιών που φθάνουν για να εκτελεστούν αργότερα (όπως στα περισσότερα συστήματα διαβίβασης μηνυμάτων) και 2) εκτέλεση ενός handler άμεσα, όπως συμβαίνει στις περιπτώσεις των ενεργών μηνυμάτων (active messages). Επειδή κάποια events απαιτούν άμεση εκτέλεση επιλέγεται η δεύτερη προσέγγιση. Η nesC είναι σε θέση να εντοπίζει στατικά περιπτώσεις ανταγωνισμού δεδομένων και έτσι εξαλείφεται ένα πλήθος από πιθανά πολύπλοκα bugs.

Ο πυρήνας του μοντέλου εκτέλεσης βασίζεται, αφ’ενός, στην εκτέλεση μέχρι την ολοκλήρωση (run-to-completion) των tasks, που αποτελούν τους υπο εκτέλεση υπολογισμούς, και αφ’ετέρου, στην εκτέλεση των interrupt handlers, οι οποίοι σηματοδοτούνται ασύγχρονα από το υλικό. Τα tasks είναι μια ξεκάθαρη οντότητα της γλώσσας. Ένα πρόγραμμα υποβάλλει στον scheduler ένα task για εκτέλεση χρησιμοποιώντας τον τελεστή *post*. Ο scheduler μπορεί να εκτελέσει τα tasks με οποιαδήποτε σειρά αλλά πρέπει να τηρεί τον κανόνα για εκτέλεση μέχρι την ολοκλήρωση. Ο συνηθισμένος scheduler του TinyOS ακολουθεί την πολιτική FIFO, αλλά έχουν υλοποιηθεί και άλλες πολιτικές, όπως η εκτέλεση αυτού με την μικρότερη προθεσμία.

Επειδή τα tasks δε γίνονται preempt αλλά “τρέχουν” μέχρι την ολοκλήρωσή τους, είναι ατομικά και σέβεται το ένα την εκτέλεση του άλλου. Δε συμβαίνει όμως το ίδιο με τους interrupt handlers ή τις commands και τα events που ενεργοποιούνται. Για να διευκολύνθούν οι περιπτώσεις συνθηκών ανταγωνισμού χωρίζουμε τον κώδικα σε σύγχρονο και ασύγχρονο:

- Σύγχρονος κώδικας (Synchronous code - SC): κώδικας, ο οποίος αποτελείται μόνο από tasks.
- Ασύγχρονος κώδικας (Asynchronous code - AC): κώδικας, ο οποίος αποτελειται από τουλάχιστον έναν interrupt handler.

Η προσέγγιση των παραδοσιακών λειτουργικών συστημάτων ως προς τον ασύγχρονο κώδικα έχει ως σκοπό να τον ελαχιστοποιήσει και να εμποδίσει τον κώδικα -στο επίπεδο χρήστη- να είναι ασύγχρονος. Κάτι τέτοιο θα ήταν πολύ περιοριστικό για το TinyOS. Τα components πρέπει να αλληλεπιδρούν με ένα μεγάλο εύρος υλικού σε πραγματικό χρόνο, το οποίο δεν είναι δυνατό να πραγματοποιηθεί, γενικά, με την προσέγγιση της ουράς αναμονής για την εκτέλεση. Για παράδειγμα, στη στοίβα του δικτύου υπάρχουν components που αλληλεπιδρούν με το ραδιοπομπό σε επίπεδο bit, byte και μέσω ισχυρού σήματος δέκτες. Ένας πρώτος στόχος είναι να επιτρέπεται στους σχεδιαστές να δημιουργούν ανταποκρινόμενες, ταυτόχρονες δομές δεδομένων οι οποίες θα επιτρέπουν την ασφαλή ανταλλαγή δεδομένων ανάμεσα σε σύγχρονο και ασύγχρονο κώδικα.

Παρόλο που με non-preemption ελαχιστοποιούμε τον ανταγωνισμό ανάμεσα σε tasks, υπάρχουν ακόμη πιθανές συνθήκες ανταγωνισμού ανάμεσα σε σύγχρονο και ασύγχρονο κώδικα, όπως και ανάμεσα σε ασύγχρονους κώδικες. Γενικά, οποιαδήποτε ενημέρωση κοινής καταστάσης, η οποία είναι προσβάσιμη από ασύγχρονο κώδικα είναι πιθανό να οδηγήσει σε ανταγωνισμό δεδομένων. Για να επαναφέρουμε την ατομικότητα σε τέτοιες περιπτώσεις, έχουμε δύο επιλογές: τη μετατροπή του ανταγωνιστικού κώδικα σε tasks (μόνο σύγχρονο κώδικα) και τη χρησιμοποίηση ατομικών τμήματα (atomic sections) για την ενημέρωση της κοινής επικράτειας. Ένα ατομικό τμήμα είναι μια σύντομη ακολουθία κώδικα, την οποία η nesC εξασφαλίζει πως θα εκτελεστεί ατομικά. Η τωρινή υλοποίηση απενεργοποιεί τα interrupts όταν εκτελούνται ατομικά τμήματα και φροντίζει να μην υπάρχουν σε αυτά επαναληπτικές διατάξεις. Η βασική συνθήκη που πρέπει να επιβάλλει η nesC είναι η εξής:

> **Συνθήκη χωρίς ανταγωνισμό (Race-Free Invariant)**: Οποιαδήποτε ενημέρωση σε κοινή επικράτεια είναι είτε σύγχρονος κώδικας μόνο είτε συμβαίνει σε κάποιο ατομικό τμήμα.

Ο compiler της nesC ενισχύει αυτή τη συνθήκη σε χρόνο compile, εμποδίζοντας σχεδόν όλες τις συνθήκες ανταγωνισμού. Είναι πιθανό να εισάγουμε μια συνθήκη ανταγωνισμού την οποία να μην μπορεί να εντοπίσει ο compiler, αλλά αυτό θα συμβεί αν συνδέσουμε πολλά ατομικά τμήματα ή tasks και χρησιμοποιήσουμε για αποθήκευση ενδιάμεσες μεταβλητές.

Το πρακτικό όφελος από την αποτροπή συνθηκών ανταγωνισμού δεδομένων είναι ουσιώδες. Αρχικά, εξαφανίζει ένα πλήθος από επίπονα μη-ντετερμινιστικά bugs. Ακόμη, αυτό σημαίνει ότι, κατά τη σύνθεση, μπορούμε να αγνοήσουμε τον ταυτοχρονισμό. Δεν παίζει ρόλο ποιά components δημιουργούν τον ταυτοχρονισμό ή πώς αυτά καλωδιώνονται μαζί, αφού ο compiler θα εντοπίσει κάθε παραβίαση κατά το compile. Η ανάλυση σε βάθος κατά το compile επιτρέπει να χρησιμοποιούμε ένα πλήθος από ταυτόχρονες δομές δεδομένων και στοιχειώδη συγχρονισμό. Ακόμη, υπάρχουν διάφορες παραλλαγές από ταυτόχρονες ουρές και μηχανές καταστάσεων. Έτσι, είναι εύκολη η διαχείριση, απευθείας, επειγουσών χρονικά ενεργειών ακόμα και όταν ενημερώνονται κοινοί πόροι.

Ενεργά μηνύματα (Active messages)

Ένα σημαντικό θέμα στο σχεδιασμό του TinyOS είναι η αρχιτεκτονική του δικτύου. Ο πυρήνας αφαίρεσης για επικοινωνίες του TinyOS βασίζεται στα ενεργά μηνύματα (Active Messages - AM), τα οποία είναι μικρά πακέτα (36 byte) που συνδέονται και με ένα ID handler του ενός byte. Κατά τη λήψη ενός ενεργού μηνύματος, ένας κόμβος στέλνει το μήνυμα (χρησιμοποιώντας ένα event) σε έναν ή περισσότερους handlers, οι οποίοι είναι καταχωρημένοι για να λαμβάνουν μηνύματα τέτοιου τύπου. Η καταχώρηση handler διεκπεραιώνεται χρησιμοποιώντας στατική καλωδίωση και παραμετροποιημένα interface.

Τα ενεργά μηνύματα προσφέρουν ένα μη αξιόπιστο, single-hop datagram πρωτόκολλο και παρέχουν ένα ενοποιημένο interface επικοινωνίας τόσο στους ραδιοπομπούς όσο και στις ενσωματωμένες σειριακές πύλες (για σύνδεση κόμβων όπως είναι οι σταθμοί εργασίας). Υψηλότερου επιπέδου πρωτόκολλα παρέχουν multihop επικοινωνία και άλλα γνωρίσματα χτισμένα πάνω από το AM interface. Υπάρχουν διάφορες εκδοχές της βασικής στοίβας ενεργών μηνυμάτων οι οποίες έχουν ενσωματωμένη ασφάλεια στο επίπεδο διασύνδεσης. Η event-driven φύση των ενεργών μηνυμάτων και η στενή σχέση ανάμεσα σε υπολογισμούς και επικοινωνία έχουν ως αποτέλεσμα η “αφαίρεση” να ταιριάζει καλά στο πεδίο ορισμού των δικτύων αισθητήρων.

3.3.2 nesC

Η nesC είναι μια προγραμματιστική γλώσσα για ενσωματωμένα δικτυακά συστήματα, όπως είναι οι κόμβοι. Η nesC υποστηρίζει ένα προγραμματιστικό μοντέλο που ενοποιεί τις αντιδράσεις στα ερεθίσματα του περιβάλλοντος, ταυτοχρονισμό και επικοινωνία. Με βελτιστοποιήσεις σε όλο το πρόγραμμα και εντοπισμό των συνθηκών ανταγωνισμού κατα το compile, η nesC απλοποιεί την ανάπτυξη εφαρμογών, τη μείωση του μεγέθους του κώδικα και ελαχιστοποιεί πιθανά σφάλματα. Η nesC είναι μια επέκταση της C για να υποστηρίξει και να εκφράσει το σχεδιασμό του TinyOS. Παρέχει ένα σύνολο από δομικά στοιχεία της γλώσσας και περιορισμούς για την υλοποίηση components και εφαρμογών με το TinyOS.

Ως προγραμματιστική γλώσσα, η nesC, καλείται να φέρει σε πέρας έναν αριθμό από ξεχωριστές προκλήσεις:

Καθοδήγηση από τις αλληλεπιδράσεις με το περιβάλλον: Σε αντίθεση με τους παραδοσιακούς υπολογιστές, οι κόμβοι χρησιμοποιούνται για να συλλέγουν δεδομένα και να ελέγχουν το περιβάλλον γύρω τους και όχι για διάφορους υπολογισμούς. Αυτή η επιλογή οδηγεί σε δύο παρατηρήσεις. Πρώτα απ’ όλα, οι κόμβοι είναι event-driven, αντιδρούν στις αλλαγές του περιβάλλοντος (λήψη μηνυμάτων) παρά καθοδηγούνται από αλληλεπιδράσεις ή μαζική επεξεργασία. Δεύτερον, η άφιξη event και η επεξεργασία δεδομένων είναι ταυτόχρονες ενέργειες, οπότε χρειάζονται μια ταυτόχρονη προσέγγιση διαχείρισης, ώστε να αποφευχθούν πιθανά σφάλματα όπως συνθήκες ανταγωνισμού.

Περιορισμένοι πόροι: Οι κόμβοι έχουν περιορισμένους φυσικούς πόρους για να πετύχουν μικρό μέγεθος, χαμηλό κόστος και χαμηλή κατανάλωση ενέργειας.

Αξιοπιστία: Παρόλο που αναμένουμε πως ατομικά οι κόμβοι θα αποτύχουν για λόγους που οφείλονται στο υλικό, πρέπει να μπορούμε να υποστηρίζουμε εφαρμογές με μεγάλο χρόνο ζωής. Σημαντικός σκοπός είναι να μειωθούν τα λάθη κατά την εκτέλεση, αφού ο μόνος τρόπος ανάκαμψης είναι η αυτόματη επανεκκίνηση του κόμβου.

Ήπιες απαιτήσεις σε πραγματικό χρόνο: Παρόλο που υπάρχουν κάποια tasks τα οποία εξαρτώνται σημαντικά από το χρόνο όπως, διαχείριση ραδιοεπικοινωνιών ή αποτελεσμάτων μετρήσεων, δεν υπάρχει εγγύηση για την εξυπηρέτηση σε πραγματικό

χρόνο. Οι χρονικοί περιορισμοί μπορούν να ξεπεραστούν με πλήρη έλεγχο της εφαρμογής και του λειτουργικού συστήματος και με περιορισμό χρήσης των πόρων.

Η nesC αποτελεί μια επέκταση της γλώσσας προγραμματισμού C και σχεδιάστηκε για να ενσωματώσει δομικές αρχές και το μοντέλο εκτέλεσης του TinyOS. Το TinyOS επαναϋλοποιήθηκε με βάση τη nesC. Οι βασικές ιδέες γύρω από τη nesC είναι οι εξής:

- Διαχωρισμός της κατασκευής και της σύνθεσης: τα προγράμματα σχηματίζονται από *components*, τα οποία συναρμολογούνται (καλωδιώνονται - wired) για να σχηματίσουν ένα πλήρες πρόγραμμα. Τα components ορίζουν δύο τύπους εμβέλειας, ο πρώτος για τον προσδιορισμό τους (περιέχει τα ονόματα των στιγμιοτύπων των *interfaces*) και ο δεύτερος για την υλοποίησή τους. Τα components επιτυγχάνουν εσωτερικό ταυτοχρονισμό με τη μορφή των *tasks*. Τα interfaces μπορούν να “περάσουν” νήματα ελέγχου στα components. Τα νήματα αυτά έχουν τις ρίζες τους σε κάποιο task ή σε κάποιο interrupt υλικού.
- Προσδιορισμός της συμπεριφοράς του component σε σχέση με το σύνολο των interfaces: Τα interfaces μπορεί να παρέχονται (provided) είτε να χρησιμοποιούνται (used) από ένα component. Τα παρεχόμενα interfaces αναπαριστούν τη λειτουργικότητα με την οποία εφοδιάζεται ο χρήστης τους. Τα χρησιμοποιήσιμα interfaces αναπαριστούν τη λειτουργικότητα που χρειάζεται το component για να φέρει σε πέρας την εργασία του.
- Τα interfaces είναι διπλής κατεύθυνσης (bidirectional): Τα interfaces καθορίζουν ένα σύνολο από συναρτήσεις που πρέπει να υλοποιηθούν από αυτόν που παρέχει το interface (*commands*) και ένα άλλο σύνολο που πρέπει να υλοποιηθεί από το χρήστη του interface (*events*). Έτσι, ένα απλό interface αναπαριστά μια πολύπλοκη αλληλεπίδραση ανάμεσα σε components (π.χ. εκδήλωση ενδιαφέροντος για κάποιο event και ενημέρωση όταν πραγματοποιηθεί το event αυτό). Αυτό είναι πολύ σημαντικό αφού οι μεγάλες commands στο TinyOS (π.χ. αποστολή πακέτων) είναι non-blocking: η ολοκλήρωση τους σηματοδοτείται από κάποιο event (sendDone). Με τον προσδιορισμό των interfaces, ένα component δεν μπορεί να καλέσει την command send, αν προηγουμένος δεν παρέχει υλοποίηση για το event sendDone. Γενικά, οι commands καλούνται προς τα κάτω (από components εφαρμογών στα components που είναι πιο κοντά στο υλικό), ενώ τα events προς τα πάνω. Κάποια στοιχειώδη events δεσμεύονται από interrupts υλικού (το είδος της δέσμευσης εξαρτάται από το σύστημα).
- Τα components συνδέονται στατικά μεταξύ τους μέσω των interfaces τους: Αυτό αυξάνει την αποτελεσματικότητα κατά την εκτέλεση, ενθαρρύνει τη γερή σχεδίαση και επιτρέπει την καλύτερη στατική ανάλυση των προγραμμάτων.
- Η nesC σχεδιάστηκε με την προσδοκία πως ο κώδικας θα παράγεται από whole-program compilers. Αυτό, επιτρέπει την καλύτερη παραγωγή και ανάλυση κώδικα. Ένα παράδειγμα, είναι ο εντοπιστής συνθηκών ανταγωνισμου σε compile-time.
- Το μοντέλο ταυτοχρονισμού της nesC βασίζεται στην εκτέλεση μέχρι την ολοκλήρωση των tasks και στους interrupt handlers οι οποίοι μπορούν να διακόπτουν τα tasks και ο ένας τον άλλο. Ο compiler της nesC εντοπίζει τις πιθανές συνθήκες ανταγωνισμού εξαιτίας των interrupt handlers.

Σχεδίαση σε nesC

Μερικές βασικές αρχές είναι οι εξής:

Η nesC είναι μια επέκταση της C: Η C παράγει αποτελεσματικό κώδικα για όλους τους μικρό-controllers που μπορεί να χρησιμοποιηθούν σε δίκτυα αισθητήρων. Ακόμα, παρέχει όλα τα χαμηλού επιπέδου γνωρίσματα για πρόσβαση στο υλικό και η αλληλεπίδραση με ήδη υπάρχοντα κώδικα C είναι απλοποιημένη. Επιπλέον, σημαντικό είναι πως πολλοί προγραμματιστές γνωρίζουν C. Η C έχει κάποια σημαντικά μειονεκτήματα: παρέχει λίγη βοήθεια ώστε να γραφτεί ασφαλής κώδικας ή δομημένες εφαρμογές. Η nesC παρέχει ασφάλεια μειώνοντας την εκφραστική δύναμη και δομή με χρήση των components. Κανένα από τα νέα γνωρίσματα της nesC δε συνδέεται στενά με τη C. Οι ίδιες ιδέες μπορούν να προστεθούν σε προστακτικές γλώσσες προγραμματισμού όπως η Modula-2.

Ανάλυση όλου του προγράμματος: Τα προγράμματα σε nesC υπόκεινται σε ανάλυση στο σύνολό τους (για ασφάλεια) και σε βελτιστοποιήσεις (για απόδοση). Το περιορισμένο μέγεθος προγράμματος που χρησιμοποιούμε στους κόμβους καθιστά αυτή την προσέγγιση πολύ βολική.

Η nesC είναι στατική γλώσσα: Δεν υπάρχει η δυνατότητα για δυναμική δέσμευση μνήμης και ο γράφος των κλήσεων είναι πλήρως γνωστός κατά το compile. Αυτοί οι περιορισμοί καθιστούν την ανάλυση όλου του προγράμματος και τις βελτιστοποιήσεις πιο απλές και πιο ακριβείς. Οι περιορισμοί αυτοί ακούγονται πιο επαχθείς από ότι είναι στην πραγματικότητα. Το μοντέλο με τα components και τα παραμετροποιημένα interfaces μειώνουν σημαντικά την ανάγκη για δυναμική δέσμευση μνήμης και δυναμική αναφορά. Ως τώρα έχουν υλοποιηθεί μια βελτιστοποίηση και μια ανάλυση: ένας απλός inliner για ολόκληρο το πρόγραμμα και ένας εντοπιστής συνθηκών ανταγωνισμού.

Η nesC υποστηρίζει και αντανακλά το σχεδιασμό του TinyOS: Η nesC βασίζεται στην έννοια των components και υποστηρίζει ευθέως το event-based μοντέλο ταυτοχρονισμού του TinyOS. Επιπλέον, η nesC διεκπεραιώνει και το θέμα της ταυτόχρονης πρόσβασης σε διαμοιραζόμενα δεδομένα. Στην πράξη, η nesC διαλευκαίνει πολλές αμφιβολίες ως προς την έννοια component και την έννοια ταυτοχρονισμός του TinyOS.

3.4 Προσομοιωτές σε επίπεδο κόμβου

3.4.1 Γενικά

Οι μεθοδολογίες σχεδίασης σε επίπεδο κόμβου σχετίζονται συνήθως με προσομοιωτές της συμπεριφοράς ενός δικτύου αισθητήρων ανά κόμβο. Με χρήση προσομοίωσης, οι σχεδιαστές μπορούν γρήγορα να μελετήσουν την απόδοση (σε όρους χρόνου, ισχύος, εύρους ζώνης και κλιμάκωσης) πιθανών αλγορίθμων χωρίς να τους υλοποιήσουν σε υλικό και χωρίς να απασχολούνται με τα αληθινά φυσικά φαινόμενα.

Ένας τυπικός προσομοιωτής σε επίπεδο κόμβου έχει τα ακόλουθα components:

- *Μοντέλο αισθητήριων κόμβων (Sensor node model)*: Ένας κόμβος σε προσομοιωτή ενεργεί ως πλατφόρμα εκτέλεσης λογισμικού, λήπτης μετρήσεων και τερματικό επικοινωνίας. Για τη διευκόλυνση των σχεδιαστών, ώστε να ασχοληθούν μόνο με τον κώδικα της εφαρμογής, ένα μοντέλο κόμβου συνήθως παρέχει ή προσομοιώνει μια στοίβα πρωτοκόλλου επικοινωνίας, τη συμπεριφορά

των αισθητήρων (π.χ. μέτρηση θορύβου) και υπηρεσίες λειτουργικού συστήματος. Αν οι κόμβοι είναι κινητοί, τότε η θέση και οι ιδιότητες του μηχανισμού κίνησης πρέπει να μοντελοποιηθούν. Αν η ενέργεια αποτελεί μέρος της σχεδίασης, τότε και η κατανάλωση ενέργειας πρέπει να μοντελοποιηθεί.

- *Μοντέλο επικοινωνίας (Communication model)*: Ανάλογα με τις λεπτομέρειες της μοντελοποίησης, η επικοινωνία μπορεί να χωρίζεται σε διαφορετικά επίπεδα. Οι πιο εξελιγμένοι προσομοιωτές μοντελοποιούν την επικοινωνία σε φυσικό επίπεδο, προσομοιώνοντας την RF καθυστέρηση διάδοσης και τις συγκρούσεις ταυτόχρονων μεταδόσεων. Εναλλακτικά, η επικοινωνία μπορεί να προσομοιωθεί σε επίπεδο MAC ή δικτύου, χρησιμοποιώντας στοχαστικές διαδικασίες για την αναπαράσταση χαμηλού επιπέδου συμπεριφοράς.
- *Μοντέλο φυσικών φαινομένων (Physical environment model)*: Ένα βασικό στοιχείο, που αφορά το περιβάλλον στο οποίο λειτουργεί ένα δίκτυο αισθητήρων, είναι τα φυσικά φαινόμενα στα οποία αναφέρεται. Το περιβάλλον μπορεί να προσομοιωθεί σε διάφορα επίπεδα λεπτομερειών. Αν το δίκτυο είναι παθητικό (δεν επιδρά αυτό στη συμπεριφορά του περιβάλλοντος), τότε το περιβάλλον μπορεί να προσομοιωθεί ξεχωριστά. Αν όμως εκτός από τις μετρήσεις, το δίκτυο πραγματοποιεί ενέργειες που επηρεάζουν τη συμπεριφορά του περιβάλλοντος, τότε απαιτείται ένας πιο αυστηρά ολοκληρωμένος μηχανισμός προσομοίωσης.
- *Στατιστικά και οπτικοποίηση (Statistics and visualization)*: Τα αποτελέσματα των προσομοιώσεων πρέπει να συλλέγονται για ανάλυση. Δεδομένου ότι ο στόχος της προσομοίωσης είναι να εξάγει καθολικά χαρακτηριστικά από την εκτέλεση ανεξάρτητων κόμβων, είναι πολύ σημαντική η οπτικοποίηση της καθολικής συμπεριφοράς. Ένα ιδανικό εργαλείο οπτικοποίησης πρέπει να επιτρέπει στους χρήστες να παρατηρούν εύκολα την κατανομή και την κινητικότητα των κόμβων, τη συνεκτικότητα ανάμεσα στους κόμβους, την ποιότητα διασύνδεσης, τις end-to-end διαδρομές επικοινωνίας και τις καθυστερήσεις, τα φαινόμενα και τη χωροχρονική δυναμική τους, τις μετρήσεις ανά κόμβο, την κατάσταση ανά κόμβο και τις παραμέτρους της διάρκειας ζωής του κόμβου (π.χ. ισχύ μπαταρίας).

Ένας προσομοιωτής δικτύου αισθητήρων προσομοιώνει τη συμπεριφορά ενός υποσυνόλου αισθητήριων κόμβων λαμβάνοντας υπόψη το χρόνο. Ανάλογα με το πώς εξελίσσεται ο χρόνος, υπάρχουν δυο μοντέλα εκτέλεσης: η cycle-driven (CD) προσομοίωση και η discrete-event (DE) προσομοίωση. Η CD προσομοίωση διαχωρίζει τη συνεχή έννοια του πραγματικού χρόνου σε ticks (χτύπους) του ρολογιού και προσομοιώνει τη συμπεριφορά του συστήματος με αυτά τα ticks. Σε κάθε tick, αρχικά, προσομοιώνονται τα φυσικά φαινόμενα και στη συνέχεια ελέγχονται όλοι οι κόμβοι για το αν έχουν να μετρήσουν, να επεξεργαστούν ή να επικοινωνήσουν. Οι μετρήσεις και οι υπολογισμοί υποτίθεται πως ολοκληρώνονται πριν το επόμενο tick. Η αποστολή ενός πακέτου υποτίθεται πως ολοκληρώνεται και αυτή μέχρι τότε, αλλά το πακέτο δε θα είναι διαθέσιμο στον κόμβο προορισμού μέχρι το επόμενο tick.

Αντίθετα με τους CD προσομοιωτές, οι DE υποθέτουν ότι ο χρόνος είναι συνεχής και ότι ένα event μπορεί να συμβεί οποιαδήποτε στιγμή. Ένα event είναι ένα ζευγάρι μιας τιμής και ενός χρονικού σημείου που υποδηλώνει πότε πρέπει να χειριστούμε το event. Τα components στους DE προσομοιωτές αντιδρούν στα εισερχόμενα events και παράγουν εξερχόμενα events. Ένας DE προσομοιωτής χρησιμοποιεί μια καθολική ουρά, όπου όλα τα events που ανταλλασσονται ανάμεσα σε κόμβους ή modules εισάγονται στην ουρά για events, σύμφωνα με τη χρονολογική τους σειρά.

Αξιολογώντας τη συμπεριφορά με βάση το χρόνο, οι DE προσομοιωτές είναι πιο ακριβείς και ως αποτέλεσμα τρέχουν πιο αργά. Το επιπλέον κόστος για την ταξιμόμηση όλων των events και των υπολογισμών υπερισχύει στο χρόνο υπολογισμού. Σε ένα

πρώιμο στάδιο σχεδίασης όταν ενδιαφερόμαστε περισσότερο για την ασυμπτωτική συμπεριφορά και όχι για τις ιδιότητες της χρονομέτρησης, οι CD προσομοιωτές χρειάζονται λιγότερο πολύπλοκα components και εξάγουν γρηγορότερα τα αποτελέσματα. Δεν υπάρχουν CD προσομοιωτές οι οποίοι να ταιριάζουν σε όλες τις απαιτήσεις των προσομοιώσεων δικτύων αισθητήρων. Έχουν αναπτυχθεί κάποιοι προσομοιωτές σε Matlab, Java και C++, οι οποίοι έχουν αναπτυχθεί για συγκεκριμένες εφαρμογές και εκμεταλλεύονται, ανάλογα με την εφαρμογή, υποθέσεις για μεγαλύτερη αποτελεσματικότητα. Οι DE επεξεργαστές θεωρούνται πιο ακριβείς υλοποιήσεις, λόγω της συνεχούς θεώρησης του χρόνου και της διακριτής θεώρησης των events. Υπάρχουν πολλοί διαθέσιμοι open-source και εμπορικοί προσομοιωτές. Μια κλάση προσομοιωτών περιλαμβάνει επεκτάσεις κλασικών προσομοιωτών δικτύων όπως ο ns-2, ο J-Sim και ο GloMoSim/QualNet. Μια άλλη κλάση προσομοιωτών είναι οι software-in-the-loop, οι οποίοι ενσωματώνουν το λογισμικό του κόμβου στην προσομοίωση. Γι' αυτό, συνήθως συνδέονται με συγκεκριμένες πλατφόρμες υλικού. Τέτοια παραδείγματα είναι το TOSSIM για κόμβους του Berkeley και το Em* (Emstar) για βασισμένους στο Linux κόμβους όπως οι πλατφόρμες Sensoria WINS NG .

Στον Πίνακα 3.4 δίνεται μια λίστα από υπάρχοντες προσομοιωτές που έχουν χρησιμοποιηθεί στην έρευνα για τα δίκτυα αισθητήρων. Ο ns-2, ο GloMoSim και ο Opnet είναι της εποχής του διαδικτύου προσομοιωτές, οι οποίοι σχεδιάστηκαν για να προσομοιώνουν στοίβες δικτύου με διαδοχικά επίπεδα. Το ns-2, αρχικά, δεν υποστήριζε ασύρματα δίκτυα (επέκταση που υποστήριζε προστέθηκε στη συνέχεια). Το GloMoSim σχεδιάστηκε ειδικά για υποστηρίζει ασύρματα δίκτυα και διαθέτει ένα πολύ καλό μοντέλο διάδοσης. Το Opnet είναι ένας βιομηχανικός, εμπορικός προσομοιωτής, εφάμιλλος του ns-2, ο οποίος είναι γνωστός για την ευρωστία του και την ικανότητα να πραγματοποιεί πολύ γρήγορες και σε μεγάλη κλίμακα προσομοιώσεις. Το SensorSim του UCLA αποτελεί επέκταση του ns-2 με την προσθήκη modules για μοντελοποίηση των μετρήσεων και της κατανάλωσης ενέργειας, ενός πρωτοκόλλου στοίβας και ένα module για την προσομοίωση ώστε να αλληλεπιδρά με πραγματικούς κόμβους. Το Em* είναι μια πλατφόρμα λογισμικού για συστήματα βασισμένα στο Linux. Παρέχει διαφορετικά επίπεδα προσομοίωσης περιβάλλοντος, αρχίζοντας από απλή προσομοίωση περιβάλλοντος σε ένα μόνο PC και φτάνοντας σε σενάρια ανάπτυξης δικτύων προερχόμενα από την πραγματική ζωή. Το SENS είναι ένας σχετικά πρόσφατος προσομοιωτής από το UIUC (University of Illinois Urbana-Champaign) και μοντελοποιεί τους κόμβους ως τρία διαφορετικά components: εφαρμογής, δικτύου και φυσικό. Ακόμα, μοντελοποιεί και το ασύρματο περιβάλλον στο οποίο υποτίθεται πως εργάζεται ο κόμβος. Το MANTIS είναι ένα λειτουργικό σύστημα σχεδιασμένο για να εκτελούνται σε αυτό οι Nymph κόμβοι, όπως και οι MICA2 της CrossBow. Το MANTIS χρησιμοποιεί έναν εξομοιωτή που εξομοιώνει κάθε κόμβο ως μια ξεχωριστή διεργασία. Το Prowler είναι ένα εργαλείο προσομοίωσης που παρέχεται από το ISIS και "τρέχει" με το MATLAB. Το βασικό χαρακτηριστικό του είναι η προσομοίωση μεταδόσεων, διασποράς, λήψεων και συγκρούσεων πακέτων σε δίκτυα. Το JProwler είναι η Java έκδοση του Prowler. Το SIESTA είναι ένας προσομοιωτής από το Vanderbilt University και έχει ως κύριο στόχο τον έλεγχο και την προσομοίωση middleware πλατφορμών.

Πίνακας 3.4: Λίστα γνωστών προσομοιωτών δικτύων και χαρακτηρισρικά τους

Simulators	Availability	Scalability	Programming Language	Sensor Modules	Wireless Models	Event/Time
NS2	√	√	C++	**X**	√	Event
GLOMOSIM	√	√	C++	**X**	√	Event
OPNET	**X**	√	C++	**X**	√	Event
SensorSim	√	√	C++	√	√	Event

TOSSIM	√	**X**	nesC	√	√	Event
EmStar	√	√	nesc/C++	√	√	Event/Time
SENS	√	√	C++	√	√	Event
MANTIS	√	**X**	C	√	**X**	Time
SIESTA	√	√	JAVA	√	**X**	Time
Prowler	√	√	MATLAB	**X**	√	Event
JProwler	√	√	JAVA	**X**	√	Event

Μετά την σύντομη αναφορά στις κατηγορίες των προσομοιωτών σε επίπεδο κόμβου και στα πιο αντιπροσωπευτικά παραδείγματα, θα ασχοληθούμε με περισσότερες λεπτομέρειες στο TOSSIM, τον προσομοιωτή για τους κόμβους του Berkeley, ο οποίος χρησιμοποιείται και στην υλοποίηση που συνοδεύει την εργασία.

3.4.2 TOSSIM

Το TOSSIM είναι ο βασικός προσομοιωτής του TinyOS. Ένας προσομοιωτής του TinyOS έχει τέσσερις βασικές απαιτήσεις:

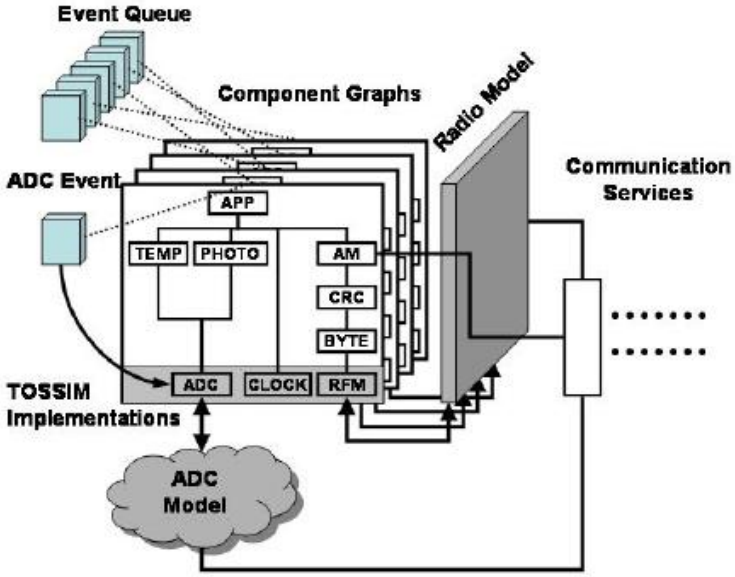
- **Κλιμάκωση (Scalability)**: Ο προσομοιωτής πρέπει να είναι σε θέση να ανταπεξέλθει σε μεγάλα δίκτυα με χιλιάδες κόμβους και σε ένα μεγάλο εύρος από διαφορετικούς σχηματισμούς. Το μεγαλύτερο δίκτυο αισθητήρων με TinyOS που έχει αναπτυχθεί είχε περίπου 850 κόμβους. Ο προσομοιωτής πρέπει να είναι σε θέση να ανταπεξέλθει σε αυτό και σε ακόμα μεγαλύτερα στο μέλλον.
- **Πληρότητα (Completeness)**: Ο προσομοιωτής πρέπει να καλύπτει όσο περισσότερες αλληλεπιδράσεις του συστήματος μπορεί και να συλλαμβάνει τη συμπεριφορά σε ένα μεγάλο εύρος επιπέδων. Αλγόριθμοι και πρωτόκολλα προσομοίωσης είναι χρήσιμα, αλλά η ευαίσθητη φύση των δικτύων αισθητήρων απαιτεί την προσομοίωση ολόκληρων εφαρμογών.
- **Ακρίβεια (Fidelity)**: Ο προσομοιωτής πρέπει να συλλαμβάνει τη συμπεριφορά του δικτύου σε στοιχειώδες επίπεδο. Η σύλληψη ευαίσθητων χρονικών αλληλεπιδράσεων σε ένα κόμβο αλλά και μεταξύ κόμβων, είναι σημαντική για την αξιολόγηση και τη δοκιμή. Ακόμη, πρέπει να προλαμβάνει απρόβλεπτες αλληλεπιδράσεις και όχι μόνο όσες έχει προβλέψει ο σχεδιαστής.
- **Γεφύρωμα (Bridging)**: Ο προσομοιωτής πρέπει να γεφυρώσει το χάσμα ανάμεσα στους αλγορίθμους και την υλοποίηση, επιτρέποντας στους σχεδιαστές να ελέγχουν και να επαληθεύουν ότι ο κώδικας θα είναι λειτουργικός σε πραγματικό υλικό.

Το TOSSIM αιχμαλωτίζει τη συμπεριφορά και τις αλληλεπιδράσεις χιλιάδων δικτυακών κόμβων του TinyOS σε διακριτά τμήματα δικτύου. Στο σχημα 3.7 φαίνεται μια γραφική άποψη του TOSSIM. Η αρχιτεκτονική του TOSSIM αποτελεί σύνθεση πέντε τμημάτων: υποστήριξη για τη μεταγλώττιση του γράφου components του TinyOS, μια διακριτή ουρά για events, έανς μικρός αριθμός από επανυλοποιημένα αφηρημένα components υλικού του TinyOS, μηχανισμούς για επεκτατά μοντέλα ραδιοεπικοινωνιών και ADC (Analog to Digital Converter) και υπηρεσίες επικοινωνιών ώστε εξωτερικά προγράμματα να μπορούν να αλληλεπιδρούν με την προσομοίωση.

Το TOSSIM εκμεταλλεύεται τη δομή του TinyOS και τη μεταγλώττιση ολόκληρου του προγράμματος για να παράγει προσομοιώσεις διακριτών events, τα οποία έχουν προέλθει απευθείας από το γράφο components του TinyOS. Εκτελεί τον ίδιο κώδικα, που εκτελείται και στους πραγματικούς κόμβους. Με την αντικατάσταση μερικών

χαμηλού επιπέδου components (τα γραμμοσκιασμένα τμήματα στο Σχήμα 3.11), το TOSSIM μετατρέπει τα interrupts σε επίπεδο υλικού σε διακριτά events του προσομοιωτή. Η ουρά γεγονότων του προσομοιωτή είναι αυτή που κατευθύνει την εκτέλεση μιας εφαρμογής σε TinyOS. Ο υπόλοιπος κώδικας σε TinyOS εκτελείται χωρίς καμία αλλαγή.

Το TOSSIM χρησιμοποιεί ένα πολύ απλό αλλά ταυτόχρονα δραστικό μηχανισμό αφαίρεσης για το ασύρματο δίκτυο. Το δίκτυο είναι ένας κατευθυνόμενος γράφος όπου κάθε κορυφή είναι ένας κόμβος και σε κάθε ακμή μπορεί να συμβεί μια μικρή πιθανότητα σφάλματος. Κάθε κόμβος διαθέτει ένα ατομικό τμήμα επικράτειας που αναπαριστά το τι ακούει στο κανάλι ραδιοεπικοινωνίας. Αυτή η αφαίρεση επιτρέπει δοκιμές υπό τέλειες συνθήκες μετάδοσης (ρυθμός σφαλμάτων μηδέν), μπορεί να εντοπίσει τα κρυμμένα προβλήματα στα ακραία τμήματα του δικτύου (για τους κόμβους *a,b,c* υπάρχουν οι ακμές *(a,b)* και *(b,c)* αλλά όχι η *(a,c)*) και μπορεί να συλλάβει πολλά διαφορετικά προβλήματα τα οποία μπορεί να προκύψουν στη μετάδοση πακέτων.



Σχήμα 3.7: Η αρχιτεκτονική του TOSSIM

Ο προσομοιωτής παρέχει ένα σύνολο από επικοινωνιακές υπηρεσίες για αλληλεπίδραση με εξωτερικές εφαρμογές. Αυτές οι υπηρεσίες επιτρέπουν σε προγράμματα να συνδεθούν στο TOSSIM μέσω TCP sockets για να παρακολουθούν ή να κατευθύνουν μια εφαρμογή. Λεπτομέρειες του ADC και του επικοινωνιακού μοντέλου, όπως μετρήσεις και ρυθμός απωλειών, μπορούν να χρησιμοποιηθούν για ανάκτηση πληροφοριών αλλά και να ρυθμιστούν. Τα προγράμματα μπορούν να λαμβάνουν υψηλότερου επιπέδου πληροφορίες, όπως μεταδόσεις και λήψεις πακέτων ή events στο επίπεδο εφαρμογής.

Το TOSSIM υποστηρίζει την αλυσίδα εργαλείων του TinyOS, κάνοντας εύκολη τη μεταβάση από προσομοιούμενο στο πραγματικό δίκτυο. Ακόμη, είναι δυνατή η χρήση παραδοσιακών εργαλείων προγραμματισμού όπως είναι οι debuggers. Δεδομένου πως το TOSSIM είναι ένας προσομοιωτής διακριτών γεγονότων, οι χρήστες μπορούν θα θέτουν breakpoints και να διατρέχουν την εκτέλεση βήμα-βήμα σε πραγματικό χρόνο χωρίς να διασπούν την εκτέλεση. Επιπλέον, οι μηχανισμοί που παρέχουν σε

προγράμματα τη δυνατότητα να αλληλεπιδρούν και να παρακολουθούν μια προσομοίωση καθιστούν το TOSSIM μια απλή και αποτελεσματική μηχανή προσομοίωσης.

Μια εφαρμογή του TinyOS αντί να εκτελεστεί σε κάποιο κόμβο μπορεί ο χρήστης να την μεταγλωττίσει στην πλατφόρμα του TOSSIM, όπου ακόμα μπορεί να κάνει debug, ελέγχους και ανάλυση αλγορίθμων σε ένα ελεγχόμενο και επαναλαμβανόμενο περιβάλλον. Το TOSSIM έχει ως πρωταρχικό σκοπό να προσομοιώσει την εκτέλεση του TinyOS και όχι τον πραγματικό κόσμο. Μπορεί να χρησιμοποιηθεί για να κατανοήσουμε τα αίτια κάποιων συμπεριφορών που παρατηρούμε, αλλά δεν πρέπει να περιμένουμε ότι θα τις αποτυπώνει όλες. Αυτός είναι και ο λόγος όπου δεν είναι πάντα η κατάλληλη επιλογή για προσομοίωση, αφού αναπαριστά άλλες συμπεριφορές με ακρίβεια και άλλες πιο απλοποιημένες. Στη συνέχεια παρατίθεται μια περίληψη των χαρακτηριστικών του:

- **Ακρίβεια (Fidelity)**: Το TOSSIM συλλαμβάνει τη συμπεριφορά του TinyOS σε πολύ χαμηλό επίπεδο. Προσομοιώνει το δίκτυο σε επίπεδο bit, κάθε ADC ξεχωριστά και κάθε interrupt του συστήματος.
- **Χρόνος (Time)**: Ενώ το TOSSIM συγχρονίζει ακριβώς τα interrupts (επιτρέποντας προσομοίωση των επικοινωνιών σε επίπεδε bit), δε μοντελοποιεί το χρόνο εκτέλεσης. Από την πλευρά του TOSSIM ένα κομμάτι κώδικα τρέχει στιγμιαία.
- **Μοντέλα (Models)**:Το TOSSIM δεν μοντελοποιεί τον πραγματικό κόσμο. Αντίθετα, παρέχει αφαιρέσεις συγκεκριμένων φαινομένων του πραγματικού κόσμου. Ο χρήστης μπορεί να χειριστεί αυτές τις αφαιρέσεις για να υλοποιήσει κάποια πολύπλοκα μοντέλο έξω από την προσομοίωση.
  - **Ραδιοεπικοινωνία (Radio)**: Το TOSSIM δε μοντελοποιεί τη διάδοση των ραδιοεπικοινωνιών. Αντίθετα, παρέχει μια αφαίρεση της κατεύθυνσης και του βαθμού σφάλματος στην επικοινωνία ανάμεσα σε δύο κόμβους. Έτσι, ένα εξωτερικό ως προς το TOSSIM πρόγραμμα μπορεί να εισάγει το δικό του μοντέλο επικοινωνίας και ρυθμούς σφαλμάτων. Η ύπαρξη συγκεκριμένων κατευθύνσεων στους ρυθμούς σφαλμάτων επιτρέπει την εύκολη μοντελοποίηση ασύμμετρων συνδέσεων. Ανεξάρτητοι ρυθμοί σφαλμάτων σημαίνει πως τα μεγαλύτερα πακέτα έχουν μεγαλύτερη πιθανότητα φθοράς και η πιθανότητα σφάλματος σε κάθε πακέτο είναι ανεξάρτητη.
  - **Ισχύς/Ενέργεια (Power/Energy)**: Πρόσφατα έχει προστεθεί το PowerTOSSIM, ένα βαθμωτό περιβάλλον προσομοίωσης, που παρέχει μια ακριβή και ανά κόμβο εκτίμηση της κατανάλωσης ενέργειας. Πρόκειται για μια επέκταση του TinyOS, η οποία αποτελείται από όργανα παρακολούθησης ενεργειακών αλλαγών καταστάσεων σε επίπεδο υλικού, ένα ακριβή μηχανισμό μέτρησης των κύκλων της CPU και ένα εργαλείο ανάλυσης και οπτικοποίησης των αποτελεσμάτων της ενεργειακής κατανάλωσης ανά κόμβο.
- **Δόμηση (Building)**: Το TOSSIM δομείται απευθείας από κώδικα του TinyOS. Για να προσομοιώσεις ένα πρωτόκολλο ή ένα σύστημα πρέπει να γράψεις μια υλοποίηση αυτού σε TinyOS. Αυτό είναι πιο δύσκολο από ότι γενικά μια προσομοίωση, αλλά η υλοποίηση αυτή μπορεί να εκτελεστεί απευθείας σε κόμβους.

- **Ατέλειες (Imperfections)**: Το TOSSIM κάνει αρκετές υποθέσεις που απλοποιούν τη συμπεριφορά του TinyOS. Έτσι, είναι πιθανό κώδικας που τρέχει στην προσομοίωση να είναι προβληματικός σε εκτελέσεις σε κανονικούς κόμβους. Για παράδειγμα, τα interrupts στο TOSSIM είναι non-preemptive, ενώ σε πραγματικούς κόμβους ένα interrupt μπορεί να ενεργοποιηθεί ενώ “τρέχει” άλλος κώδικας. Ακόμα, αν ένας interrupt handler “τρέξει” πολύ, ένας κόμβος σε πραγματική εφαρμογή μπορεί να καταρρεύσει, ενώ στο TOSSIM κάτι τέτοιο δε θα συμβεί αφού ο κώδικας εκτελείται στιγμιαία και έτσι το πρόβλημα δε θα εμφανιστεί.
- **Δικτύωση (Networking)**: Το TOSSIM προσομοιώνει τη στοίβα δικτύου του 40 Kbit RFM mica, συμπεριλαμβανομένων του MAC (Medium Access Control), κωδικοποίησης, χρονομέτρησης και σύγχρονων επιβεβαιώσεων. Δεν προσομοιώνει τη στοίβα του mica2 ChipCon CC1000 αφού η συμπεριφορά του δεν έχει κατανοηθεί πλήρως για να δημιουργηθεί προσομοιωτής.
- **Κύρος (Authority)**: Η εμπειρία, που έχει προέλθει από αρχικές αναπτύξεις δικτύων του TinyOS στον πραγματικό κόσμο, έχει δείξει ότι έχουν πολύπλοκη και ευμετάβλητη συμπεριφορά. Τα αποτελέσματα που προέρχονται από το TOSSIM δεν πρέπει να θεωρούνται αξιωματικά. Ακόμη, το TOSSIM δεν πρέπει να θεωρείται το τέλος μιας αξιολόγησης αλλά ένα σύστημα που επιτρέπει στους χρήστες να αφαιρούν το θόρυβο του περιβάλλοντος για να κατανοήσουν καλύτερα τη συμπεριφορά αλγορίθμων.

Μέρος Β

Αρχιτεκτονική διαχείρισης πληροφορίας με γνώση ενέργειας και σφάλματος

ΚΕΦΑΛΑΙΟ 4

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

4.1 Εισαγωγή

Στο πρώτο μέρος της εργασίας αυτής έγινε μια εκτενής εισαγωγή στο πεδίο των ασυρμάτων δικτύων αισθητήρων. Αφού πρώτα αναφερθήκαμε στα γενικά χαρακτηριστικά των δικτύων αυτών, αλλά και των κόμβων που τα αποτελούν, στη συνέχεια παρουσιάστηκαν μια σειρά από πρωτόκολλα και τεχνικές δρομολόγησης, που χρησιμοποιούνται στα δίκτυα αυτά. Τέτοιες τεχνικές λαμβάνουν υπόψη διάφορους από τους πολλούς και σημαντικούς σχεδιαστικούς παράγοντες, βελτιώνοντας, η κάθε μία, μία πτυχή του συνολικού προβλήματος της αποτελεσματικής δρομολόγησης. Στη συνέχεια του πρώτου μέρους παρουσιάστηκε το πιο διαδεδομένο λειτουργικό για την ανάπτυξη εφαρμογών σε Α.Δ.Α., το TinyOS. Το λειτουργικό αυτό προτείνει μια διαφορετική αντιμετώπιση του προγραμματισμού σε σχέση με άλλες συμβατικές γλώσσες προγραμματισμού μέσα από τη γλώσσα nesC που είναι μία δηλωτική επέκταση της C.

Μετά από αυτήν την εισαγωγή είμαστε έτοιμοι να περιγράψουμε, αρχικά, τις σχεδιαστικές αρχές μιας αρχιτεκτονικής προώθησης μηνυμάτων που προτείνουμε για τη διαχείριση πληροφορίας σε ένα ασύρματο δίκτυο αισθητήρων. Σε αυτό το κεφάλαιο περιγράφονται οι αρχές και η φιλοσοφία με τις οποίες σχεδιάστηκε η προτεινόμενη αρχιτεκτονική διαχείρισης πληροφορίας. Οι κεντρικοί άξονες του κεφαλαίου αυτού είναι η περιγραφή των διακριτών ρόλων που έχουν οι κόμβοι σε ένα Α.Δ.Α., η περιγραφή του μηχανισμού πρόβλεψης που χρησιμοποιείται για τον υπολογισμό των τιμών που δεν αποστέλλονται και η περιγραφή της συνάρτησης αξιολόγησης που χρησιμοποιείται για να αξιολογηθεί η αναγκαιότητα μιας αποστολής πληροφορίας.

4.2 Χρησιμότητα της αρχιτεκτονικής

Ένα από τα γενικά συμπεράσματα του Μέρους Α είναι ότι οι πόροι σε κάθε κόμβο ενός Α.Δ.Α. είναι περιορισμένοι και κατ'επέκταση οι πόροι ενός Α.Δ.Α. είναι περιορισμένοι. Μια ακόμη σημαντική παράμετρος είναι το ενεργειακό κόστος των διάφορων διεργασιών που επιτελεί ένας κόμβος. Με στοιχεία από τη βιβλιογραφία (βλέπε και Παράρτημα Β) παρατηρούμε ότι η τάξη μεγέθους του ενεργειακού κόστους της αποστολής δεδομένων ανά bit είναι 100 φορές μεγαλύτερη από αυτήν της εκτέλεση μιας εντολής από τον επεξεργαστή. Γεννιέται, λοιπόν, ένα πολύ ισχυρό κίνητρο δημιουργίας μιας αρχιτεκτονικής που «αντικαθιστά» μεταδόσεις και λήψεις μηνυμάτων με εκτέλεση εντολών από τον επεξεργαστή. Η βασική ιδέα λοιπόν της αρχιτεκτονικής που προτείνεται στην εργασία αυτή είναι η αποφυγή αποστολών μηνυμάτων που δεν είναι απαραίτητα για τη διατήρηση της συνοχής των δεδομένων. Ένα επιπλέον στοιχείο είναι ότι, πέρα από την ενέργεια που καταναλώνεται για την εκτέλεση εντολών και την αποστολή και παραλαβή πακέτων καταναλώνεται ενέργεια από τον επεξεργαστή για όση ώρα παραμένει ανοιχτός. Αυτή η ενέργεια αν δε γίνει καμμία έξυπνη διαχείριση του χρόνου είναι μεγάλη και δεν προσφέρει τίποτα. Αυτό μπορεί να αντισταθμιστεί αφού κάθε κόμβος μπορεί να λειτουργεί σε κατάσταση αδράνειας (idle), σε κατάσταση αναμονής (stand by) ή και σε κατάσταση εξοικονόμησης ενέργειας (power-save mode).

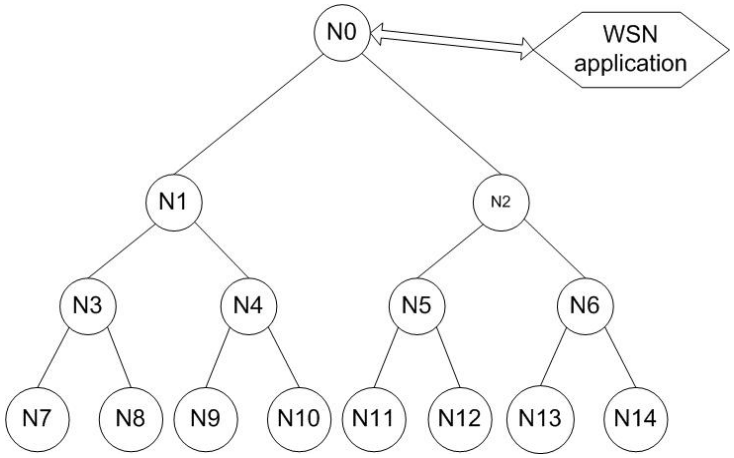
Ένας πρώτος στόχος είναι να αποφεύγονται οι περιττές αποστολές και ένας επόμενος στόχος είναι να διατηρείται κάθε κόμβος σε μια κατάσταση με όσο το δυνατόν μικρότερη ενεργειακή απώλεια.

4.3 Ιεραρχία δικτύου

Ο σχεδιασμός της αρχικτεκτονικής βασίζεται σε κόμβους που έχουν τρεις διακριτούς ρόλους:

- Αισθητήριοι κόμβοι (ή πηγές), οι οποίοι μετρούν συγκεκριμένες φυσικές παραμέτρους και τις μεταδίδουν προς άλλους κόμβους του συστήματος με τελικό παραλήπτη τον κόμβο-βάση (ή καταβόθρα)
- Κόμβοι επικοινωνίας (ή κόμβοι αναμετάδοσης), οι οποίοι λαμβάνουν ασύρματα μετρήσεις από τις πηγές (ή από άλλους κόμβους επικοινωνίας) και τις αναμεταδίδουν προς τον τελικό αποδέκτη αυτής της πληροφορίας. Οι κόμβοι επικοινωνίας χρησιμοποιούνται όταν η απ'ευθείας επικοινωνία μεταξύ των πηγών και του κόμβου-καταβόθρα δεν είναι εφικτή (λόγω περιορισμένων πόρων, όπως, π.χ., ενέργεια και υποδομή επικοινωνίας) καθιστώντας την εφικτή.
- Κόμβοι-καταβόθρες, οι οποίοι είναι οι τελικοί αποδέκτες της μετρούμενης πληροφορίας. Οι κόμβοι αυτοί είναι, συνήθως, συνδεδεμένοι με κάποιο συμβατικό υπολογιστικό σύστημα για πιο σύνθετη επεξεργασία των συγκεντρωμένων μετρήσεων. Εναλλακτικά, οι κόμβοι-καταβόθρες μπορεί να σχηματίζουν μια πιο εξελιγμένη τοπολογία δικτύου (όπως ένα WLAN ή ένα ενσύρματο δίκτυο) για επιπλέον διαβίβαση.

Οι προαναφερθέντες κόμβοι είναι οργανωμένοι σε έναν κατευθυνόμενο ακυκλικό γράφο, δημιουργώντας μια δενδρική δομή, της οποίας η ρίζα διαδραματίζει ένα διακριτό ρόλο. Η ρίζα του δέντρου είναι ο κόμβος-καταβόθρα (υπάρχει ακριβώς ένας τέτοιος κόμβος) και όλοι οι άλλοι κόμβοι είναι είτε κόμβοι επικοινωνίας είτε πηγές (τουλάχιστον μία πηγή είναι απαραίτητη). Στο Σχήμα 4.1 ο κόμβος-καταβόθρα είναι ο N0, οι κόμβοι N14, N13, N12, N11, N10, N9, N8 και N7 είναι οι πηγές (φύλλα του δέντρου) και οι υπόλοιποι κόμβοι θα μπορούσαν να είναι πηγές αλλά πρωτίστως παίζουν το ρόλο των κόμβων αναμετάδοσης, των ροών των δεδομένων (προς τη ρίζα, N0).



Σχήμα 4.1: Παράδειγμα δενδρικής τοπολογίας σε Α.Δ.Α.

Όλοι οι ενδιάμεσοι κόμβοι στην τοπολογία αυτή δεσμεύουν χώρο στη μνήμη τους ανάλογα με το πλήθος των διακριτών ροών δεδομένων (ΡΔ) που εξυπηρετούν. Μια ΡΔ είναι μια συσχέτιση μεταξύ ενός κόμβου φύλλου (αισθητήριου κόμβου) με τη ρίζα της τοπολογίας. Για παράδειγμα ο N1 εξυπηρετεί τέσσερις διακριτές ΡΔ, μεταξύ των κόμβων N7 → N0, N8 → N0, N9 → N0 και N10 → N0. Σε κάθε κόμβο, η δεσμευμένη

μνήμη επιτρέπει τη δημιουργία buffers (ανά ΡΔ) στους οποίους θα αποθηκεύεται συγκεκριμένος αριθμός μετρήσεων από κάθε ροή. Οι αισθητήριοι κόμβοι δεσμεύουν, ομοίως, χώρο μόνο για τη δική τους ροή.

Οι κόμβοι επικοινωνίας και οι πηγές προσπαθούν να βελτιστοποιήσουν την κατανάλωση ενέργειας μέσα στο Α.Δ.Α. Συγκεκριμένα, όταν εμφανιστεί η ανάγκη για προώθηση μιας ΡΔ, ένας ενσωματωμένος μηχανισμός μετάδοσης (Transmission Control Mechanism, TCM) αξιολογεί τη συγκεκριμένη μετάδοση. Ο μηχανισμός αυτός λαμβάνει υπ'όψη του διάφορα κριτήρια και μπορεί να αποφασίσει ότι η διάδοση της συγκεκριμένης ροής δεν είναι απαραίτητη. Οι αντίστοιχοι μηχανισμοί άλλων κόμβων πρέπει να μπορούν να αντιλαμβάνονται τι συνέβη και να δρουν ανάλογα. Παρακάτω, θα αναφερθούμε εκτενώς στα κριτήρια που χρησιμοποιούν οι μηχανισμοί αυτοί για να αξιολογήσουν την επαναμετάδοση ενός μηνύματος μιας ροής. Πριν, όμως από αυτό, πρέπει να τονιστεί ότι όλοι οι μηχανισμοί λειτουργούν συγχρονισμένα και επιτρέπονται μικρές, μόνο, αποκλίσεις από έναν καθολικό συγχρονισμό του ασυρμάτου δικτύου.

Η αρχιτεκτονική προώθησης μηνυμάτων που συζητείται βασίζεται στην υπό προϋποθέσεις μετάδοση πληροφορίας μέσα στο Α.Δ.Α. και άρα οι κόμβοι του δικτύου πρέπει να μπορούν να καθορίσουν εάν οι κόμβοι από τους οποίους αναμένεται η πληροφορία είναι λειτουργικοί. Για να επιτευχθεί αυτό χρησιμοποιούνται Heart-Beat (ΗΒ) μηνύματα. Τα μηνύματα αυτά μεταδίδονται χρησιμοποιώντας αξιόπιστες υπηρεσίες μετάδοσης ώστε να είναι σίγουρη η παράδοση τους. Τα μηνύματα αυτά προωθούνται από την αρχιτεκτονική σε κάθε περίπτωση και όταν κάποιο ΗΒ μήνυμα χαθεί (π.χ. δεν παραδοθεί μέσα στον προκαθορισμένο χρόνο) ο κόμβος στον οποίο κατευθύνεται η ροή θεωρεί πως κάποιος κόμβος από τα χαμηλότερα επίπεδα δεν είναι πια λειτουργικός. Παρόμοια μηνύματα που υποδηλώνουν την ορθή λειτουργία κάποιων κόμβων μπορεί να αποστέλλονται και σε πιο χαμηλό επίπεδο δρομολόγησης, αλλά κάτι τέτοιο δε θα μας απασχολήσει σε αυτό το επίπεδο. Οι μηχανισμοί TCM είναι πλήρως ενήμεροι για την κατάσταση των μηνυμάτων ΗΒ, δηλαδή γνωρίζουν πότε παρελήφθη το προηγούμενο και πότε αναμένεται το επόμενο. Τα μηνύματα αυτά δεν έχουν ως μοναδικό ρόλο την ενημέρωση του δικτύου για την κατάσταση κάποιων κόμβων. Μαζί μ'αυτό φέρουν και πληροφορία, δηλαδή τιμές των αντιστοίχων ΡΔ, και έτσι ενσωματώνονται και οι δύο αυτές λειτουργίες σε ένα μήνυμα κερδίζοντας, έτσι, μια επιπλέον αποστολή χωρίς δεδομένα. Αυτό το γεγονός μας δίνει την δυνατότητα να γνωρίζουμε ότι αν το δίκτυο είναι λειτουργικό τότε θα λάβουμε σίγουρα ένα μήνυμα με πληροφορία μετά από ένα προκαθορισμένο χρονικό διάστημα. Συνεπώς, όταν ξέρουμε ότι θα σταλεί σε μικρό χρονικό διάστημα ένα τέτοιο μήνυμα μπορούμε να αποφύγουμε μια επιπλέον αποστολή.

Κάθε μηχανισμός TCM υλοποιεί ένα σχήμα εκτίμησης της τιμής που ελήφθη[1]. Κάθε φορά, λοιπόν, που μια νέα μέτρηση φτάνει στο μηχανισμό, αυτός υπολογίζει εάν ο μηχανισμός του επόμενου κόμβου μπορεί να υπολογίσει την τιμή αυτή χωρίς να τη λάβει πραγματικά. Για να επιτευχθεί αυτός ο στόχος όλοι οι μηχανισμοί χρησιμοποιούν έναν προσυμφωνημένο μηχανισμό εκτίμησης (κοινός σε όλο το Α.Δ.Α.). Ο τοπικός μηχανισμός υπολογίζει την εκτιμώμενη τιμή (ΕΤ) της υπό μέτρηση φυσικής μεταβλητής, με χρήση των μετρήσεων που έχουν αποθηκευτεί στη μνήμη των ΡΔ. Η τιμή αυτή συγκρίνεται με την αληθινή τιμή και το παραγόμενο σφάλμα χρησιμοποιείται για την αξιολόγηση της μετάδοσης. Εάν, τελικά, η μετάδοση δεν πραγματοποιηθεί, τότε ο μηχανισμός του επόμενου κόμβου (πάνω στο μονοπάτι προς τον κόμβο-καταβόθρα) θα πραγματοποιήσει ακριβώς την ίδια εκτίμηση και η τιμή που υπολογίστηκε, ΕΤ, θα είναι η τιμή που θα διαδοθεί στο Α.Δ.Α. Η υποχρεωτική μετάδοση των ΗΒ μηνυμάτων

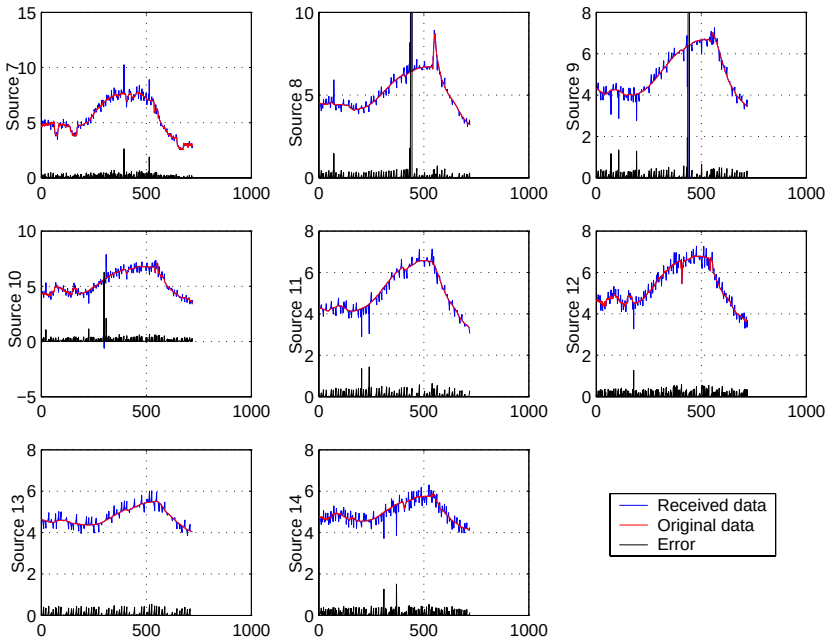
[1] Η φυσική παράμετρος που είναι υπό μέτρηση θεωρείται ότι μεταβάλεται συναρτήσει του χρόνου σχετικά ομαλά, π.χ. πολυωνυμικά.

αποτρέπει τη χρονική αλλά και χωρική διεύρυνση εσφαλμένων εκτιμήσεων. Αυτή η ιδέα εφαρμόζεται για κάθε ροή που χειρίζεται ο τρέχων κόμβος.

Η μέθοδος υπολογισμού της εκτιμώμενης τιμής που χρησιμοποιείται είναι το πολυώνυμο Lagrange. Αυτή η επιλογή βασίζεται στην ανάγκη για χαμηλής πολυπλοκότητας, αρκετά ακριβή σχέση προσέγγισης. Μια σχέση μεγαλύτερης πολυπλοκότητας θα ήταν ασύμβατη με τους πόρους που μας παρέχει ένας κόμβος (τυπικές τιμές είναι 512 KB μνήμης, 4-8 KHz CPU και περιορισμένα αποθέματα ενέργειας). Ο υπολογισμός της ΕΤ βασίζεται στις *n-1* πιο πρόσφατα ληφθείσες μετρήσεις, καταλήγοντας σε ένα πολυώνυμο παρεμβολής βαθμού *n*:

$$P(t) = \sum_{j=0}^{n} P_j(t) \cdot f_j, \; where \; P_j(t) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{t - t_k}{t_j - t_k} \qquad (4.1)$$

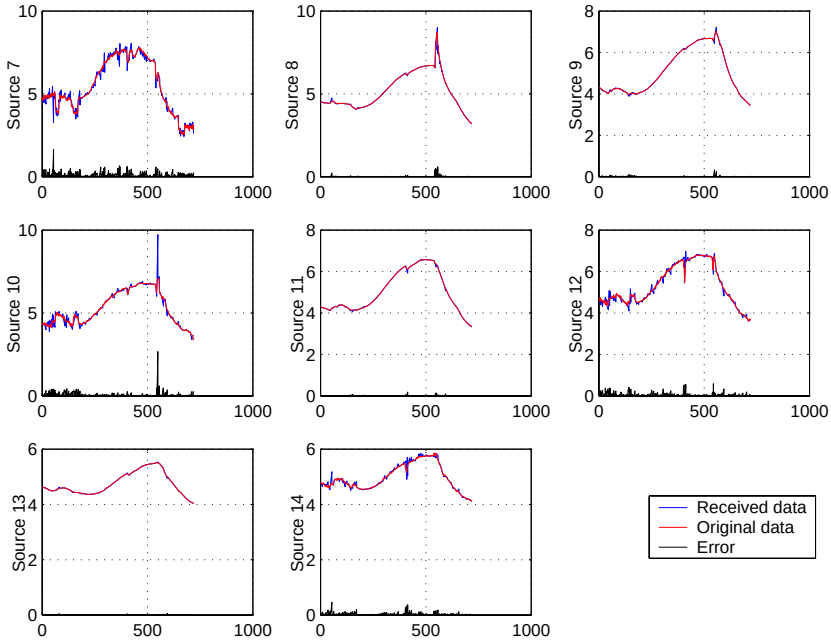
f_i είναι οι προηγούμενες μετρήσεις, $P_j(t)$ είναι οι συντελεστές Lagrange, $P(t)$ είναι η εκτιμώμενη τιμή και t είναι ο χρόνος. Στην υλοποίηση που πραγματοποιήσαμε το διάστημα μεταξύ δυο διαδοχικών μετρήσεων t_i και t_{i+1} ($t_{i+1} - t_i = \delta t$) είναι σταθερό και η ΕΤ, $P(t)$, είναι η τιμή που αντιστοιχεί στην επόμενη μέτρηση.



Σχήμα 4.2: Δεδομένα που παρελήφθησαν από την καταβόθρα σε σύγκριση με τα αρχικά δεδομένα, με χρήση κυβικής παρεμβολής

Μετά από σειρά προσομοιώσεων μεταδόσης δεδομένων θερμοκρασίας από ένα δίκτυο όπως περιγράφεται παραπάνω παρατηρήθηκε ότι αυξημένος βαθμός πολυωνυμικής προσέγγισης είχε αποτέλεσμα αυξημένη ταλάντωση, με συνέπεια την εισαγωγή τοπικού σφάλματος απαγορευτικής τιμής. Πιο συγκεκριμένα, σύμφωνα με τα Σχήματα 4.2 και 4.3 παρατηρούμε ότι οι γραμμικές προσεγγίσεις (Σχήμα 4.3) δίνουν αποτελέσματα που μπορούν να θεωρηθούν αποδεκτά ενώ οι κυβικές προσεγγίσεις όχι.

Στα δύο σχήματα (4.2 και 4.3) παρατηρούμε τις τιμές των μετρήσεων που πραγματοποιήθηκαν από 8 ασιθητήριους κόμβους σε ένα Α.Δ.Α. σε σχέση με τις τιμές που έφτασαν στην καταβόθρα, καθώς και το σφάλμα που υπεισήλθε. Είναι εμφανές ότι οι ταλαντώσεις της κυβικής παρεμβολής μειώνουν την ποιότητα των δεδομένων και μας οδηγούν στο να επιλέξουμε τη γραμμική παρεμβολή που είναι, επιπλέον, και πιο απλή ως προς την πολυπλοκότητα, αλλά και τις ανάγκες αποθήκευσης δεδομένων.



Σχήμα 4.3: Δεδομένα που παρελήφθησαν από την καταβόθρα σε σύγκριση με τα αρχικά δεδομένα με χρήση γραμμικής παρεμβολής

Η μέθοδος εκτίμησης που προαναφέρθηκε έχει προσομοιωθεί για δεδομένα εξωτερικής θερμοκρασίας (που έχουμε πάρει από μετεωρολογικό σταθμό). Η μετρική σφάλματος που χρησιμοποιείται στη συνάρτηση αξιολόγησης (παρακάτω αναλύεται πιο διεξοδικά) υπολογίζεται ως εξής:

$Y = \left(\left\| d(i) - est(i) \right\| / \left\| d(i) \right\| \right)$	(4.2)

d(i) είναι η πραγματική μέτρηση και *est(i)* είναι η εκτιμώμενη τιμή. Η τιμή *Y* είναι το ποσοστιαίο σφάλμα που υπεισέρχεται λόγω της εκτίμησης της τιμής. Το ποσοστό αυτό είναι ένα από τα στοιχεία που χρησιμοποιεί η συνάρτηση αξιολόγησης για να αποφανθεί για τη χρησιμότητα μιας μετάδοσης. Το ποσοστό αυτό δεν μπορεί να υπερβαίνει ένα προκαθορισμένο κατώφλι, *T*, και όταν συμβεί αυτό τότε η μετάδοση της τιμής είναι επιβεβλημένη προκειμένου να διατηρηθεί η συνοχή των δεδομένων.

Μια άλλη παράμετρος που επηρεάζει την απόφαση του μηχανισμού TCM είναι το τρέχον απόθεμα ενέργειας του εν λόγω κόμβου. Ο μηχανισμός γνωρίζει το ακριβές ενεργειακό απόθεμα και το λαμβάνει υπόψη όταν πραγματοποιεί την εκτίμηση της επικείμενης μετάδοσης. Τέλος, ο μηχανισμός λαμβάνει υπόψη την αποστολή HB μηνυμάτων έτσι ώστε να εκμεταλλεύεται το γεγονός ότι είναι γνωστή η ανά τακτά

χρονικά διαστήματα μετάδοση και να μειώθεί η πιθανότητα μετάδοσης όταν είμαστε λίγο μετά ή λίγο πριν από τη μετάδοση ενός τέτοιου μηνύματος.

4.4 Σχεδιασμός συνάρτησης αξιολόγησης (utility function)

Σε αυτήν την ενότητα θα αναφερθούμε εκτενώς στη μέθοδο που υιοθετεί ο μηχανισμός TCM για να υπολογίσει τη συνάρτηση αξιολόγησης μιας μετάδοσης προς τον κόμβο-καταβόθρα. Έστω m_i ένα μήνυμα, M το σύνολο των μηνυμάτων και H το σύνολο των Heart-Beat μηνυμάτων. Επιπλέον, έστω U_k η αξιολόγηση του κόμβου k ως προς τη μετάδοση ενός νέου (όχι HB) μηνύματος προς τον κόμβο-καταβόθρα. Η U_k είναι συνάρτηση του χρόνου, του τρέχοντος αποθέματος ενέργειας του κόμβου και της μέτρησης που έχει ληφθεί από τη συγκεκριμένη ροή. Η U_k υπολογίζεται ως εξής:

$U_k = w_1 \cdot U^k_{energy} + w_2 \cdot U^k_{error} + (1 - \sum_{i=1,2} w_i) \cdot U^k_{time},\ \sum_{i=1,2} w_i \le 1,\ \ m_i \in (M - H)$	(4.3)
$U_k = 1,\ \ m_i \in H$	(4.4)

Τα βάρη w_i εξαρτώνται από την εκάστοτε εφαρμογή και είναι μη αρνητικά. Οι τρεις συνιστώσες της συνάρτησης αξιολόγησης για ένα συγκεκριμένο κόμβο k υπολογίζονται ως εξής:

$$U^k_{energy} = 1 - e^{\left[-10\frac{E}{E_{\max}}\right]} \qquad (4.5)$$

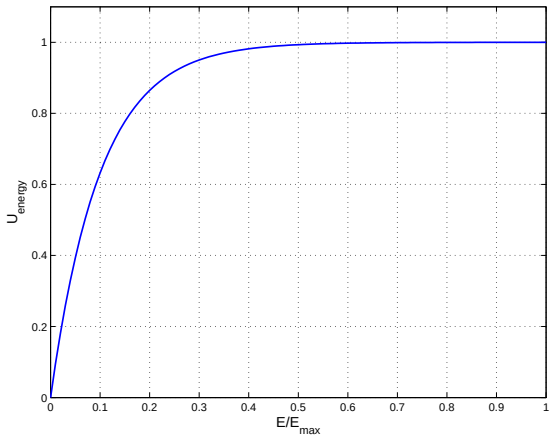
$$U^k_{error} = \begin{cases} err^2 / \left(err_{threshold}\right)^2 & , \quad err \le err_{threshold} \\ U_k = 1 & , \quad err \ge err_{threshold} \end{cases} \qquad (4.6)$$

$$U^k_{time} = \begin{cases} 2 \cdot \Delta t / \Delta \mathrm{T} & , \quad 0 \le \Delta t \le \Delta \mathrm{T}/2 \\ -2 \cdot \Delta t / \Delta \mathrm{T} + 2 & , \quad \Delta \mathrm{T}/2 \le \Delta t \le \Delta \mathrm{T} \end{cases} \qquad (4.7)$$

όπου E είναι το τρέχον ενεργειακό απόθεμα του εν λόγω κόμβου, E_{max} είναι η μέγιστη ενέργεια που μπορεί να αποθηκευτεί στο συσσωρευτή του κόμβου, *err* είναι το σφάλμα που οφείλεται στην εκτίμηση που είναι κοινή σε όλο το Α.Δ.Α., $err_{threshold}$ είναι η μέγιστη αποδεκτή απόκλιση από την πραγματική τιμή, ΔT είναι ο χρόνος μεταξύ HB μηνυμάτων και Δt είναι ο χρόνος που πέρασε από τη μετάδοση του προηγούμενου HB μηνύματος. Εάν το σφάλμα είναι μεγαλύτερο από το κατώφλι τότε η τιμή της συνάρτησης αξιολόγησης γίνεται ίση με ένα, ανεξάρτητα από τις τιμές των επί μέρους συνιστωσών. Αυτό γίνεται γιατί αν και η εξοικονόμηση ενέργειας είναι ο στόχος της αρχιτεκτονικής αυτό δεν πρέπει να γίνεται σε βάρος της επιθυμητής ακρίβειας και διακριτότητας των δεδομένων. Εδώ πρέπει να τονιστεί πώς η τιμή *err* είναι ίση με το αποτέλεσμα του τύπου (4.2).

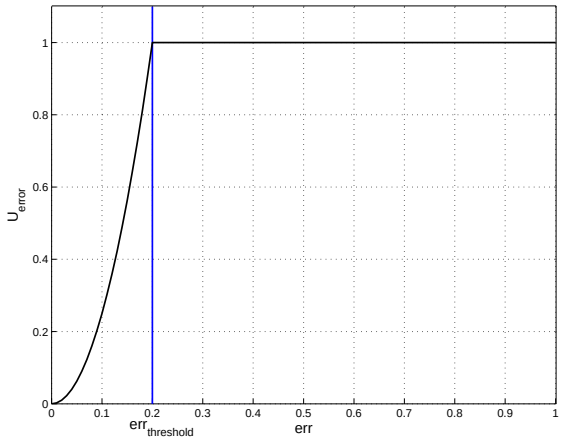
Όταν η τιμή της συνάρτησης αξιολόγησης μιας μετάδοσης για έναν συγκεκριμένο κόμβο και μια συγκεκριμένη ροή πέφτει κάτω από ένα προκαθορισμένο κατώφλι g (που εξαρτάται, όμως, από τη φύση της εφαρμογής) τότε ο κόμβος δεν πραγματοποιεί την επαναμετάδοση του μηνύματος. Έτσι η συνθήκη για τη μετάδοση είναι:

$U_k \geq g > 0$	(4.8)



Σχήμα 4.4: Συνάρτηση αξιολόγησης, συνιστώσα ενέργειας

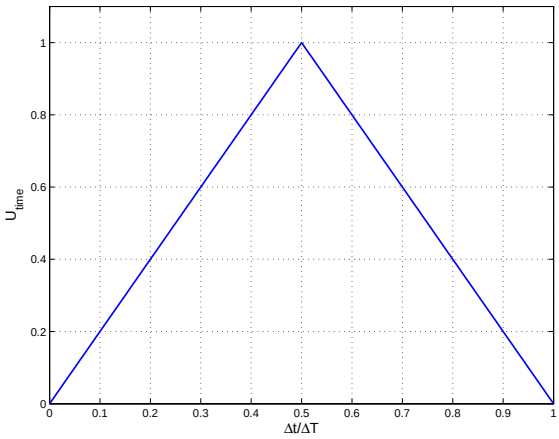
Οι τρεις συνιστώσες της συνάρτησης χρησιμότητας παρέχουν μια πλήρη σύνοψη της τρέχουσας κατάστασης του δικτύου, δηλαδή η ενεργειακή συνιστώσα αντικατοπτρίζει την κατάσταση του κόμβου, η συνιστώσα σφάλματος μας πληροφορεί για την κατάσταση μιας ροής δεδομένων και η χρονική συνιστώσα αντικατοπτρίζει το χρονισμό της τοπολογίας ως σύνολο.



Σχήμα 4.5: Συνάρτηση αξιολόγησης, συνιστώσα σφάλματος

Η συνάρτηση αξιολόγησης μπορεί να αποδώσει την αναγκαιότητα μιας μετάδοσης λαμβάνοντας υπόψη ποικίλα κριτήρια με πολλούς διαφορετικούς τρόπους. Μεταβάλλοντας τιμές των βαρών w_i μπορούμε να υλοποιήσουμε μια εφαρμογή, η οποία δίνει πιο μεγάλη σημασία στην επιμήκυνση του χρόνου ζωής του δικτύου από ότι στην ακρίβεια, ή το ανάποδο. Επίσης, η παράμετρος $err_{threshold}$ μας επιτρέπει να αυξήσουμε ή να μειώσουμε την ευαισθησία της συνάρτησης στο σφάλμα ανάλογα με την εφαρμογή. Η τελευταία παράμετρος που μπορεί να μεταβληθεί είναι το κατώφλι της τιμής της συνάρτησης αξιολόγησης. Ανάλογα με την κίνηση του δικτύου και με τους πόρους που μπορούμε να χρησιμοποιήσουμε, είναι δυνατό να επιλέξουμε μια τιμή μεγάλη αν θέλουμε να μειώσουμε το πλήθος των μεταδόσεων, ή πιο μικρή, εάν έχουμε τη

δυνατότητα για περισσότερες μεταδόσεις χωρίς να επιβαρυνθεί το δίκτυο με κίνηση που δεν μπορεί να εξυπηρετήσει.



Σχήμα 4.6: Συνάρτηση αξιολόγησης, συνιστώσα χρόνου

Μέχρι τώρα δεν έχει γίνει ιδιαίτερη μνεία στη χρονική συνιστώσα της συνάρτησης αξιολόγησης. Παρ'όλα αυτά, η χρονική συνιστώσα εισάγει μια πολύ ενδιαφέρουσα ιδέα στη λήψη απόφασης κατά τη διαδικασία της δρομολόγησης. Συγκεκριμένα, λαμβάνοντας υπ'όψη ότι στην αρχιτεκτονική που προτείνεται υιοθετείται η λογική των Heart-Beat μηνυμάτων, είμαστε σε θέση να γνωρίζουμε ότι σε τακτά χρονικά διαστήματα κάποιο μήνυμα αυτού του τύπου αναμένεται. Η χρονική συνιστώσα μας προτρέπει να αποφύγουμε μια αποστολή που εισάγει ανεκτό σφάλμα αν αυτή πρόκειται να πραγματοποιηθεί – χρονικά – κοντά στην αποστολή κάποιου Heart-Beat μηνύματος. Έτσι αναδεικνύεται και ο ρόλος αυτών των μηνυμάτων που δεν είναι άλλος από τη διάδοση εύρωστων δεδομένων στο δίκτυο.

4.5 Μηχανισμός αποστολής δεδομένων

Η συνάρτηση αξιολόγησης που αναπτύχθηκε στην προηγούμενη ενότητα αξιολογεί τη χρησιμότητα μετάδοσης της τρέχουσας τιμής μιας συγκεκριμένης ροής. Σε μια δενδρική δομή, όμως, οι ενδιάμεσοι κόμβοι (επικοινωνίας) εξυπηρετούν παραπάνω από μία ροή, με συνέπεια να αξιολογούν την τρέχουσα τιμή της κάθε ροής χωριστά.

Πριν προωθηθούν τα μηνύματα από έναν κόμβο επικοινωνίας προς τον επόμενο με τελικό αποδέκτη τον κόμβο-καταβόθρα αξιολογούνται όλες οι μεταδόσεις και πραγματοποιούνται μόνο αυτές που προκύπτει από την συνάρτηση αξιολόγησης τιμή μεγαλύτερη από το κατώφλι. Έτσι οι τιμές που κρίνονται ως αναγκαίες προς μετάδοση συναθροίζονται σε ένα μήνυμα μαζί με κάποια μεταδεδομένα για την ορθή διαχείριση από τον κόμβο αποδέκτη. Εκτός από το ενεργειακό κέρδος, που είναι συνέπεια της μη αποστολής κάποιων τιμών, έχουμε το επιπλέον κέρδος της συνάθροισης των δεδομένων. Αν τα μηνύματα έφευγαν χωριστά τότε για κάθε ροή δεδομένων που εξυπηρετεί ο κάθε κόμβος, σε περίπτωση που έπρεπε να προωθήσει την τρέχουσα τιμή, θα έπρεπε να «πληρώσει» το overhead του κάθε μηνύματος, κάτι που κάνει μόνο μία φορά για όλες τις ΡΔ που εξυπηρετεί.

Το μόνο που απομένει να εξηγηθεί στο σημείο αυτό είναι τι γίνεται σε περίπτωση που ένας ενδιάμεσος κόμβος δε λάβει κάποιες τιμές. Ποιες τιμές αναλαμβάνει να προωθήσει και πώς προκύπτουν.

4.6 Μηχανισμός αναπλήρωσης δεδομένων

Σε αυτήν την ενότητα θα αναφερθούμε στη διαδικασία που χρησιμοποιούν οι κόμβοι για να εκτιμήσουν τα δεδομένα που δεν έλαβαν λόγω της απόφασης της μη αποστολής δεδομένων από τους αντίστοιχους μηχανισμούς TCM κόμβων χαμηλότερου επιπέδου.

Ο μηχανισμός TCM υπολογίζει, σε κάθε κόμβο επικοινωνίας ή στον κόμβο καταβόθρα, τις τιμές που αποτελούν την εκάστοτε τρέχουσα τιμή (δηλαδή την τιμή που θα διαδοθεί από το τρέχον σημείο του δικτύου και πέρα ως τρέχουσα τιμή της ΡΔ) σε περίπτωση που δε ληφθεί. Πιο συγκεκριμένα, με τη βοήθεια του καθολικού μηχανισμού συγχρονισμού της αρχιτεκτονικής που προαναφέρθηκε, καθορίζεται ένα συγκεκριμένο παράθυρο μέσα στο οποίο ο κάθε κόμβος περιμένει να λάβει τα μηνύματα με τις μετρήσεις. Εάν αυτό δε συμβεί, δηλαδή εάν ο κόμβος δε λάβει κάποιο μήνυμα μέσα στο προκαθορισμένο παράθυρο υπολογίζει την τιμή αυτή με χρήση της προκαθορισμένης μεθόδου παρεμβολής. Στα παραδείγματα που πραγματοποιήθηκαν η παρεμβολή είναι γραμμική.

4.7 Μοντέλο αποφόρτισης και επαναφόρτισης

Σημαντικός παράγοντας της συνάρτησης αξιολόγησης είναι η ενεργειακή κατάσταση του κόμβου. Συνεπώς, κρίθηκε απαραίτητο να ενσωματωθεί ένα ενεργειακό μοντέλο αποφόρτισης. Επίσης σχεδιάστηκε ένα μοντέλο επαναφόρτισης το οποίο όμως αρχικά δε χρησιμοποιήθηκε. Προκειμένου να ποσοτικοποιηθεί η ενέργεια που καταναλώνεται ανά περίπτωση έγινε μια ειδίκευση. Οι τιμές που αναφέρονται παρακάτω είναι τιμές που αντιστοιχούν στον κόμβο τύπου mica2. Mica2 είναι γενικά ο κόμβος που υποθέτουμε ότι χρησιμοποιείται σε κάθε προσομοίωση.

Οι αρχές που διέπουν το μοντέλο αποφόρτισης είναι οι παρακάτω: αρχικά, κάθε εντολή που εκτελεί ο επεξεργαστής έχει ένα καθορισμένο ενεργειακό κόστος ίσο με 4 nJ/instruction, σύμφωνα με τη βιβλιογραφία[2]. Επιπλέον, η αποστολή ενός μηνύματος έχει ένα σημαντικό ενεργειακό κόστος μετρούμενο σε ενέργεια ανά bit που αποστέλλεται. Συγκεκριμένα, η κατανάλωση είναι ίση με 720 nJ/bit και, αντίστοιχα, η ενέργεια που καταναλώνεται κατά την παραλαβή ενός μηνύματος είναι 110 nJ/bit. Όπως, αναφέρθηκε και νωρίτερα, ο επεξεργαστής καταναλώνει ενέργεια όση ώρα είναι σε λειτουργία άσχετα από το εάν εκτελεί κάποια συγκεκριμένη εντολή ή όχι. Ο επεξεργαστής ενός κόμβου μπορεί να βρεθεί σε τρεις διαφορετικές καταστάσεις. Μπορεί να είναι σε κατάσταση αδράνειας (idle) οπότε και καταναλώνονται 9.6 mW, μπορεί να είναι σε κατάσταση αναμονής (stand-by) οπότε η κατανάλωση είναι 648 μW, ή σε κατάσταση εξοικόμησης ενέργειας οπότε καταναλώνονται 330 μW. Επίσης, η μετάβαση από τη μια κατάσταση στην άλλη απαιτεί την κατανάλωση ενός μικρού ποσού ενέργειας, συγκεκριμένα 62 nJ ανά αλλαγή κατάστασης.

Όπως γίνεται αντιληπτό, είναι πολύ σημαντικό να γίνεται προσεκτική χρήση του επεξεργαστή και να αποφεύγεται η παραμονή του επεξεργαστή σε κατάσταση αναμονής, αφού η κατανάλωση είναι 15 με 30 φορές μεγαλύτερη από το αν επιλεγεί μια πιο οικονομική κατάσταση λειτουργίας.

Η μοντελοποίηση που υιοθετείται στην προτεινόμενη αρχιτεκτονική προϋποθέτει τη γνώση του αρχικού ενεργειακού αποθέματος. Καθώς εκτελείται ο κώδικας ενημερώνεται η τιμή του τρέχοντος ενεργειακού αποθέματος αφαιρώντας την ενέργεια των διεργασιών που λαμβάνουν χώρα. Επίσης, σε τακτά χρονικά διαστήματα αφαιρείται ενέργεια που έχει να κάνει με το χρόνο παραμονής σε λειτουργία του εκάστοτε κόμβου και εξαρτάται επίσης από την κατάσταση στην οποία παρέμεινε ο κόμβος.

[2] Για πιο λεπτομερή περιγραφή του ενεργειακού κόστους βλέπε Παράρτημα Β.

Το μοντέλο επαναφόρτισης δεν έχει υλοποιηθεί πλήρως και είναι μια μελλοντική πρόταση για περαιτέρω έρευνα στη συμπεριφορά ενός δικτύου που έχει και τη δυνατότητα επαναφόρτισης. Πιο συγκεκριμένα, κάθε κόμβος μπορεί να έχει έναν ηλιακό συσσωρευτή πάνω του. Αυτό το γεγονός του δίνει τη δυνατότητα να χρησιμοποιεί την ηλιακή ενέργεια, όταν αυτή υπάρχει, για να φορτίσει την μπαταρία του. Εκτός όμως από την ηλιακή ενέργεια μπορεί να χρησιμοποιηθεί τόσο η ενέργεια από ηλεκτρικούς λαμπτήρες όσο και ενέργεια σε άλλη μορφή, π.χ. κινητική ή χημική, που δε θα συζητηθεί όμως, στο παρόν κείμενο. Σύμφωνα με τη βιβλιογραφία η ηλιακή ενέργεια κατά τη διάρκεια μιας καλής, αίθριας μέρας φορτίζει με 15 mW/cm², σε μια συννεφιασμένη μέρα η φόρτιση είναι 150 μW/cm² ενώ σε εσωτερικό χώρο είναι 6 μW/cm² και 570 μW/cm² εάν χρησιμοποιηθεί λάμπα γραφείου. Σε ένα πλήρες μοντέλο επαναφόρτισης οι παραπάνω τιμές πρέπει να προστίθενται στο τρέχον ενεργειακό απόθεμα με κάποια πιθανότητα, ενσωματώνοντας σε αυτήν την πιθανότητα το ενδεχόμενο να είναι μέρα, νύχτα καθώς και το εάν το δίκτυο είναι εσωτερικού ή εξωτερικού χώρου. Επίσης, μπορούμε να υποθέσουμε χωρίς μεγάλη απόκλιση ότι το εμβαδόν του ηλιακού συλλέκτη θα είναι γύρω στο 1 cm², οπότε και οι παραπάνω τιμές απλοποιούνται μόνο σε Watt. Στην υλοποίηση που παρουσιάζεται στο επόμενο κεφάλαιο έχει ενσωματωθεί το μοντέλο αποφόρτισης αλλά όχι και το μοντέλο επαναφόρτισης που, περιληπτικά, αναφέρθηκε παραπάνω.

ΚΕΦΑΛΑΙΟ 5

ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΜΟΝΤΕΛΟΥ ΣΕ TinyOS, NesC, TOSSIM

5.1 Components που χρησιμοποιούνται στην υλοποίηση

Όπως έχει αναφερθεί νωρίτερα, κάθε πρόγραμμα σε TinyOS μπορεί να χρησιμοποιήσει components που είναι ήδη έτοιμα και παρέχονται από την ίδια την πλατφόρμα. Αυτό γίνεται για να είναι εύκολο να προγραμματιστούν κάποιες υπηρεσίες που θεωρούνται απαραίτητες σε πολλά δίκτυα αισθητήρων. Μέσω των components αυτών μπορούμε να παρέχουμε στην εφαρμογή μας τη χρήση κάποιων έτοιμων interfaces. Τα components που χρησιμοποιούνται είναι τα εξής: Main, TimerC, GenericComm, TinyAlloc, RandomLFSR και το McmfaM, το οποίο είναι το component που υλοποιεί την προτεινόμενη αρχιτεκτονική. Στο αρχείο Mcmfa.nc πραγματοποιείται η συναρμολόγηση των components συνδέοντας τα interfaces που είναι απαραίτητα. Οι αρχές που διέπουν το σχεδιασμό και τη συναρμολόγηση των components έχουν περιγραφεί στο κεφάλαιο 3. Στο αρχείο McmfaM.nc βρίσκεται ο κώδικας που υλοποιεί τα commands των interfaces που χρησιμοποιούνται και τα events των interfaces που παρέχονται.

Το component Main χρησιμοποιείται για να μπορεί να κληθεί το πρόγραμμα από τον scheduler του TinyOS και να εκκινήσει η εφαρμογή. Το component αυτό είναι το πρώτο που καλείται σε μια TinyOS εφαρμογή. Συγκεκριμένα, η πρώτη εντολή που εκτελείται είναι η Main.StdControl.init() και ακολουθείται από την Main.StdControl.start(). Συνεπώς, κάθε εφαρμογή TinyOS πρέπει να έχει ένα τέτοιο component στο configuration του. Το StdControl είναι ένα κοινό interface που χρησιμοποιείται για να αρχικοποιεί και να εκκινεί τα διάφορα TinyOS components. Το component TimerC χρησιμοποιείται για να μας παρέχει τη δυνατότητα χρήσης χρονομέτρη. Συγκεκριμένα μπορούμε να ενεργοποιούμε το χρονομέτρη σύμφωνα με τις ανάγκες μας (το χρονικό διάστημα της αντίστροφης μέτρησης και το πλήθος τον φορών που θα λειτουργήσει ο χρονομέτρης). Επιπλέον, όταν τελειώσει το επιθυμητό χρονικό διάστημα, εκκινεί η εκτέλεση ενός event, Timer.fired(), το οποίο αναλαμβάνουμε να υλοποιήσουμε. Το συγκεκριμένο event είναι πολύ σημαντικό για την εφαρμογή αφού η αρχιτεκτονική που προτείνεται βασίζεται στο συγχρονισμό ενεργειών, κάτι που επιτυγχάνεται με χρήση του χρονομέτρη. Η επικοινωνία μεταξύ των κόμβων επιτυγχάνεται με χρήση του component GenericComm. Το component αυτό παρέχει δύο interfaces, το SendMsg και το ReceiveMsg, μέσω των οποίων μπορούμε να αποστείλουμε και παραλάβουμε μηνύματα. Τα events SendMsg.sendDone() και ReceiveMsg.receive() υλοποιούνται στην εφαρμογή προκειμένου να γίνει η διαχείριση της πληροφορίας που αποστέλλεται και λαμβάνεται αντίστοιχα. Τέλος, τα components TinyAlloc και RandomLFSR έχουν βοηθητικό ρόλο. Το πρώτο χρησιμοποιείται για να μπορέσουμε να δεσμεύσουμε δυναμικά μνήμη, ενώ το δεύτερο χρησιμοποιείται στο μοντέλο επαναφόρτισης, όπου χρειαζόμαστε μια γεννήτρια ψευδοτυχαίων αριθμών.

5.2 Σχόλια για το component Mcmfa

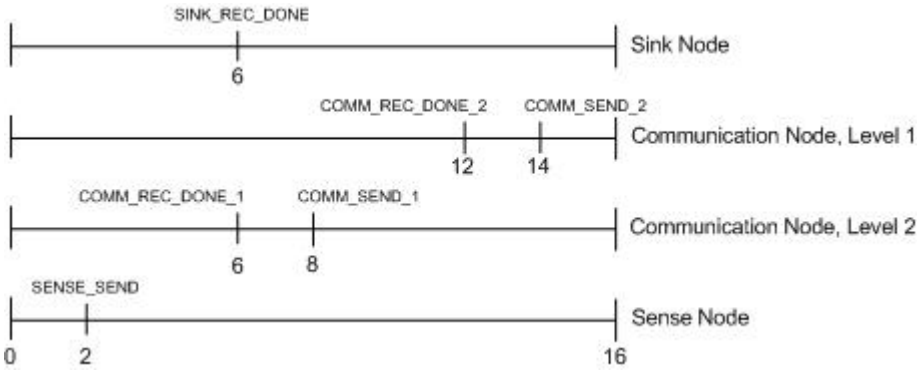
Το component αυτό είναι το βασικό κομμάτι κώδικα που υλοποιήθηκε. Ο σκοπός του είναι να παράγει αποτελέσματα μετρήσεων μέσω προσομοιώσεων με χρήση του TOSSIM και όχι να τρέξει αυτούσιος σε κόμβους. Ο κώδικας, γενικά, πληροί τις προδιαγραφές που απαιτούνται για να τρέξει σε κόμβους, αλλά χρειάζεται κάποιες τροποποιήσεις όσον αφορά το πρωτόκολλο δρομολόγησής του και τον αλγόριθμο αρχικοποίησης. Ο κώδικας του component αυτού χωρίζεται σε τρία αρχεία. Στο

configuration αρχείο, στο οποίο γίνονται οι συναρμολογήσεις των components που προαναφέρθηκαν, στο module αρχείο στο οποίο υλοποιείται ο κώδικας της εφαρμογής και στο header αρχείο στο οποίο γίνονται κάποιες δηλώσεις.

5.2.1 Σχεδιαστικές επιλογές

Η ιδιαίτερη σημασία του συγχρονισμού στην αρχιτεκτονική οδήγησε σε χρήση ενός χρονομέτρη, σύμφωνα με τον οποίο επιτελούνται οι διάφορες διεργασίες σε συγκεκριμένα χρονικά βήματα. Ο συγχρονισμός είναι απαραίτητος ώστε η αισθητήρια λειτουργία των κόμβων, οι αποστολές και οι λήψεις μηνυμάτων να γίνονται σε διακριτές χρονικές στιγμές με σκοπό οι ροές δεδομένων να προχωρούν προς τον προορισμό (κόμβος καταβόθρα) απρόσκοπτα. Οι διάφορες διεργασίες του κόμβου θεωρείται ότι έχουν συγκεκριμένη χρονική διάρκεια και έτσι κάθε κόμβος γνωρίζει πόσο χρόνο έχει στη διάθεση του για να λάβει, να στείλει ή να επεξεργαστεί κάποιο μήνυμα.

Ο κύκλος λειτουργίας ενός κόμβου αποκαλείται STEP. Μέσα σε αυτόν ένας κόμβος μπορεί να πάρει μια μέτρηση (αν είναι αισθητήριος κόμβος), να λάβει μετρήσεις (αν είναι κόμβος επικοινωνίας ή ο κόμβος καταβόθρα) και να αποστείλει μετρήσεις (αν είναι αισθητήριος κόμβος ή κόμβος επικοινωνίας). Ένα STEP αποτελείται από ένα πλήθος CLICK, το κάθε ένα από τα οποία θεωρείται το στοιχειώδες χρονικό διάστημα. Οι τιμές ελέγχονται από τον προγραμματιστή με χρήση defines. Στις εκτελέσεις που έγιναν, ένα STEP αποτελείται από 16 CLICK και κάθε CLICK έχει διάρκεια 500ms. Επίσης, η διάρκεια του CLICK είναι και η χρονική διαφορά μεταξύ διαδοχικών πυροδοτήσεων του χρονομέτρη. Μέσα στον κύκλο λειτουργίας, ο κόμβος ανάλογα με τον τύπο του, επιτελεί τις λειτουργίες του σε διαφορετικά χρονικά παράθυρα, όπως φαίνεται στο Σχήμα 5.1.



Σχήμα 5.1: Παράδειγμα κύκλου λειτουργίας δικτύου δενδρικής δομής (Σχήμα 4.1)

Τη χρονική στιγμή SINK_REC_DONE, ο κόμβος-βάση θεωρεί πως όλα τα μηνύματα που αναμένει στον τρέχον κύκλο έχουν ληφθεί, οπότε αρχίζει τη διαδικασία επεξεργασίας των δεδομένων που έχουν εισέλθει σε αυτόν. Τη χρονική στιγμή COMM_REC_DONE_2 ή COMM_REC_DONE_1 ο αντίστοιχος κόμβος επικοινωνίας θεωρεί πως έχει λάβει όλα τα μηνύματα του τρέχοντος κύκλου οπότε αρχίζει την επεξεργασία των ληφθέντων πακέτων με σκοπό στη χρονική στιγμή COMM_SEND_2 ή COMM_SEND_1, να αποσταλούν τα αντίστοιχα πακέτα στο επόμενο επίπεδο, εφόσον κάτι τέτοιο αποφασιστεί από τη συνάρτηση αξιολόγησης. Η παρουσία δυο διαφορετικών χρονικών στιγμών (δύο για την ολοκλήρωση των παραλαβών και δύο για την αποστολή) κρίθηκε αναγκαία, γιατί, σε διαφορετική περίπτωση, ένας κόμβος θα προσπαθούσε να στείλει τη στιγμή που θα έπρεπε και να λάβει, προκαλώντας έτσι ενδεχόμενες απώλειες δεδομένων. Η εναλλαγή των τιμών αυτών γίνεται για όλο το βάθος της δενδρικής δομής του δικτύου.

Άλλες παράμετροι που χρησιμοποιούνται για τη ρύθμιση της λειτουργίας του προγράμματος αναφέρονται στα γενικά χαρακτηριστικά του προγράμματος. Η σταθερά HEART_BEAT_FREQ καθορίζει κάθε πόσα STEPS αποστέλλεται η αλληλουχία των Heart-Beat μηνυμάτων. Το πλήθος των μηνυμάτων αυτών καθορίζεται από το βαθμό παρεμβολής που χρησιμοποιείται και πιο συγκεκριμένα στις εκτελέσεις που πραγματοποιήθηκαν είναι ίσο με 2. Η τιμή αυτή ελέγχεται από τη σταθερά EXTR_DEGREE. Μια σταθερά που ελέγχει εάν όλα τα μηνύματα θα αποσταλλούν σε μορφή Heart-Beat είναι η σταθερά HB_ALL η οποία χρησιμοποιήθηκε για να προκύψουν τα συγκριτικά αποτελέσματα που θα παρουσιαστούν σε επόμενη ενότητα. Ακόμη, η σταθερά OVERHEAD αφορά το επιπλέον κόστος της αποστολής ενός μηνύματος. Πέρα από την καθ'αυτό πληροφορία ενός μηνύματος, ο κόμβος που στέλνει το μήνυμα προωθεί και 7 bytes μεταδεδομένων σε επίπεδο υλοποίησης λειτουργικού συστήματος.

Στο αρχείο McmfaM.nc ορίζονται αρκετές επιπλέον σταθερές που σχετίζονται με το μοντέλο αποφόρτισης και το (ανενεργό) μοντέλο επαναφόρτισης. Πιο συγκεκριμένα, οι υπολογισμοί της κατανάλωσης ενέργειας βασίζονται στο ρεύμα στον επεξεργαστή το οποίο εξαρτάται από την κατάσταση λειτουργίας και στην τάση που προέρχεται από την πηγή τροφοδοσίας. Κάθε φορά που ενεργοποιείται το event του χρονομέτρη, σε περιπτώσεις όπως, όταν γίνεται επεξεργασία, όταν λαμβάνουμε ή στέλνουμε μήνυμα, η ενέργεια που έχει ξοδευτεί υπολογίζεται (συνάρτηση update_energy) και αφαιρείται από το ενεργειακό απόθεμα (CUR_E). Για τον υπολογισμό της κατανάλωσης χρησιμοποιούνται οι παρακάτω σταθερές:

- EPI: Κατανάλωση ενέργειας για την εκτέλεση μιας εντολής ενός κύκλου στον επεξεργαστή (nJ/εντολή)
- EPBS: Κατανάλωση για την αποστολή ενός bit πληροφορίας (nJ/bit)
- EPBR: Κατανάλωση για τη λήψη ενός bit πληροφορίας (nJ/bit)
- EPSI: Κατανάλωση όταν το CPU βρίσκεται σε κατάσταση idle (nJ/sec)
- EPSSB: Κατανάλωση όταν ο επεξεργαστής βρίσκεται σε κατάσταση stand-by (nJ/sec)
- EPMC: Ενεργειακό κόστος για την εναλλαγή από την μια ενεργειακή κατάσταση στην άλλη (nJ/αλλαγή κατάστασης).

Οι σταθερές EPSI, EPSSB και EPMC συνδυάζονται με τις μεταβλητές IdlePC και SBPC, οι οποίες εκφράζουν το ποσοστό της λειτουργίας του STEP όπου ο επεξεργαστής είναι idle και stand-by αντίστοιχα. Με το συνδυασμό αυτών των σταθερών μπορούμε να υπολογίζουμε την ενεργειακή κατανάλωση του επεξεργαστή σε κάθε STEP, θεωρώντας ότι για ένα ποσοστό της διάρκειας του STEP, ο επεξεργαστής είναι σε κατάσταση idle (το διάστημα που κάνει "listening" πριν λάβει μήνυμα) ενώ για ένα άλλο ποσοστό θεωρείται ότι είναι stand-by (ο επεξεργαστής έχει μικρή κατανάλωση – είναι "ανενεργός"). Με το διαχωρισμό του χρόνου λειτουργίας του επεξεργαστή σε idle και stand-by μπορούμε να έχουμε στατιστικά στοιχεία που αφορούν την κατανάλωση ενέργειας πολύ κοντά στην πραγματική συμπεριφορά του επεξεργαστή. Όσον αφορά το μοντέλο επαναφόρτισης υπάρχει ένα flag το ENABLE_CHARGING με το οποίο μπορούμε να ενεργοποιούμε και να απενεργοποιούμε το μοντέλο ανάλογα με την προσομοίωση. Ακόμη, χρησιμοποιούνται στοιχεία από πραγματικές μετρήσεις για το ενεργειακό κέρδος που αποφέρει το μοντέλο επαναφόρτισης ανάλογα με το αν ο αισθητήρας βρίσκεται σε ανοιχτό χώρο (CHARGE_DIRECT_SUN για ηλιόλουστη μέρα, CHARGE_CLOUDY_DAY για συννεφιασμένη μέρα) ή σε κλειστό χώρο (CHARGE_OFFICE_DESK για φόρτιση από σε ένα γραφείο από το φως που μπαίνει

από το παράθυρο και CHARGE_DESK_LAMP για φόρτιση από λάμπα γραφείου). Τέλος, τόσο στο μοντέλο επαναφόρτισης, όσο και στους υπολογισμούς για την αποφόρτιση, πολύ σημαντικές είναι οι δύο σταθερές NUMOF_READINGS (αριθμός μετρήσεων που θα προσομοιωθούν) και NUMOF_MINUTES (πραγματικός χρόνος σε λεπτά που θα προσομοιώσουμε). Με χρήση αυτών των σταθερών μπορούμε να μελετήσουμε τη συμπεριφορά της αρχιτεκτονικής για μεγάλο χρονικό διάστημα πραγματικού χρόνου σε μικρό χρόνο προσομοίωσης. Αυτό επιτυγχάνεται με τον υπολογισιμό ενός χρονικού παράγοντα με τη βοήθεια των δύο πραπάνω σταθερών που ανάγει το χρόνο προσμοίωσης σε πολύ μεγαλύτερα διαστήματα πραγματικού χρόνου και έτσι μπορούμε να εξάγουμε αποτελέσματα με ακρίβεια. Οι παραπάνω τιμές, είναι εύκολο να αλλάξουν, αφού είναι παράμετροι της λειτουργίας του προγράμματος, και έτσι να έχουμε διαφορετικά στατιστικά ανάλογα με τον τύπο κόμβου τον οποίο προσομοιώνουμε.

Εκτός από τις σταθερές που αναφέραμε παραπάνω, χρησιμοποιούνται και κάποιες λογικές μεταβλητές (flags) που είναι απαραίτητες για το συγχρονισμό της αρχιτεκτονικής. Πιο συγκεκριμένα, η μεταβλητή senseSend αφορά τους αισθητήριους κόμβους και καθορίζει πότε μπορούμε να πραγματοποιήσουμε μια μέτρηση. Αν η τιμή του είναι TRUE σημαίνει πως μπορούμε να πραγματοποιήσουμε μια μέτρηση. Σε διαφορετική περίπτωση (FALSE) δεν μπορούμε να αρχίσουμε μια νέα μέτρηση, αφού κάποια άλλη μέτρηση βρίσκεται σε εξέλιξη ή ο αισθητήριος κόμβος δεν έχει προλάβει να προωθήσει την προηγούμενη μέτρηση. Δύο ακόμα σημαντικές λογικές μεταβλητές είναι οι commReceiveMode (για ενδιάμεσους κόμβους) και η sinkReceiveMode (για τον κόμβο δεξαμενή) που έχουν παρόμοια λογική. Η commReceiveMode όταν είναι TRUE σημαίνει ότι ο ενδιάμεσος κόμβος δεν πραγματοποιεί κάποια επεξεργασία ή δεν εκκρεμεί κάποια αποστολή μηνύματος από αυτόν, οπότε μπορεί να λάβει μηνύματα από κόμβους προγόνους στην ιεραρχία. Όταν το flag αυτό γίνει FALSE, ξεκινά η επεξεργασία των δεδομένων που έχουν ληφθεί. Αν κάποια μηνύματα δεν έχουν φτάσει, ενώ θα έπρεπε, θεωρούμε πως έχουν χαθεί και προσεγγίζουμε τις τιμές τους με παρεμβολή. Αν ληφθούν μηνύματα όταν το flag είναι FALSE, τότε δεν τα λαμβάνουμε υπόψη. Αντίστοιχα, δουλεύει και το flag sinkReceiveMode μόνο που σηματοδοτεί το παράθυρο για την παραλαβή μηνυμάτων που προορίζονται για τον κόμβο δεξαμενή.

Επιπλέον, κρίνεται σκόπιμο να αναφερθούμε και στις μεταβλητές ERR_THRESHOLD, UT_THRESHOLD, W1, W2 και W3. Η ERR_THRESHOLD καθορίζει το κατώφλι σφάλματος για το error component της συνάρτησης αξιολόγησης. Το UT_THRESHOLD θέτει την τιμή του κατωφλίου για την ίδια τη συνάρτηση αξιολόγησης, ενώ τα W1, W2 και W3 αποτελούν τα βάρη για κάθε component της συνάρτησης και οι τιμές τους πρέπει να καθορίζονται ανάλογα με το σκοπό της εφαρμογής, αλλά και το φαινόμενο το οποίο εξετάζουμε. Τέλος, οι μεταβλητές E_MAX και CUR_E χρησιμοποιούνται για το ενεργειακό μοντέλο. Η πρώτη μεταβλητή αποτελεί το αρχικό ενεργειακό απόθεμα της πηγής και η δεύτερη εκφράζει την ενέργεια του κόμβου κάθε στιγμή.

5.2.2 Κυριότερες συναρτήσεις

Ο κώδικας που προσομοιώνει την αρχιτεκτονική βρίσκεται στο module του component στο αρχείο McmfaM.nc. Στη συνέχεια, θα παραθέσουμε στοιχεία που αφορούν τις κυριότερες συναρτήσεις που δομούν το πρόγραμμα της προσομοίωσης. Αυτές είναι οι update_energy, extrap_val και eval_utility και ο κώδικας που υλοποιεί το event fired() του χρονομέτρη και το event receive() του interface ReceiveMsg. Οι τρεις πρώτες έχουν βοηθητικό ρόλο στους υπολογισμούς, ενώ το event fired() του χρονομέτρη είναι το τμήμα του κώδικα που αποτελεί τη ραχοκοκκαλιά του προγράμματος προσομοίωσης.

α) *void update_energy (int amount, uint8_t type)*: Η συνάρτηση αυτή υλοποιεί τους υπολογισμούς για την αποφόρτιση και την επαναφόρτιση κάθε κόμβου. Χρησιμοποιείται για την ενημέρωση μετά από κάθε υπολογιστικό βήμα της μεταβλητής CUR_E για να έχουμε το τρέχον ενεργειακό απόθεμα του κόμβου. Δέχεται δυο ορίσματα, το πρώτο εκ των οποίων, βοηθά στον υπολογισμό του ποσού ενέργειας που θα αφαιρέσουμε ή θα προσθέσουμε στο απόθεμα και το δεύτερο προσδιορίζει την αιτία που οδήγησε στην αλλαγή της ενέργειας. Το κόστος λειτουργίας του επεξεργαστή στις διάφορες καταστάσεις, αλλά και το κόστος ανάλογα με τις διεργασίες που πραγματοποιούμε, αναφέρθηκαν προηγουμένως. Όταν το *type* έχει τιμή T_INSTR τότε ο επεξεργαστής είναι active και υπολογίζουμε την κατανάλωση για κάθε υπολογιστική πράξη που εκτελείται ανάλογα με το πλήθος των εντολών που προσδιορίζονται από το *amount*. Οι περιπτώσεις T_RECEIVE και T_SEND αναφέρονται για την κατανάλωση σε περίπτωση αποστολής και στην περίπτωση λήψης μηνύματος αντίστοιχα. Εδώ, το *amount* προσδιορίζει το πλήθος των bytes που περιλαμβάνει το μήνυμα και αυτό αυξάνεται με το OVERHEAD κάθε αποστολής. Ακόμη, με το *type* T_IDLE προσδιορίζεται η κατανάλωση όταν ο επεξεργαστής είναι σε κατάσταση idle ή σε κατάσταση stand-by (ένα ποσοστό του χρόνου για την μια κατάσταση και ένα ποσοστό στην άλλη). Το *amount* σε αυτή την περίπτωση εκφράζει τον χρόνο για τον οποίο προσομοιώνουμε τον επεξεργαστή στις δύο προαναφερθείσες καταστάσεις. Με τις παραπάνω τιμές για το *type* έχουμε μια πλήρη απεικόνιση της αποφόρτισης της πηγής. Η ίδια συνάρτηση που υπολογίζει την αποφόρτιση μπορεί να χρησιμοποιείται και όταν το μοντέλο επαναφόρτισης είναι ενεργοποιημένο. Πιο συγκεκριμένα, όταν το *type* έχει τιμή T_CHARGE τότε έχει ενεργοποιηθεί το μοντέλο και με βάση – μεταξύ άλλων και – την τιμή του *amount* προσδιορίζεται η αύξηση του ενεργειακού αποθέματος. Εδώ, σημαντικό ρόλο διαδραματίζουν οι μεταβλητές NUMOF_READINGS και NUMOF_MINUTES με τις οποίες υπολογίζεται μια καλή εκτίμηση του χρόνου που διαρκεί η προσομοίωση και έτσι μπορούμε να γνωρίζουμε ποιό ποσοστό της προσομοίωσης πρέπει να κερδίζει ο κόμβος ενέργεια (είναι μέρα) και ποιο ποσοστό δεν κερδίζει (είναι νύχτα). Επιπλέον, το ποσό της ενέργειας που κερδίζει ο κόμβος σε κάθε ενεργοποίηση της συνάρτησης εκτιμάται με μια ομοιόμορφη κατανομή.

β) *double extrap_val(Extr_Data data)*: Η συνάρτηση αυτή χρησιμοποιείται για την εκτίμηση μιας μέτρησης, στην περίπτωση που δε λάβαμε μήνυμα για αυτή, αλλά και στους υπολογισμούς για το αν θα σταλεί κάποια τιμή ή όχι. Δέχεται ως όρισμα μια *struct* τύπου *Extr_Data* στην οποία έχουμε αποθηκεύσει τις τιμές που μας χρειάζονται για την προσέγγιση. Στη συγκεκριμένη προσομοίωση χρησιμοποιείται γραμμική παρεμβολή και γι'αυτό πρέπει να έχουμε κρατήσει πληροφορία για δύο προηγούμενες τιμές. Η συνάρτηση επιστρέφει την τιμή που θα προκύψει από την προσέγγιση. Γενικά, αντί για γραμμική παρεμβολή θα μπορούσε να χρησιμοποιηθεί κάποια άλλη αριθμητική μέθοδος (πολυώνυμο μεγαλύτερου βαθμού, πεπερασμένες διαφορές, κυβικές splines κ.ά) ανάλογα με την εφαρμογή που σχεδιάζεται και το φαινόμενο που εξετάζουμε. Αν χρειαστεί να χρησιμοποιηθεί διαφορετική μέθοδος τότε αρκεί να τροποποιηθεί κατάλληλα η παρούσα συνάρτηση καθώς και η *struct* που δέχεται ως όρισμα ώστε να έχει τις κατάλληλες τιμές.

γ) *uint8_t eval_utility(float err)*: Η συνάρτηση αυτή χρησιμοποιείται για τον υπολογισμό της συνάρτησης αξιολόγησης. Δέχεται ως όρισμα έναν float που εκφράζει το ποσοστιαίο σφάλμα για τη συνιστώσα σφάλματος. Υπολογίζει κάθε συνιστώσα χωριστά, και στη συνέχεια, συνδυάζει και τις τρεις με κάποια βάρη ώστε να προκύψει το τελικό αποτέλεσμα της συνάρτησης αξιολόγησης. Η συνιστώσα της ενέργειας υπολογίζεται με βάση το τρέχον ενεργειακό απόθεμα. Αν η τιμή *err* που λάβαμε ως όρισμα στη συνάρτηση είναι μεγαλύτερη από το ERR_THRESHOLD (κατώφλι σφάλματος της συνιστώσα σφάλματος) τότε ανεξάρτητα από την τιμή των άλλων συνιστωσών της

συνάρτησης αξιολόγησης, αποστέλλουμε την τρέχουσα τιμή αφού σε διαφορετική περίπτωση το σφάλμα που θα προκύψει στην τιμή της μέτρησης θα είναι μη αποδεκτό. Ακόμη, υπολογίζεται και η χρονική συνιστώσα της οποίας η τιμή επηρεάζεται από το πόσο κοντά στο επόμενο Heart-Beat μήνυμα είναι το STEP στο οποίο βρισκόμαστε. Αν βρισκόμαστε κοντά, χρονικά, σε κάποιο Heart-Beat, η χρονική συνιστώσα θα μας "συμβουλέψει" να μην στείλουμε κάποιο μήνυμα, αλλά να αφήσουμε τον κόμβο που θα λάμβανε το κανονικό μήνυμα να προσεγγίσει την τιμή με την αριθμητική μέθοδο. Αυτό γίνεται, αφού σε σύντομο STEP θα λάβει ροή κατευθείαν από τους κόμβους που πραγματοποιούν τις μετρήσεις. Τέλος, αφού υπολογιστεί καθεμία από τις συνιστώσες της συνάρτησης αξιολόγησης υπολογίζεται η συνολική τιμή, λαμβάνοντας υπόψη και τα βάρη, και η συνάρτηση επιστρέφει το αποτέλεσμα της σύγκρισης της τιμής αυτής με το UT_THRESHOLD (κατώφλι συνάρτησης αξιολόγησης). Τιμή της συνάρτησης αξιολόγησης μεγαλύτερη από αυτή του κατωφλίου σημαίνει πως πρέπει να πραγματοποιηθεί αποστολή. Η *eval_utility* καλείται στα κατάλληλα σημεία του κώδικα, στις περιπτώσεις που χρειάζεται να λάβουμε απόφαση για το αν θα στείλουμε μια τιμή ή όχι.

δ) *event result_t Timer.fired()*: Η συνάρτηση αποτελεί το σημαντικότερο τμήμα του προγράμματος και εκτελείται κάθε φορά που ο χρονομέτρης "πυροδοτεί" το event του. Ο κώδικας εκτελείται σε κάθε CLICK. Ανάλογα με τον τύπο κόμβου, εκτελείται διαφορετικό τμήμα της συνάρτησης αφού κάθε κόμβος επιτελεί διαφορετικά καθήκοντα. Για τους αισθητήριους κόμβους, ο κώδικας εκτελείται όταν βρισκόμαστε στο κατάλληλο παράθυρο (SENSE_SEND) του STEP για να πραγματοποιήσουμε μέτρηση και εφ'όσον έχει ολοκληρωθεί η προηγούμενη μέτρηση. Οι πρώτες μετρήσεις που λαμβάνει ο αισθητήρας αντιμετωπίζονται ως Heart-Beat μηνύματα, ώστε να έχουμε τις αναγκαίες τιμές για να κάνουμε παρεμβολή και προωθούνται προς τους επόμενους κόμβους της ιεραρχίας. Αν το μήνυμα δεν είναι Heart-Beat τότε ενεργοποιείται η συνάρτηση αξιολόγησης και, ανάλογα με την εκτίμησή της, προωθούμε ή όχι τη μέτρηση. Κάθε μήνυμα που στέλνεται αρχικά περιέχει έναν μετρητή ο οποίος έχει διπλό ρόλο. Η απόλυτη τιμή του είναι το πλήθος των μετρήσεων που περιέχει το μήνυμα και αν η τιμή του είναι αρνητική σημαίνει πως το μήνυμα είναι Heart-Beat. Αυτή η σύμβαση χρησιμοποιείται ώστε ο κόμβος που θα λάβει το μήνυμα να γνωρίζει ότι πρόκειται για Heart-Beat. Εκτός από τη μέτρηση, αποστέλλεται και η ταυτότητα του αισθητήρα που πραγματοποίησε τη μέτρηση, για στατιστικούς λόγους. Επιπλέον, με τη βοήθεια της συνάρτησης *shift_insert_sense* ανανεώνεται και ο πίνακας με τις τιμές που χρησιμοποιούνται στην παρεμβολή. Το τμήμα του κώδικα που αφορά τους ενδιάμεσους κόμβους ενεργοποιείται εφόσον έχουμε λάβει τουλάχιστον ένα μήνυμα. Αν βρισκόμαστε στο κατάλληλο tick του STEP (COMM_REC_DONE) τότε αρχίζει η διαδικασία επεξεργασίας των μηνυμάτων. Όσο διαρκεί η επεξεργασία αυτών των μηνυμάτων, απορρίπτονται τα μηνύματα που φτάνουν στον κόμβο. Σε πρώτο στάδιο, ελέγχουμε αν έχουμε λάβει τιμές για όλες τις μετρήσεις που θα έπρεπε. Αν όχι, οι τιμές αυτές αναπαράγονται με τη γραμμική παρεμβολή. Στη συνέχεια, για κάθε μια μέτρηση εφαρμόζουμε τη συνάρτηση αξιολόγησης και έτσι αποφασίζουμε αν η τιμή αυτή θα προωθηθεί ή όχι. Σε περίπτωση Heart-Beat μηνύματος όλες οι τιμές προωθούνται. Με τη βοήθεια της συνάρτησης *shift_insert* ενημερώνεται και πάλι ο πίνακας με τιμές για την προσομοίωση. Ο κώδικας για τους ενδιάμεσους κόμβους εκτελείται και στην περίπτωση όπου έχουμε να προωθήσουμε μήνυμα και το μήνυμα έχει ετοιμαστεί. Η αποστολή αυτή συμβαίνει στο χρονικό παράθυρο του STEP που καθορίζεται από τη σταθερά COMM_SEND. Αντίστοιχη λογική με αυτή των ενδιάμεσων κόμβων χρησιμοποιούμε και στην περίπτωση του κόμβου-δεξαμενή. Πιο συγκεκριμένα, στο χρονικό παράθυρο που καθορίζεται από τη σταθερά SINK_REC_DONE θεωρούμε ότι η δεξαμενή έχει δεχθεί όσα μηνύματα έπρεπε να δεχθεί και ξεκινά την επεξεργασία τους. Και πάλι, αν λάβουμε μηνύματα κατα τη διάρκεια της επεξεργασίας, τα απορρίπτουμε.

Το πρώτο βήμα είναι η αναπαραγωγή τυχόν τιμών που δεν εστάλησαν από κατώτερους στην ιεραρχία κόμβους με τη βοήθεια της *extrap_val*. Στη συνέχεια, τα αποτελέσματα που προκύπτουν αποθηκεύονται σε ένα αρχείο και είναι έτοιμα να προωθηθούν για περαιτέρω επεξεργασία. Στα παραπάνω πρέπει να προσθέσουμε ότι σε κάθε βήμα, για όλους τους τύπους κόμβου, υπολογίζεται το ενεργειακό κόστος, τόσο της λειτουργίας του επεξεργαστή, όσο και το ενεργειακό κόστος για αποστολές και λήψεις μηνυμάτων. Ακόμη, φροντίζουμε τα λογικά flags να έχουν τις κατάλληλες τιμές, ώστε σε συνδυασμό με τα χρονικά παράθυρα που έχουν οριστεί με τις σταθερές να διατηρείται ο συγχρονισμός, ο οποίος είναι ζωτικής σημασίας για την εφαρμογή. Επίσης, συλλέγονται τα κατάλληλα στατιστικά στοιχεία που θα αναλύσουμε στη συνέχεια και εμφανίζουν τη συμπεριφορά της αρχιτεκτονικής.

ε) *event TOS_MsgPtr MyReceive.receive(TOS_MsgPtr m)*: Η συνάρτηση αυτή υλοποιεί το event που πυροδοτείται όταν έχουμε λήψη μηνύματος. Το event αυτό το χρησιμοποιούν μόνο οι ενδιάμεσοι κόμβοι και ο κόμβος-δεξαμενή. Όπως αναφέραμε και παραπάνω, οι λήψεις μηνυμάτων γίνονται σε συγκριμένα χρονικά διαστήματα όταν ο επεξεργαστής βρίσκεται σε κατάσταση idle (listening) και τυχόν μηνύματα που ληφθούν για οποιοδήποτε λόγο εκτός του προκαθορισμένου παραθύρου απορρίπτονται. Σε κάθε περίπτωση ελέγχεται αν το μήνυμα που λάβαμε είναι Heart-Beat και πραγματοποιείται μια αρχική επεξεργασία, η οποία οργανώνει την πληροφορία. Εν συνεχεία υπολογίζεται και το κόστος της λήψης του μηνύματος.

5.2.3 Υπόλοιπες συναρτήσεις

Εκτός από τις βασικές συναρτήσεις που αναφέραμε παραπάνω κρίνεται σκόπιμο να αναφερθούμε σύντομα και στις υπόλοιπες. Αυτές είναι οι εξής:

- *void initialize_tree()*: Αρχικοποιεί μια δυαδική δενδρική δομή η οποία χρησιμοποιείται για τη δρομολόγηση των ροών. Ακόμα, παρέχει στους κόμβους κάποια αρχική πληροφορία σχετικά με τον πατέρα τους (στον οποίο στέλνουν μηνύματα) αλλά και από πόσα φύλλα δέχονται ροές.
- *int memorycpy(int8_t* dest, int8_t* src, int16_t size)*: Η συνάρτηση αυτή απλά υλοποιεί αντιγραφή size θέσεων μνήμης από το src στο dest.
- *void shift_insert_sense(float newval)*: Υλοποιεί το “φρεσκάρισμα” των τιμών που αποθηκεύονται για παρεμβολή στους αισθητήριους κόμβους. Η newval είναι η νέα τιμή που θα εισαχθεί.
- *void shift_insert(uint8_t pos, float newval)*: Αντίστοιχα με την παραπάνω, μόνο που χρησιμοποιείται για φρεσκάρισμα στους ενδιάμεσους κόμβους και στον κόμβο δεξαμενή. Η διαφορά με την παραπάνω έγκειται στο ότι στα φύλλα κρατάμε τιμές μόνο για μια παρεμβολή ενώ πιο πάνω στην ιεραρχία για τις παρμεβολές που αντιστοιχούν σε περισσότερα φύλλα οπότε πρέπει να προσδιορίζουμε πού αναφέρεται το “φρεσκάρισμα” με χρήση του *pos*.
- *command result_t StdControl.init()*: Εκκινεί την εφαρμογή, αρχικοποιεί μεταβλητές, δεσμεύει δυναμικά μνήμη.
- *command result_t StdControl.start()*: Χρησιμοποιείται για την εκκίνηση του χρονομέτρη
- *command result_t StdControl.stop()*: Αποτελεί την υλοποίηση του stop για το interface StdControl.
- *event result_t MySend.sendDone(TOS_MsgPtr sent,result_t success)*: Πυροδοτείται όταν πραγματοποιηθεί η αποστολή ενός μηνύματος.

- *event result_t MemAlloc.allocComplete(HandlePtr handle, result_t success)*: Πυροδοτείται όταν πραγματοποιηθεί η δυναμική δέσμευση μνήμης και σε αυτή γίνονται αρχικοποιήσεις στη μνήμη που μόλις δεσμεύτηκε. Η δεσμευμένη μνήμη επιστρέφεται μέσω του *handle*.
- *event result_t MemAlloc.compactComplete()*: Δε χρησιμοποιείται, η παρουσία της είναι αναγκαία για τη χρήση του interface MemAlloc.
- *event result_t MemAlloc.reallocComplete(Handle handle, result_t success)*: Ομοίως με την προηγούμενη.

Εκτός από τα σχόλια που αναφέρονται παραπάνω, λεπτομέρειες που αφορούν τον κώδικα, την επιλογή των σταθερών και των flags, υπάρχουν στο εσωτερικό του κώδικα (βλέπε Παράρτημα Δ) σε μορφή σχολίων.

ΚΕΦΑΛΑΙΟ 6

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΕΩΝ

6.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιαστούν τα αποτελέσματα των πιο σημαντικών προσομοιώσεων που έγιναν. Το πολυώνυμο παρεμβολής που επιλέχθηκε ήταν γραμμικό. Επιπλέον τα Heart-Beat μηνύματα αποστέλλονταν κάθε 9 βήματα. Αφού το πολυώνυμο παρεμβολής ήταν γραμμικό, 2 HB μηνύματα αποστέλλονταν διαδοχικά. Οι βασικές διαφορές των δύο προσομοιώσεων που παρουσιάζονται παρακάτω είναι η εκτίμηση για το ποσοστό του χρόνου που παραμένει κάθε κόμβος σε κατάσταση αδράνειας (idle) και σε κατάσταση εξοικονόμησης ενέργειας (power-save). Στο πρώτο παράδειγμα θεωρείται ότι κάθε κόμβος μένει για ένα χρονικό διάστημα σε αδράνεια (χρόνος κατά τον οποίο ο κόμβος περιμένει για μήνυμα, ή απλά η διαχείριση χρόνου δεν είναι όσο καλή θα μπορούσε να γίνει), ενώ στο δεύτερο παράδειγμα ο κόμβος, χρησιμοποιώντας όλες τις δυνατές πληροφορίες, δεν παραμένει σε κατάσταση αδράνειας καθόλου και από ενεργός μεταβαίνει άμεσα σε κατάσταση εξοικόνομησης ενέργειας και αντίστροφα.

6.2 Παράδειγμα που περιλαμβάνει χρόνο σε κατάσταση αδράνειας

Σε αυτήν την ενότητα παρουσιάζονται τα αποτελέσματα της προσομοίωσης με την υπόθεση ότι ο εκάστοτε κόμβος μένει στις διάφορες καταστάσεις σύμφωνα με τον Πίνακα 6.1.

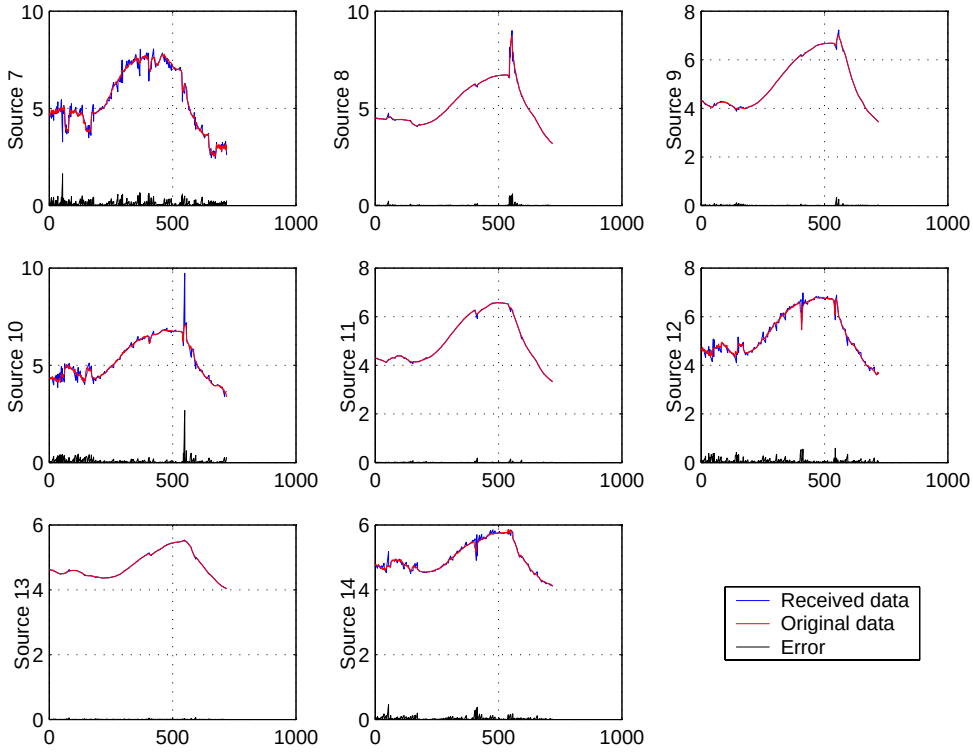
Πίνακας 6.1: Καταστάσεις επεξεργαστή και χρονική συμμετοχή στον κύκλο λειτουργίας

CPU	Energy Cost	Time contribution per Step
Idle	9600000 nJ/sec	0,48/16 (3% per step) for sense-nodes 2/16 (12.5% per step) for others
Power-save	330000 nJ/sec	14,72/16 (92% per step) for sense-nodes 13,2/16 (82.5% per step) for others
Active	4 nJ/Instruction	
Change mode	62 nJ/per change	

Αν αθροιστούν τα ποσοστά του χρόνου που, σε κάθε βήμα, ο κόμβος δουλεύει σε idle και σε power-save κατάσταση το άθροισμα βγαίνει 95%. Θεωρούμε ότι το υπόλοιπο του χρόνου είναι ο χρόνος που χρειάζεται για όλες τις άλλες λειτουργίες που γίνονται. Δηλαδή για τους υπολογισμούς, τις αποστολές και τις λήψεις πακέτων και τις αλλαγές καταστάσεων.

Η προσομοίωση αυτή αναφέρεται σε ένα δίκτυο δενδρικής δομής όπως αυτό του Σχήματος 4.1. Υποθέτουμε ένα επίπεδο 8 αισθητήριων κόμβων που στέλνουν

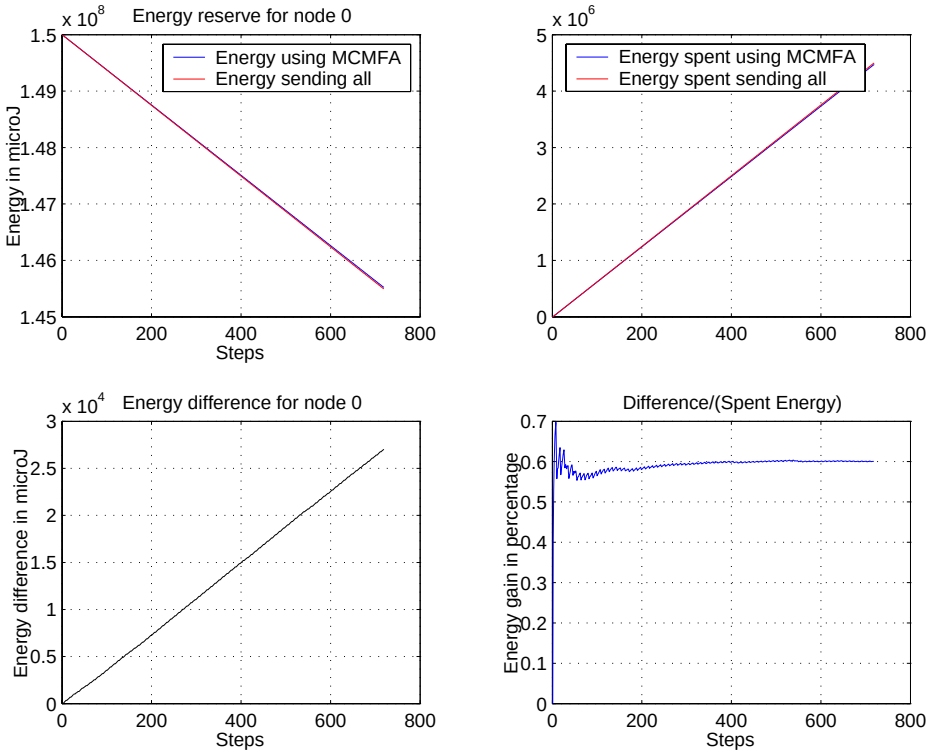
μετρήσεις προς τον κόμβο δεξαμενή και καθώς τα δεδομένα διαδίδονται πραγματοποιείται και συνάθροιση δεδομένων (data aggregation) με αποτέλεσμα σε έναν κύκλο λειτουργίας μέχρι ένα πακέτο να αποστέλλεται από κάθε κόμβο ενώ μπορεί να λάβει μέχρι δύο πακέτα, ένα από κάθε κόμβο-απόγονο της δενδρικής δομής. Το πρώτο σχήμα (Σχήμα 6.1) απεικονίζει τις ροές δεδομένων που έφτασαν τελικά στον προορισμό, σε σύγκριση με τα αρχικά δεδομένα. Τυπώνεται επίσης και το σχετικό σφάλμα το οποίο δεν ξεπερνά το κατώφλι που έχουμε θέσει στην υλοποίηση (εν προκειμένω 10%). Οι τιμές που αποστέλλονται έχουν ληφθεί από έναν μετεωρολογικό σταθμό[3] και αντιστοιχούν σε πραγματικές τιμές θερμοκρασίας εξωτερικού χώρου στη Χαβάη. Η συμπεριφορά των καμπυλών δεν είναι όμοια για τις διάφορες πηγές και έτσι μπορουμε να παρατηρήσουμε μια μη κοινή συμπεριφορά της αρχιτεκτονικής. Αυτό οφείλεται στο ότι η συμπεριφορά της συνάρτησης αξιολόγησης εξαρτάται από το σφάλμα που παράγεται από τις προσεγγίσεις και ο κάθε κόμβος πραγματοποιεί αρκετά διαφορετικές προσεγγίσεις.



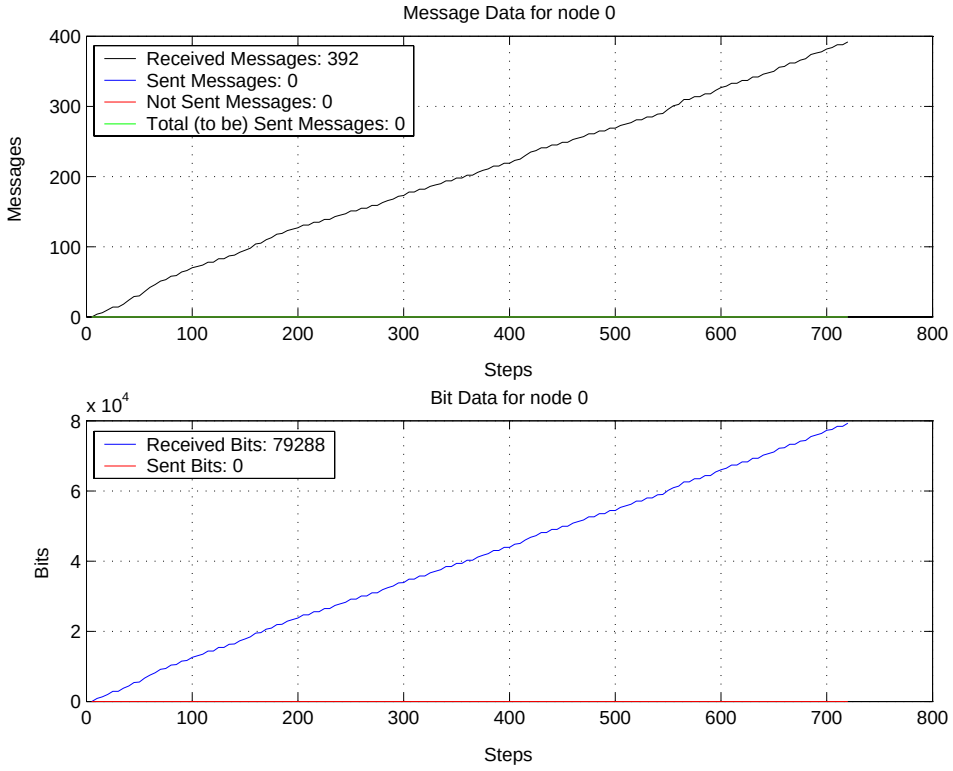
Σχήμα 6.1: Ροές δεδομένων των 8 πηγών, initial and received data

Στα Σχήματα 6.2 και 6.3 φαίνονται διάφορα δεδομένα γύρω από τον κόμβο 0, δηλαδή τον κόμβο-δεξαμενή της συγκεκριμένης τοπολογίας. Συγκεκριμένα, στο Σχήμα 6.2 φαίνεται το ενεργειακό απόθεμα, η καταναλωθείσα ενέργεια και οι αντίστοιχες τιμές σε σχέση με την περίπτωση αποστολής όλων των μηνυμάτων, δηλαδή της μη χρήσης της συνάρτησης αξιολόγησης και της προώθησης των μηνυμάτων, χωρίς να χρειάζεται να εκπληρώνονται κάποια κριτήρια. Στο ίδιο σχήμα, φαίνονται δύο ακόμα μετρικές του ενεργειακού κέρδους. Η πρώτη, είναι η διαφορά μεταξύ του ενεργειακού αποθέματος σε περίπτωση αποστολής όλων των μηνυμάτων αδιακρίτως και του αποθέματος σε περίπτωση χρήσης της προτεινόμενης αρχιτεκτονικής, και η δεύτερη, είναι ο λόγος της προηγούμενης διαφοράς προς την ενέργεια που έχει καταναλωθεί μέχρι την τρέχουσα χρονική στιγμή, δηλαδή το ποσοστό του ενεργειακού κέρδους ανά κύκλο λειτουργίας.

[3] http://iqup.ifa.hawaii.edu/domeenv/



Σχήμα 6.2: Ενεργειακή κατάσταση κόμβου 0 (κόμβος βάση)

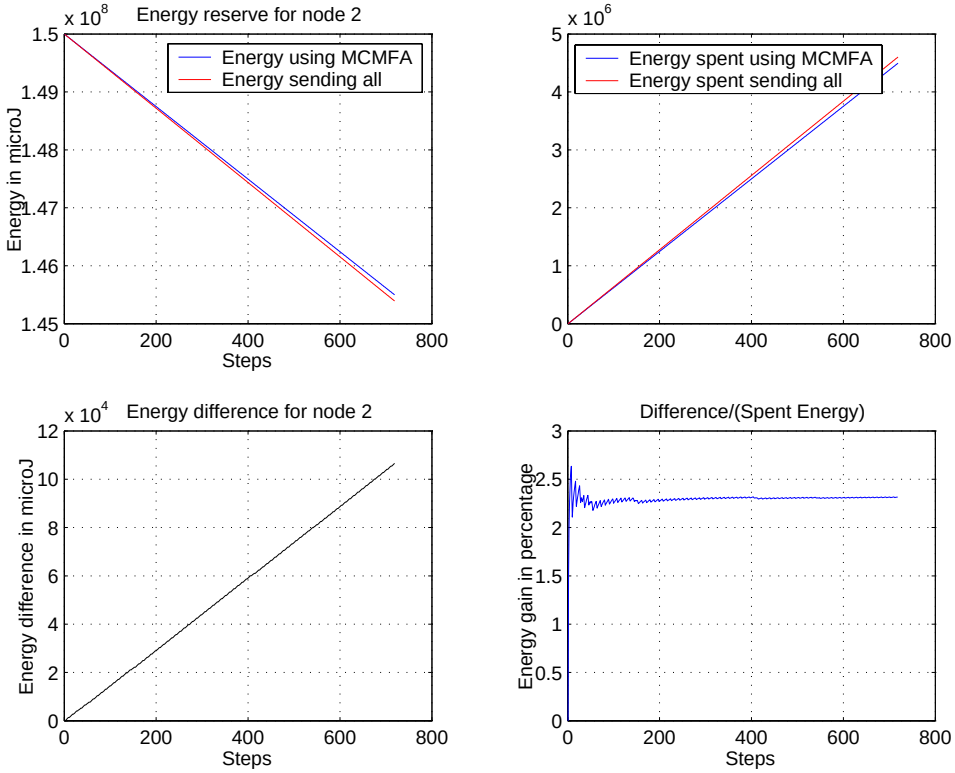


Σχήμα 6.3: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 0

Έτσι, εάν υποθέσουμε ότι ο κόμβος-δεξαμενή έχει τις ίδιες ενεργειακές προδιαγραφές με τους υπόλοιπους κόμβους του δικτύου, τότε, έχει ένα ποσοστιαίο κέρδος της τάξης του 0.6%, κάτι που είναι λογικό αφού το μόνο που καταφέρνει να «γλιτώσει» αυτός ο κόμβος είναι μερικές λήψεις μηνυμάτων από το πιο κάτω επίπεδο. Στο Σχήμα 6.3 βλέπουμε κάποια άλλα στατιστικά γύρω από τον κόμβο δεξαμενή που δεν έχουν, όμως,

μεγάλο ενδιαφέρον αφού ο κόμβος αυτός απλά δέχεται μηνύματα, αποθηκεύει τις τιμές και προσεγγίζει τις τιμές που δεν έλαβε ανά κύκλο λειτουργίας. Τα στατιστικά που παρουσιάζονται στο Σχήμα 6.3 είναι οι αποστολές και οι λήψεις μηνυμάτων και bit, και παρουσιάζουν μεγαλύτερο ενδιαφέρον στους κόμβους επικοινωνίας που θα δούμε παρακάτω.

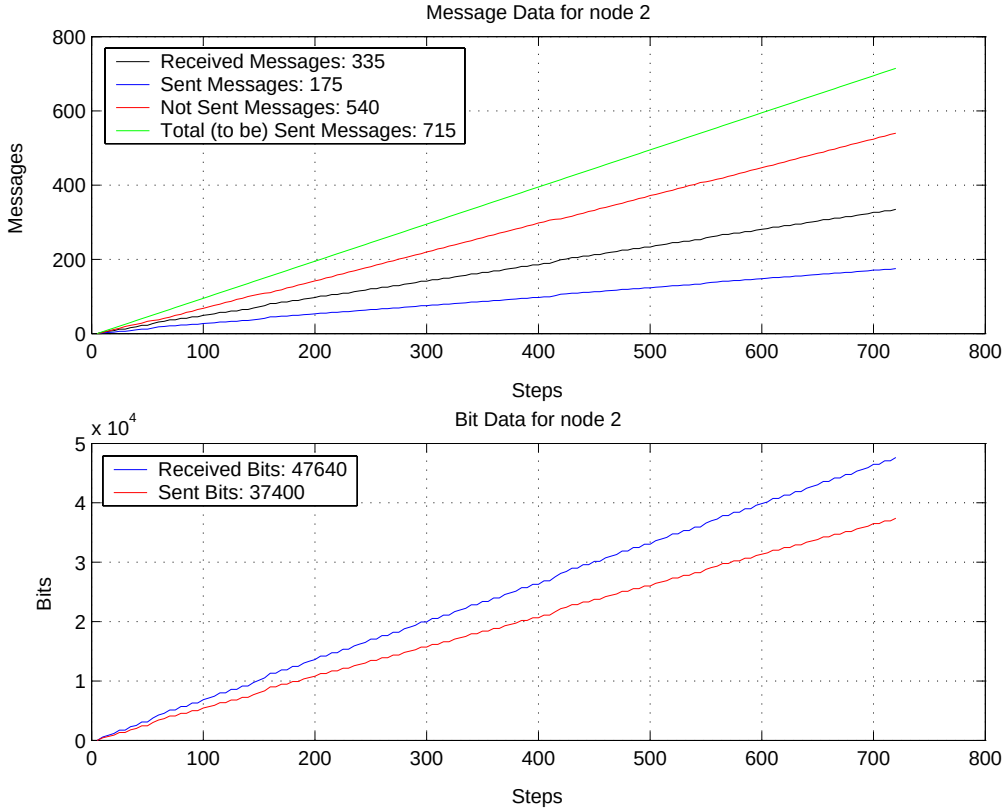
Στις επόμενες σελίδες παρατίθενται τα αντίστοιχα σχήματα για κάποιους ακόμα κόμβους της τοπολογίας. Η τοπολογία του δικτύου, που όπως προαναφέρεται είναι αυτή του Σχήματος 4.1, είναι ιεραρχική και κάθε κόμβος έχει έναν διακριτό ρόλο μέσα στο δίκτυο. Έτσι, ο κόμβος 2, είναι ένας κόμβος επικοινωνίας που λαμβάνει μηνύματα από άλλους κόμβους επικοινωνίας (από τον κόμβο 5 και τον κόμβο 6) και στέλνει τα συναθροισμένα δεδομένα στο κόμβο-δεξαμενή. Με τη χρήση της προτεινόμενης αρχιτεκτονικής, όπως φαίνεται από το Σχήμα 6.4, ο κόμβος έχει ένα ποσοστιαίο ενεργειακό κέρδος της τάξης του 2% - 2,5%. Αυτό προέκυψε, όπως φαίνεται από το Σχήμα 6.5, γιατί ο κόμβος αυτός κατάφερε να αποφύγει 540 αποστολές μηνυμάτων από τις 715 που θα μπορούσε να είχε κάνει σε περίπτωση που έστελνε όλα τα μηνύματα, και, όπως μεταφράζεται αυτό σε bit ενώ έλαβε 47640 bits έστειλε 37400. Τα bit βέβαια που έστειλε δεν αντιστοιχούν πλήρως με τις τιμές που διαδίδονται στο δίκτυο αφού όσες τιμές δεν έλαβε τις αντικατέστησε με την προσέγγιση που έγινε και επαναξιολόγησε την αποστολή τους.



Σχήμα 6.4: Ενεργειακή κατάσταση κόμβου 2

Μετά από 720, περίπου, πλήρεις κύκλους λειτουργίας ο κόμβος 2 έχει ένα συνολικό ενεργειακό κέρδος της τάξης των 100 mJ, ενέργεια αρκετή για αποστολή 135 Kbit. Το κέρδος αυτό όμως χάνει μεγάλο μέρος από την αξία του αν δε γίνει σωστή διαχείριση του χρόνου παραμονής του κόμβου σε κάθε κατάσταση. Αυτό οφείλεται στο ότι εάν ο κόμβος είναι σε κατάσταση αδράνειας καταναλώνει 9.6 mW δηλαδή θα χρειαστεί μόλις 10.4 δευτερόλεπτα για να καταναλώσει 100 mJ, ενώ εάν είναι σε κατάσταση εξοικονόμησης ενέργειας ο χρόνος αυτός θα αυξηθεί κατά 30 φορές. Παρ'όλο που το

ενεργειακό κέρδος είναι μεγάλο σε σχέση με την αποστολή μηνυμάτων και την εκτέλεση εντολών δεν είναι τόσο μεγάλο σε σχέση με την πάγια κατανάλωση ενέργειας. Αυτό είναι ένα πολύ χρήσιμο συμπέρασμα που μας οδηγεί στο να συνδυάζουμε την επιλεκτική προώθηση δεδομένων με την προσεκτική κατανομή χρόνου της εκάστοτε λειτουργίας του κόμβου.

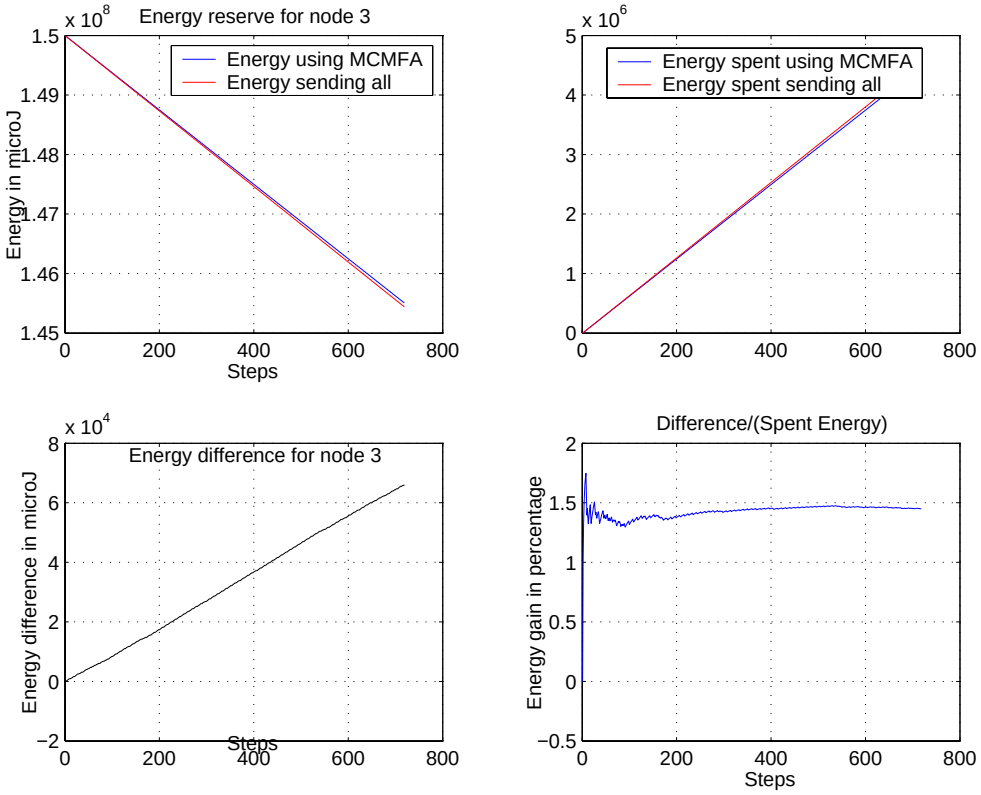


Σχήμα 6.5: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 2

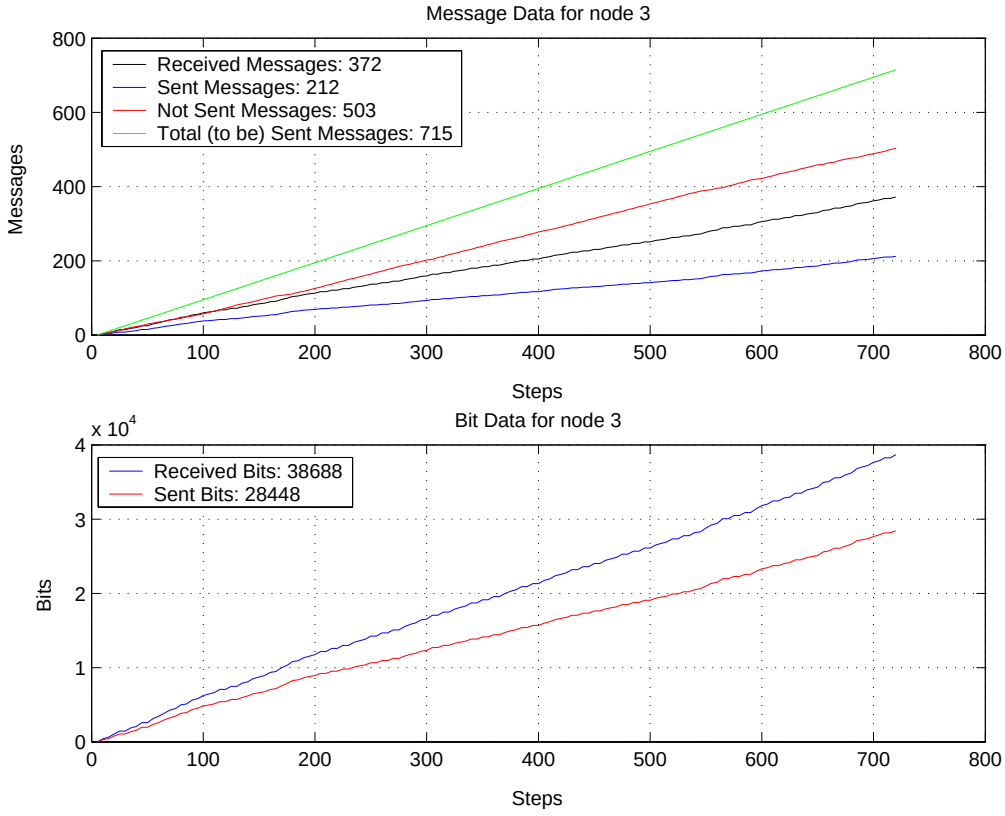
Ο κόμβος 3, ένας κόμβος επικοινωνίας, είναι ο επόμενος του οποίου παρουσιάζουμε τα στατιστικά. Ο κόμβος αυτός λαμβάνει μετρήσεις από τους αισθητήριους κόμβους, δηλαδή, βρίσκεται στο τελευταίο επίπεδο επικοινωνίας. Το ενεργειακό κέρδος του είναι λίγο μικρότερο από τον κόμβο 2 και συγκεκριμένα είναι της τάξης του 1.5%. Αυτό το ποσοστιαίο κέρδος είναι αποτέλεσμα της μη αποστολής 503 μηνυμάτων έναντι της αποστολής μόνο 212. Μεταφρασμένο σε bits, ενώ ο κόμβος έλαβε 38688 bits προώθησε 28448. Η διαφορά αυτή οφείλεται όχι μόνο στη μη αποστολή κάποιων μηνυμάτων λόγω της αξιολόγησης των μεταδόσεων αλλά και στη συνάθροιση δεδομένων που πραγματοποιείται σε κάθε επίπεδο επικοινωνίας. Η συνάθροιση δεδομένων οδηγεί σε ενεργειακό κέρδος γιατί για κάθε μήνυμα που διαβιβάζεται υπάρχει ένα επιπλέον κόστος (overhead), το οποίο αποφεύγεται όταν τα μηνύματα συναθροίζονται. Το καθαρό ενεργειακό κέρδος είναι γύρω στα 70 mJ μετά από 720 αποστολές.

Τα στατιστικά του κόμβου 5 παρουσιάζονται στα Σχήματα 6.8 και 6.9, παρατηρώντας μια παραπλήσια συμπεριφορά με τον κόμβο 3, ειδικά ως προς το ποσοστιαίο κέρδος. Γενικά, όμως, η εικόνα των δύο κόμβων επικοινωνίας του δευτέρου επιπέδου είναι συγγενική, κάτι που δείχνει ότι η αρχιτεκτονική οδηγεί σε μια παρόμοια συμπεριφορά για κόμβους που επιτελούν τον ίδιο ρόλο. Τα μηνύματα που έλαβε ο κόμβος 5 είναι 333 από ένα μέγιστο 715*2 = 1430 μηνύματα, οπότε το κέρδος δεν περιορίζεται στη μείωση

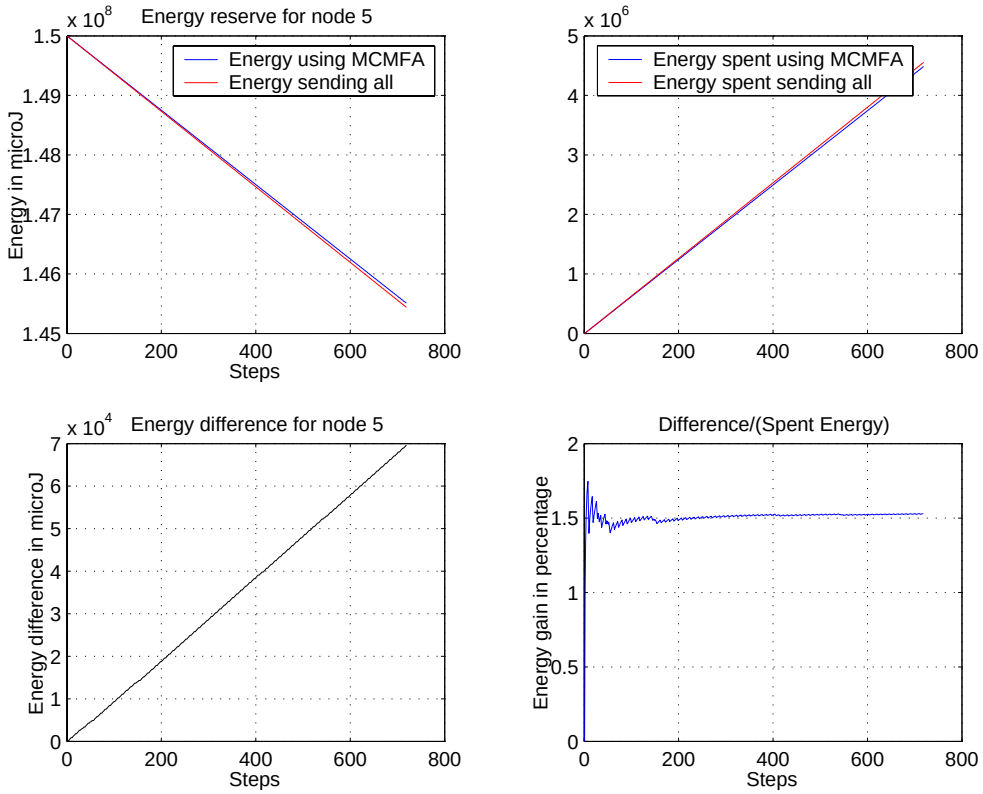
των αποστολών αλλά οφείλεται και στη μείωση των παραλαβών. Ο κόμβος 5 τελικά προώθησε 173 μηνύματα έναντι του μεγίστου των 715. Έλαβε 34632 bits και προώθησε 24392.



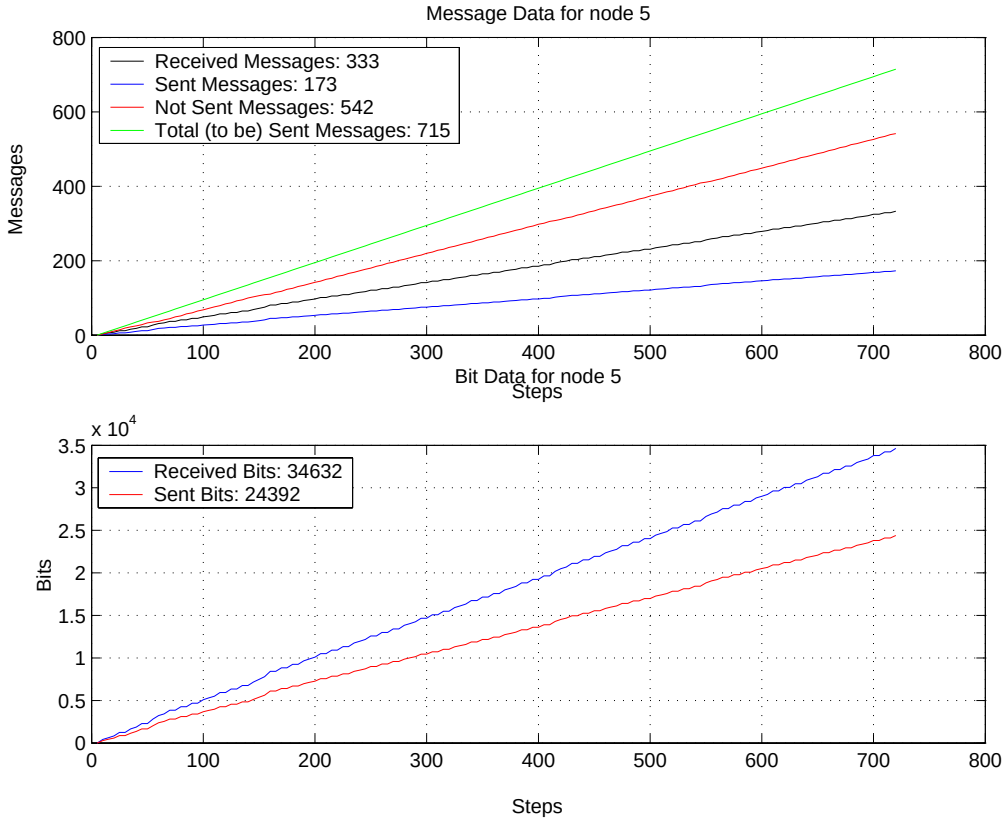
Σχήμα 6.6: Ενεργειακή κατάσταση κόμβου 3



Σχήμα 6.7: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 3

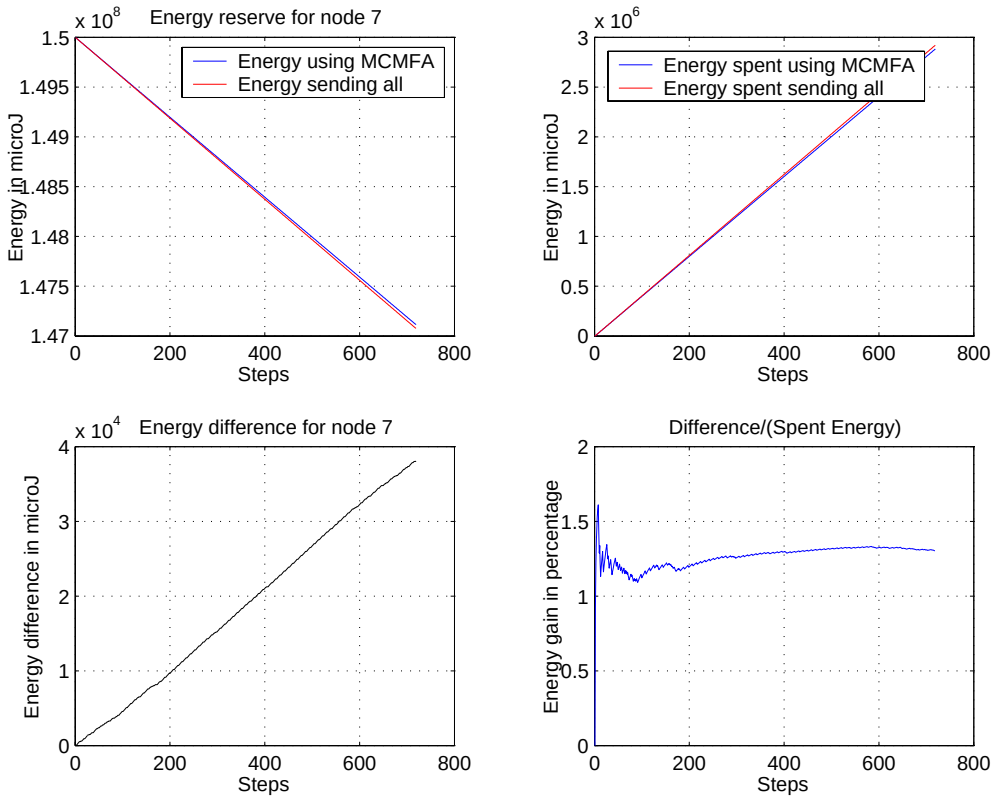


Σχήμα 6.8: Ενεργειακή κατάσταση κόμβου 5



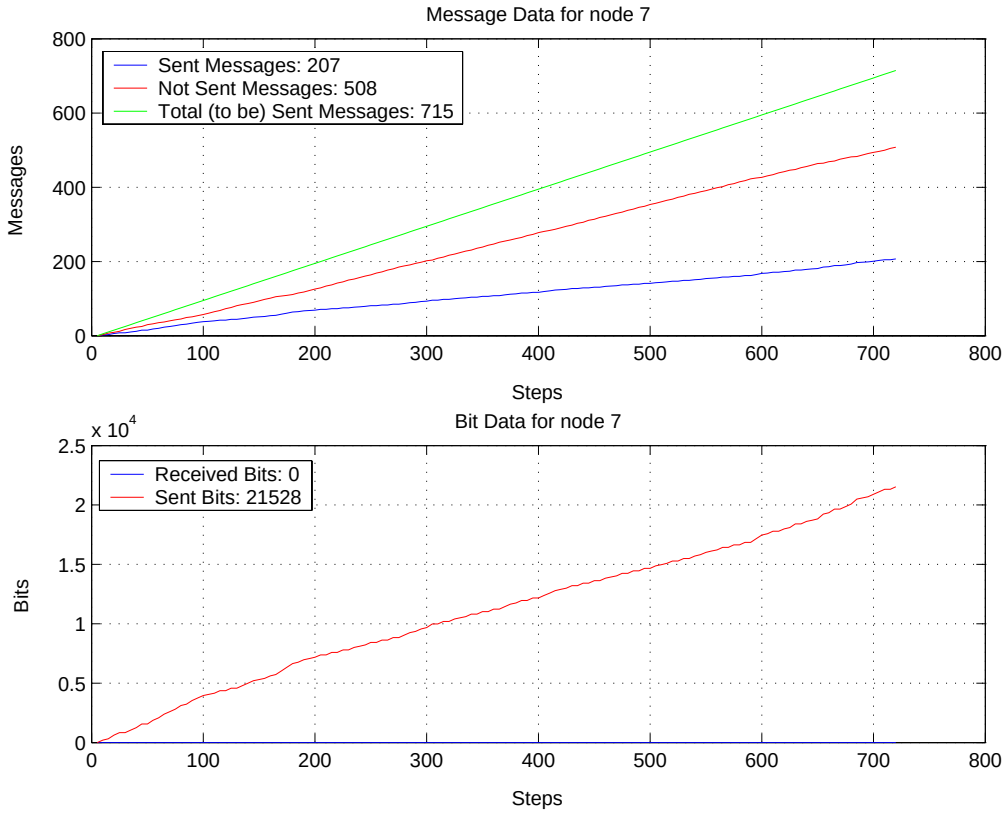
Σχήμα 6.9: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 5

Οι κόμβοι 7 και 12 είναι δύο από τους οκτώ κόμβους-πηγές (αισθητήριους κόμβους). Οι κόμβοι αυτοί (και όλοι οι υπόλοιποι αισθητήριοι κόμβοι) μέσα σε έναν πλήρη κύκλο λειτουργίας μετράνε μία τιμή, που στα παραδείγματά μας είναι τιμή εξωτερικής θερμοκρασίας, και εάν ικανοποιούνται τα κριτήρια της συνάρτησης αξιολόγησης την αποστέλλουν προς τον αντίστοιχο κόμβο επικοινωνίας. Πιο συγκεκριμένα, ο κόμβος 7 στέλνει στον κόμβο 3 και ο κόμβος 12 στέλνει στον κόμβο 5. Για τους δύο κόμβους παρατηρείται συγγενική συμπεριφορά αφού και οι δύο έχουν ποσοστιαίο ενεργειακό κέρδος ίσο με 1% έως 1.5%. Το καθαρό κέρδος είναι της τάξης των 4 mJ για τον κάθε κόμβο και οφείλεται στο ότι από το συνολικό μέγιστο των 715 αποστολών, ο κόμβος 7 έστειλε 207 μηνύματα, κάτι που μεταφράζεται σε 21528 bits και ο κόμβος 12 έστειλε 173 μηνύματα, συνολικού μεγέθους 17992 bits. Αν ένας αισθητήριος κόμβος έστελνε όλα τα μηνύματα, τότε θα αναγκαζόταν να στείλει 13 bytes για κάθε μήνυμα αφού κάθε αισθητήριος κόμβος αποστέλλει μια τιμή κινητής υποδιαστολής τεσσάρων bytes μαζί με δύο ακεραίους (με χρήση flag) του 1 byte έκαστος αλλά και 7 bytes λόγω του κόστους που προστίθεται κατά την εκάστοτε αποστολή (overhead). Έτσι για 715 αποστολές ένας αισθητήριος κόμβος θα έστελνε 13*8*715 = 74360 bits, ενώ με χρήση της προτεινόμενης αρχιτεκτονικής στέλνει μόνο το 24% - 29% αυτών των bits.

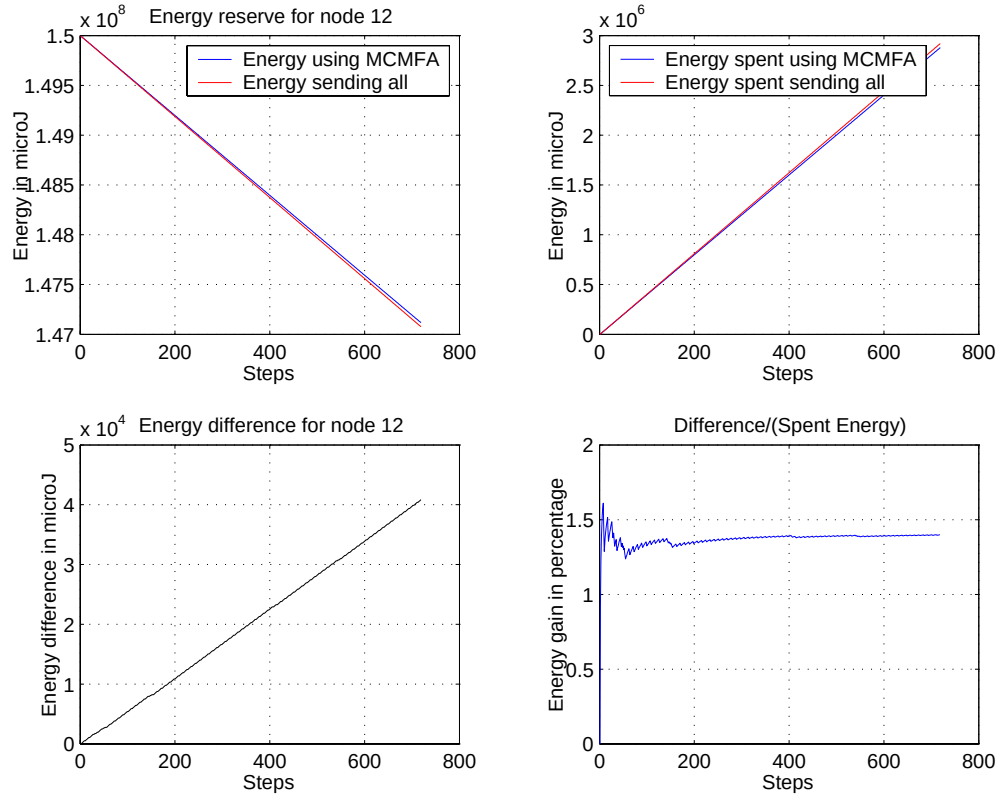


Σχήμα 6.10: Ενεργειακή κατάσταση κόμβου 7

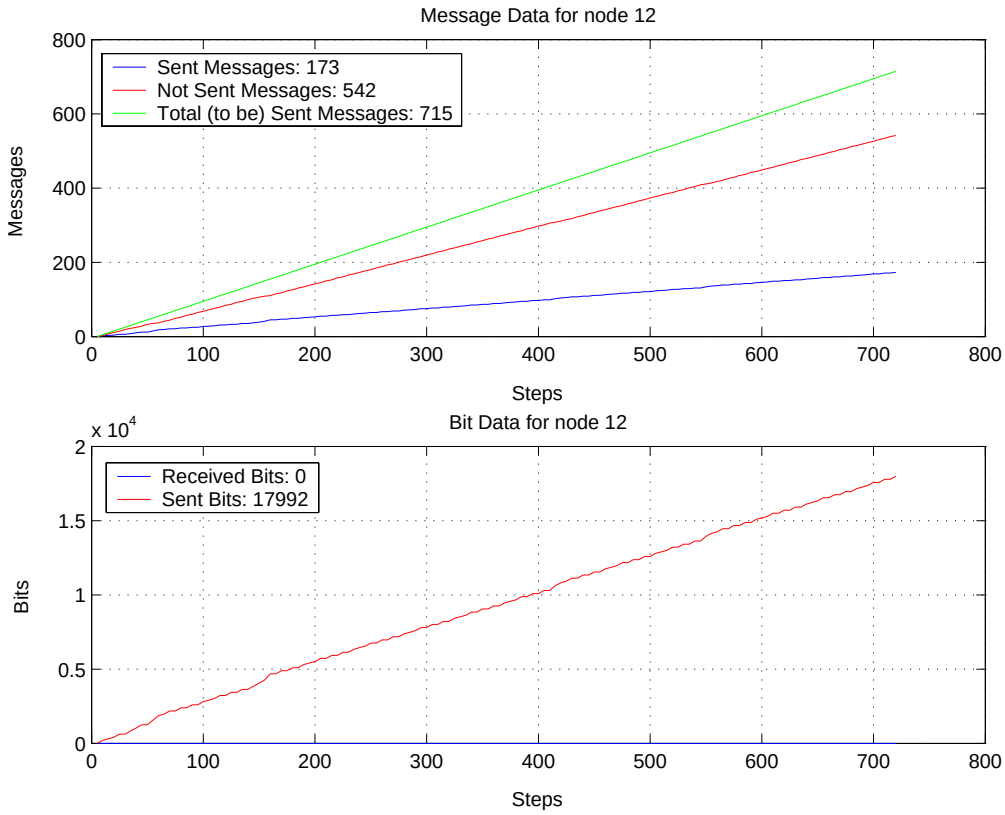
Τα Σχήματα 6.10 έως και 6.13 μας δείχνουν τη συμπεριφορά που περιγράφεται παραπάνω. Όλοι οι αισθητήριοι κόμβοι είχαν αντίστοιχη συμπεριφορά και γι'αυτό επιλέχθηκαν μόνο κάποιοι από αυτούς για να παρουσιαστούν τα στατιστικά τους και όχι όλοι. Οι κόμβοι δεν επιλέχθηκαν με κάποιο ιδαίτερο κριτήριο, με σκοπό να είναι αρκετά αντιπροσωπευτικοί ως προς όλο το δείγμα των αισθητήριων κόμβων.



Σχήμα 6.11: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 7



Σχήμα 6.12: Ενεργειακή κατάσταση κόμβου 12



Σχήμα 6.13: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 12

6.3 Παράδειγμα που περιλαμβάνει μόνο χρόνο σε κατάσταση λειτουργίας ή σε κατάσταση εξοικονόμησης ενέργειας

Σε αυτήν την ενότητα παρουσιάζονται τα αποτελέσματα της προσομοίωσης με την υπόθεση ότι ο εκάστοτε κόμβος μένει στις διάφορες καταστάσεις σύμφωνα με τον Πίνακα 6.2.

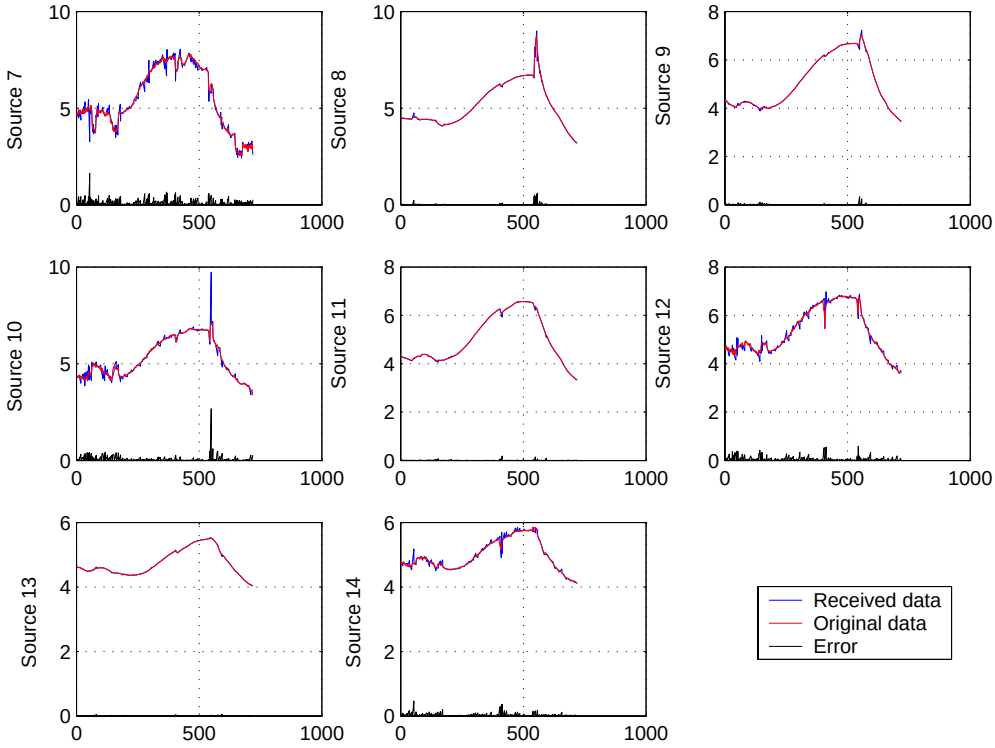
Πίνακας 6.2: Καταστάσεις επεξεργαστή και χρονική συμμετοχή στο κύκλο λειτουργίας

CPU	Cost	Time contribution per Step
Idle	9600000 nJ/sec	Every node avoids staying in idle mode
Power-save	330000 nJ/sec	15,2/16 (95% per step) for sense-nodes 15,2/16 (95% per step) for others
Active	4 nJ/Instruction	
Change mode	62 nJ/per change	

Σε αυτήν την προσομοίωση οι κόμβοι λειτουργούν μόνο σε power-save όσο δεν επιτελούν κάποια συγκεκριμένη λειτουργία. Ο χρόνος αυτός (power save time) είναι το 95% του συνολικού χρόνου ανά βήμα. Θεωρούμε ότι το υπόλοιπο του χρόνου είναι ο χρόνος που χρειάζεται για όλες τις άλλες λειτουργίες που γίνονται. Δηλαδή για τους υπολογισμούς, τις αποστολές και τις λήψεις πακέτων και τις αλλαγές καταστάσεων.

Στο Σχήμα 6.14 φαίνονται οι ροές δεδομένων αυτής της προσομοίωσης, οι οποίες είναι σχεδόν ίδιες με τις ροές δεδομένων του προηγούμενου πειράματος. Το σφάλμα δεν ξεπερνά το κατώφλι του 10% οι καμπύλες που προκύπτουν στον προορισμό από το συνδυασμό δεδομένων και προσεγγίσεων δεν έχουν πολύ μεγάλη ταλάντωση. Συνοπτικά, η εικόνα των δεδομένων είναι ικανοποιητική με βάση την εφαρμογή που υλοποιήθηκε, αλλά και τις τιμές των παραμέτρων που ορίζουν μια ανοχή στο σφάλμα της τάξης του 10%. Επίσης, το κατώφλι για τη συνάρτηση αξιολόγησης είναι ίσο με 0.85 προσπαθώντας έτσι να αποφύγουμε τις μεταδόσεις σχεδόν όλες τις φορές που το σφάλμα είναι κάτω από 10%.

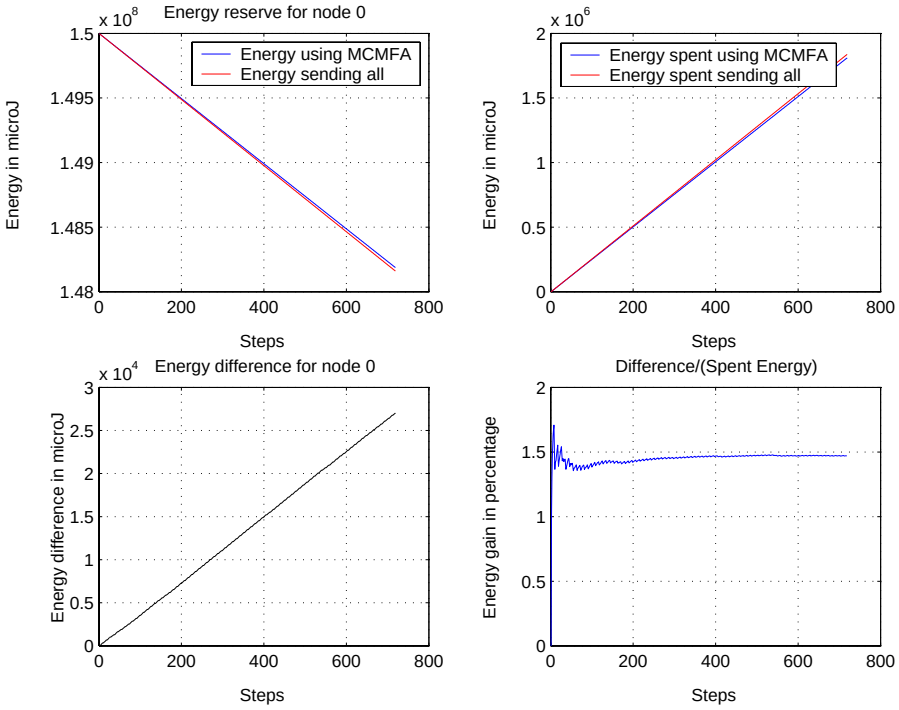
Η βασική διαφορά αυτής της προσομοίωσης από την προηγούμενη είναι ότι θεωρούμε ότι ο κόμβος δεν παραμένει καθόλου σε κατάσταση αδράνειας (idle state). Αντιθέτως, μόλις ολοκληρωθεί μια διεργασία μεταπίπτει, αμέσως, σε κατάσταση εξοικονόμησης ενέργειας και αντίστροφα, αμέσως πριν αρχίσει κάποια άλλη διεργασία ο επεξεργαστής αρχίζει να λειτουργεί πληρώνοντας μόνο το κόστος της μετάβασης από την κατάσταση εξοικονόμησης ενέργειας στην κατάσταση λειτουργίας. Είναι αναμενόμενο το ενεργειακό κέρδος να είναι σχετικά αυξημένο σε σχέση με το προηγούμενο πείραμα. Η συμπεριφορά όμως του δικτύου είναι η ίδια σε γενικές γραμμές.



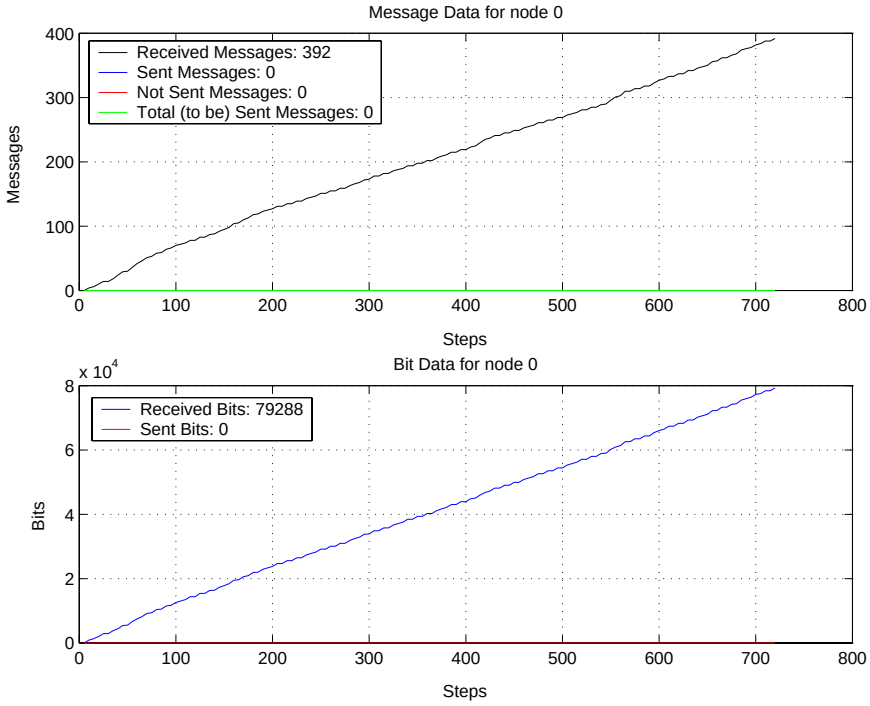
Σχήμα 6.14: Ροές δεδομένων των 8 πηγών, initial and received data

Αρχικά παρατηρούμε πως το ποσοστιαίο ενεργειακό κέρδος του κόμβου 0 έχει αυξηθεί από 0.6% σε 1.5%. Το καθαρό ενεργειακό κέρδος είναι ίσο με 25 mJ το οποίο αντιστοιχεί στην παραλαβή 222 Kbit. Στα Σχήματα 6.15 και 6.16 φαίνονται όλα τα στατιστικά για τον κόμβο 0, τον κόμβο-δεξαμενή. Η υπόθεση ότι αυτός ο κόμβος είναι

όμοιος με τους υπόλοιπους από άποψη επεξεργαστικών και ενεργειακών πόρων δεν είναι πάντα αληθής, πολλές φορές, όμως, συμβαίνει. Πιο συγκεκριμένα, πολλές φορές ο κόμβος-βάση είναι ένα πιο εξελιγμένο υπολογιστικό σύστημα το οποίο μπορεί να αποθηκεύσει και να επεξεργαστεί μεγάλες ποσότητες δεδομένων. Σε άλλες περιπτώσεις, το ρόλο της συλλογής των δεδομένων τον αναλαμβάνει ένας κοινός κόμβος, ο οποίος επικοινωνεί με κάποιο πιο εξελιγμένο σύστημα για να προωθήσει τα δεδομένα.

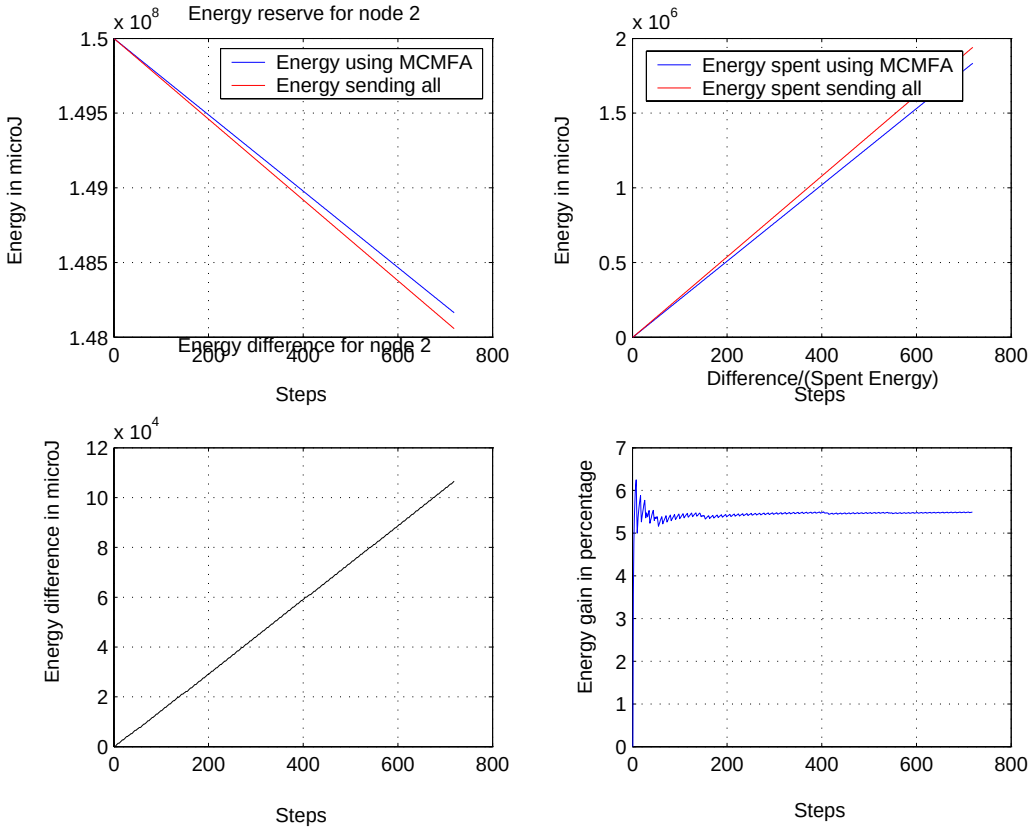


Σχήμα 6.15: Ενεργειακή κατάσταση κόμβου 0 (κόμβος βάση)



Σχήμα 6.16: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 0

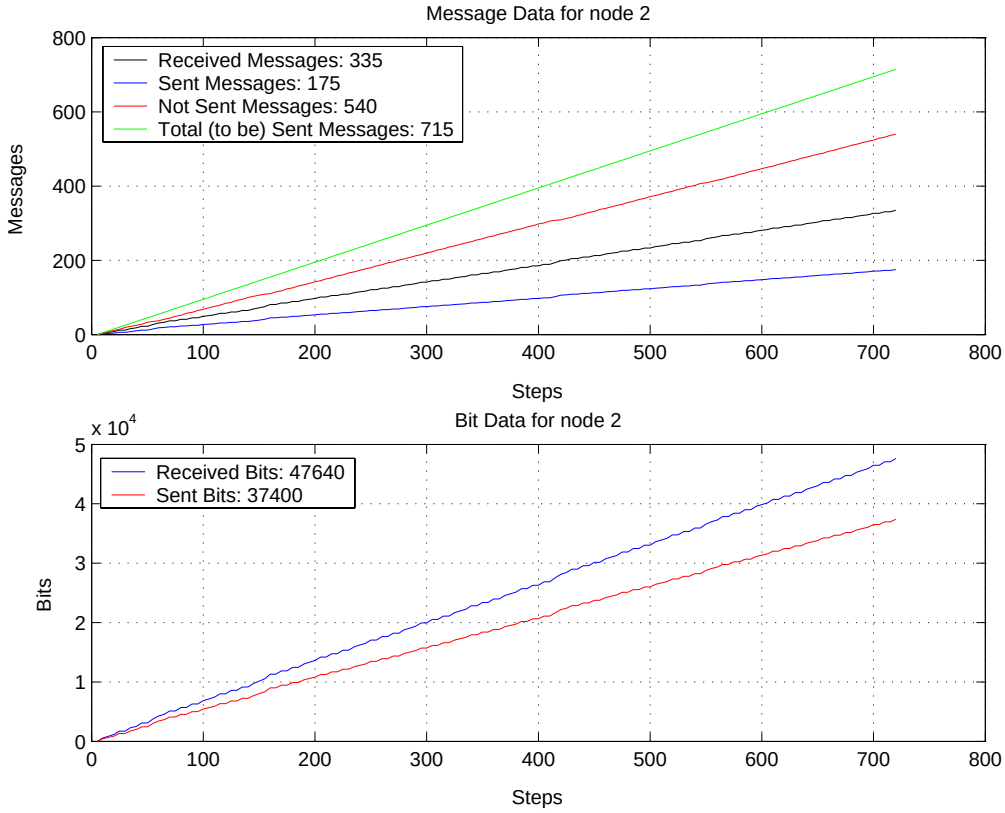
Η λειτουργία της προτεινόμενης αρχιτεκτονικής φαίνεται καλύτερα στα σχήματα που αφορούν τους κόμβους επικοινωνίας. Ο κόμβος 2, όπως φαίνεται στο Σχήμα 6.17, έχει πιο μεγάλο ποσοστιαίο ενεργειακό κέρδος από ότι στο προηγούμενο πείραμα, που φτάνει το 5.5%. Αυτό το ποσοστό μπορεί να μεταφραστεί σε ποσοστό επιμήκυνσης του χρόνου ζωής του κόμβου χάρη στη λειτουργία της συνάρτησης αξιολόγησης. Το καθαρό ενεργειακό κέρδος είναι συγκρίσιμο με πριν και ο λόγος είναι απλός. Η τιμή της ενέργειας που καταναλώνεται χωρίς τη χρήση της εφαρμογής (κόκκινη γραμμή στο Σχήμα 6.17) αναφέρεται και πάλι στη θεώρηση ότι ο κόμβος δε μένει καθόλου αδρανής, οπότε και η πάγια κατανάλωση – που είναι και η μεγαλύτερη ως προς το χρόνο – είναι εξίσου μικρότερη. Τελικά το καθαρό ενεργειακό κέρδος είναι ίσο με 120 mJ. Στο ποσοστιαίο κέρδος, όμως, αντικατοπτρίζεται η βελτίωση που οφείλεται στην αρχική υπόθεση. Το ποσοστιαίο κέρδος ανήλθε από 2% σε 5.5%.



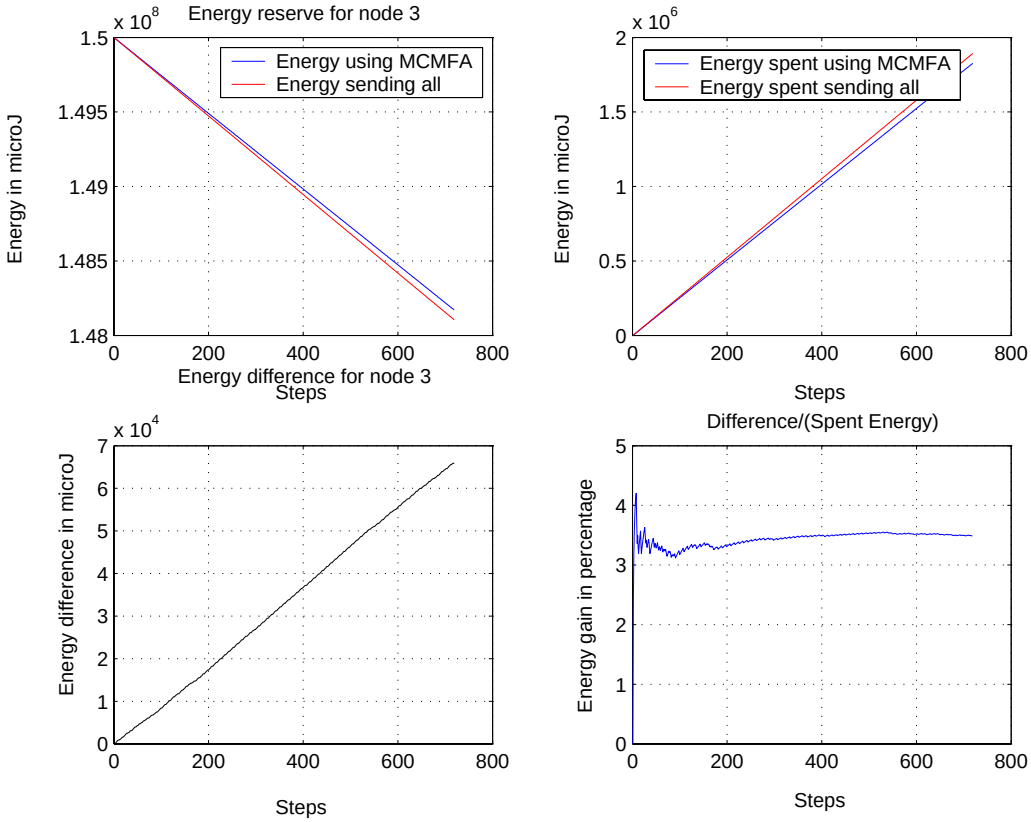
Σχήμα 6.17: Ενεργειακή κατάσταση κόμβου 2

Η συμπεριφορά του κόμβου 2 ως προς το πλήθος των μηνυμάτων και των bit που έλαβε και έστειλε είναι πρακτικά η ίδια, όπως αναμενόταν, και φαίνεται στο Σχήμα 6.18. Συνοπτικά ο κόμβος 2 έλαβε 335 μηνύματα (καταναλώνοντας συνολικά 5.2 mJ για 47640 bits) και έστειλε 175 από ένα μέγιστο 715 μηνυμάτων (ποσοστό 24.5%) καταναλώνοντας συνολικά 26.9 mJ για 37400 bits.

Ο κόμβος 3, ένας κόμβος επικοινωνίας που δέχεται μηνύματα από αισθητήριους κόμβους παρουσιάζει βελτίωση σε σχέση με την προηγούμενη θεώρηση. Πιο συγκεκριμένα, το ποσοστιαίο ενεργειακό κέρδος είναι ίσο με 3.5% έναντι 1.5%. Το καθαρό ενεργειακό κέρδος είναι πρακτικά το ίδιο, γύρω στα 70 mJ όπως φαίνεται και στο Σχήμα 6.19.

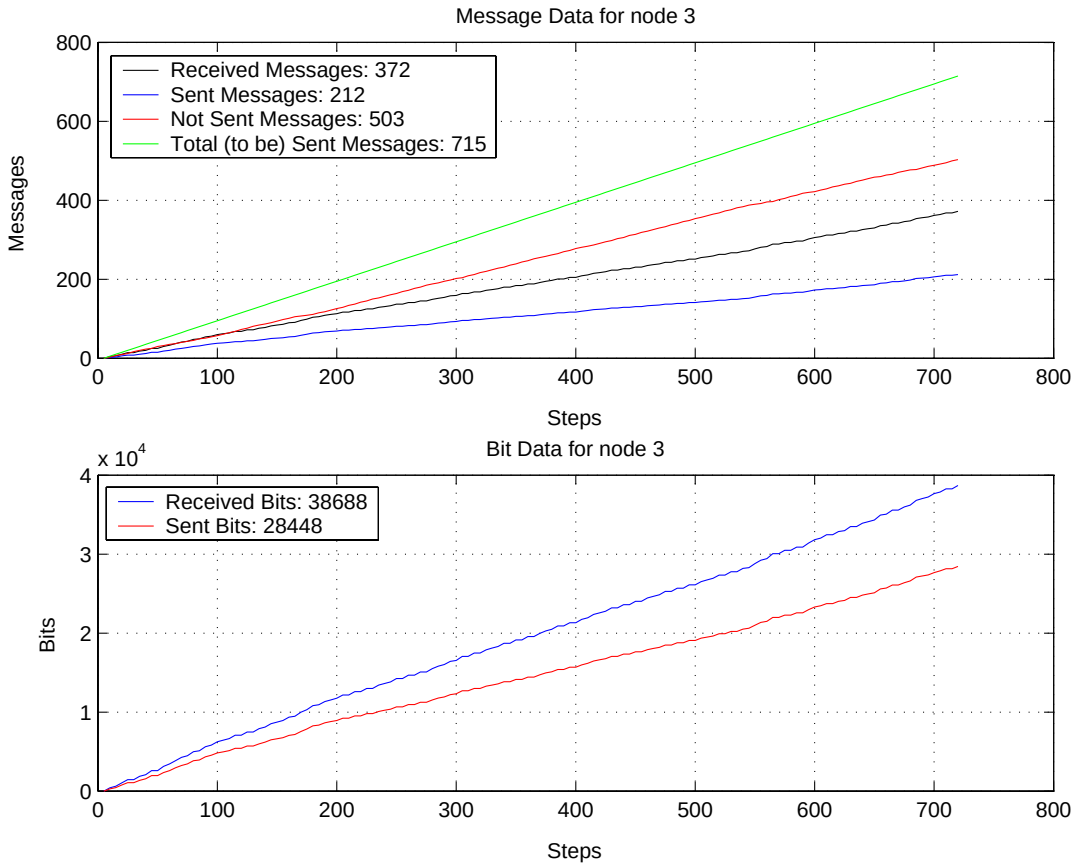


Σχήμα 6.18: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 2



Σχήμα 6.19: Ενεργειακή κατάσταση κόμβου 3

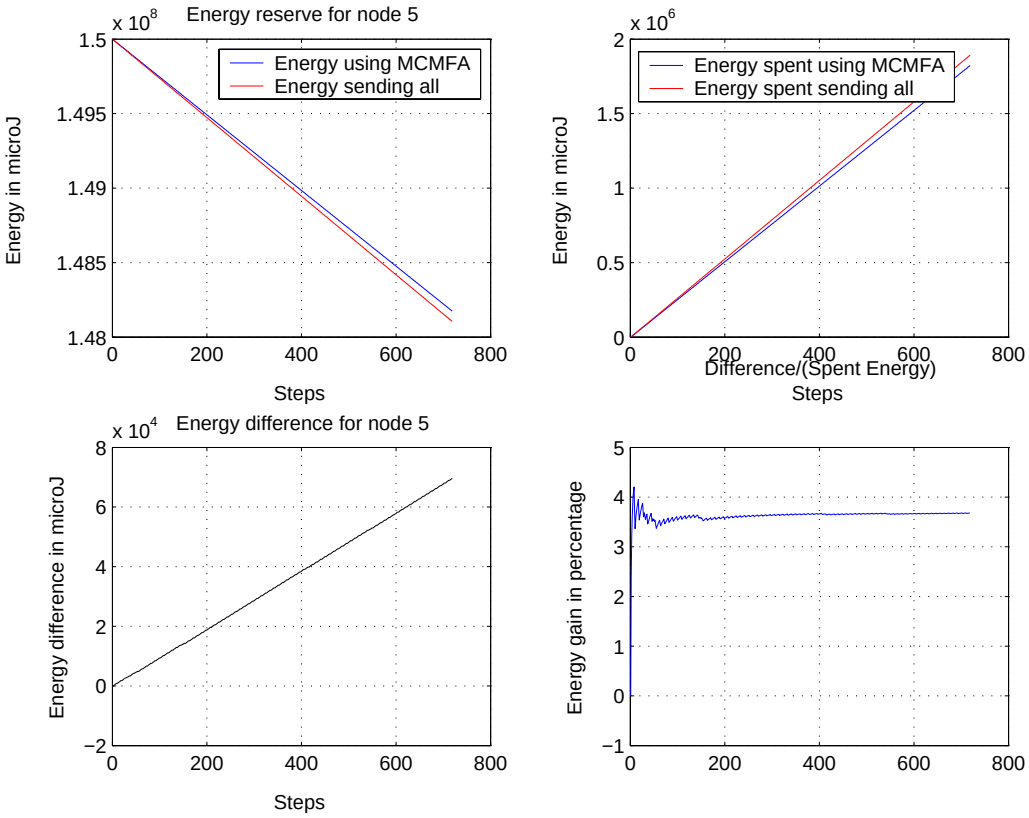
Αντίστοιχα σχόλια ισχύουν και για τις αποστολές και τις λήψεις του κόμβου 3. Ο κόμβος έλαβε σε 372 μηνύματα, 38688 bits και έστειλε μόνο το 29.7% από το μέγιστο αριθμό μηνυμάτων που θα έστελνε χωρίς την αρχιτεκτονική, δηλαδή έστειλε σε 212 μηνύματα 28448 bit. Οι τιμές αυτές φαίνονται στο Σχήμα 6.20.



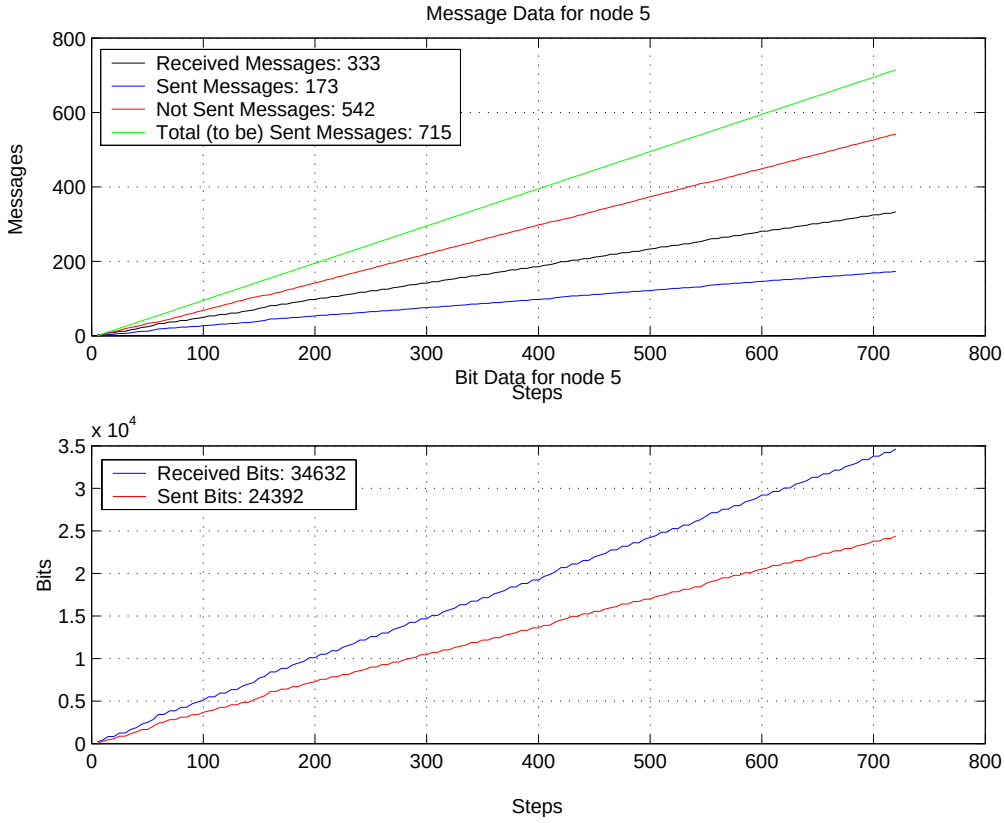
Σχήμα 6.20: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 3

Ο κόμβος 5 έχει έναν αντίστοιχο ρόλο στην ιεραρχία με τον κόμβο 3. Συγκεκριμένα, δέχεται και αυτός μηνύματα από αισθητήριο κόμβο. Ο κόμβος 5, λοιπόν, όπως φαίνεται και από το Σχήμα 6.21 έχει αυξημένο ποσοστιαίο ενεργειακό κέρδος, το οποίο φτάνει την τιμή του 4%, σε σχέση με το πρώτο πείραμα. Το καθαρό ενεργειακό κέρδος είναι, όπως αναμενόταν, 70 mJ.

Όπως φαίνεται στο Σχήμα 6.22 ο κόμβος 5 έλαβε 333 μηνύματα και έστειλε 173. Αυτό, μεταφρασμένο σε bit, ισοδυναμεί με λήψη 34632 bits και αποστολή 24392 bits. Μπορούμε να αναλογιστούμε ότι αυτός ο κόμβος θα λάμβανε, σε περίπτωση που αποστέλλονταν όλα τα μηνύματα, 72.6 Kbit ενώ τελικά λαμβάνει 33.8 Kbit κάτι που σημαίνει ότι μειώνεται περίπου στο μισό το κόστος λήψης δεδομένων, εξοικονομώντας 4.4 mJ. Αντίστοιχα, στέλνει 24392 bits ενώ θα έστελνε 108680 bits δηλαδή εξοικονομεί 60.7 mJ.



Σχήμα 6.21: Ενεργειακή κατάσταση κόμβου 5

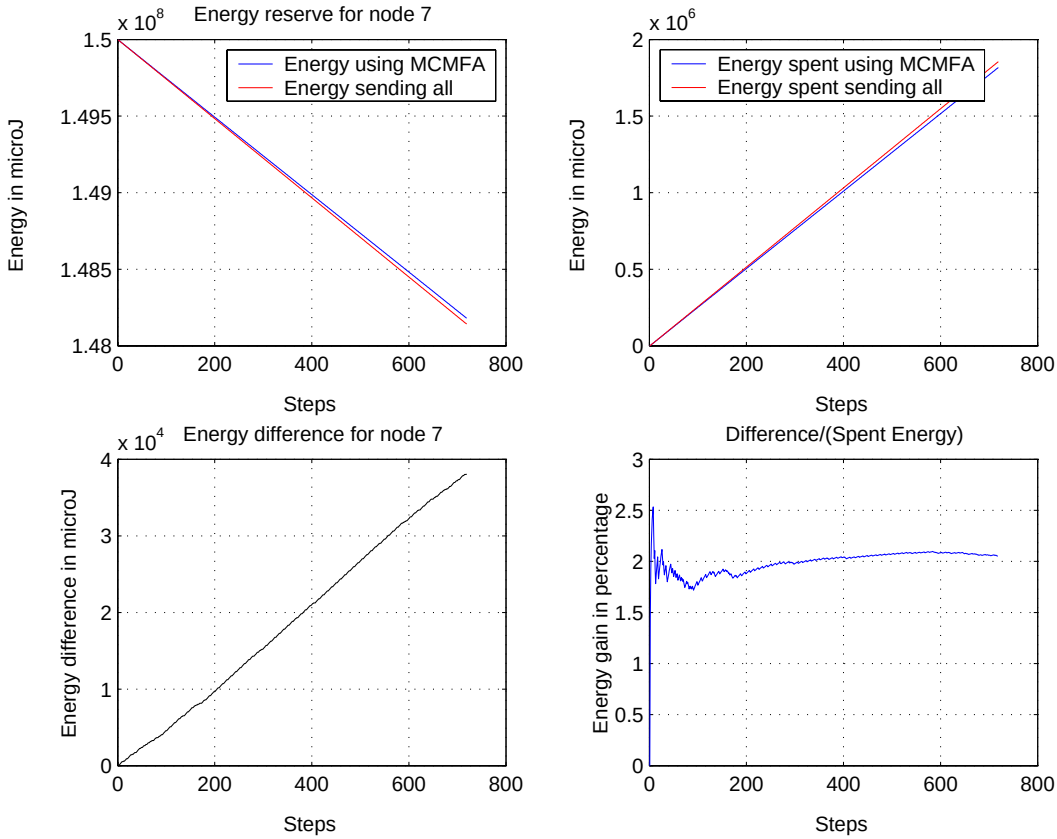


Σχήμα 6.22: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 5

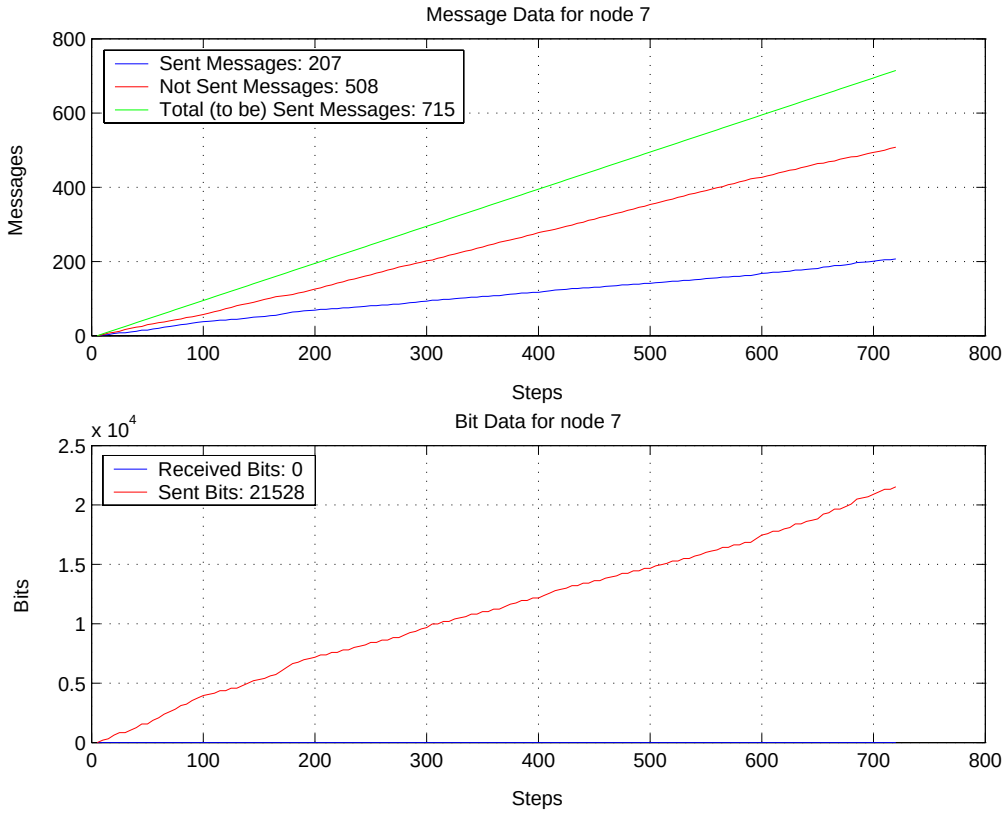
Οι δύο τελευταίοι κόμβοι που παρουσιάζονται στο δεύτερο πείραμα, στα Σχήματα 6.23 έως 6.26 είναι δύο αισθητήριοι κόμβοι, ο κόμβος 7 και ο κόμβος 12. Το ποσοστιαίο ενεργειακό κέρδος για τους δύο αυτούς κόμβους είναι από 2% περίπου και το καθαρό ενεργειακό κέρδος είναι περί τα 40 mJ. Είναι λογικό το ποσοστό του κέρδους να είναι μικρότερο σε αυτούς τους κόμβους γιατί οι λειτουργίες που επιτελούν γενικά είναι λιγότερες, οπότε υπάρχουν λιγότερα σημεία από τα οποία μπορεί να εξοικονομηθεί ενέργεια. Ένας αισθητήριος κόμβος δε λαμβάνει μηνύματα και επιπλέον στέλνει μόνο μία μέτρηση σε κάθε πλήρη κύκλο λειτουργίας.

Όπως φαίνεται στα Σχήματα 6.24 και 6.26 οι αισθητήριοι κόμβοι έστειλαν μόνο το 24.2% - 28.96% του μέγιστου πλήθους μηνυμάτων μειώνοντας έτσι την καταναλωθείσα ενέργεια κατά 12.9 mJ έως 15.5 mJ. Οι υπόλοιποι αισθητήριοι κόμβοι είχαν παρόμοια συμπεριφορά και, έτσι, δεν κρίθηκε σκόπιμο να παρουσιαστούν τόσο αναλυτικά τα δεδομένα για όλους τους αισθητήριους κόμβους.

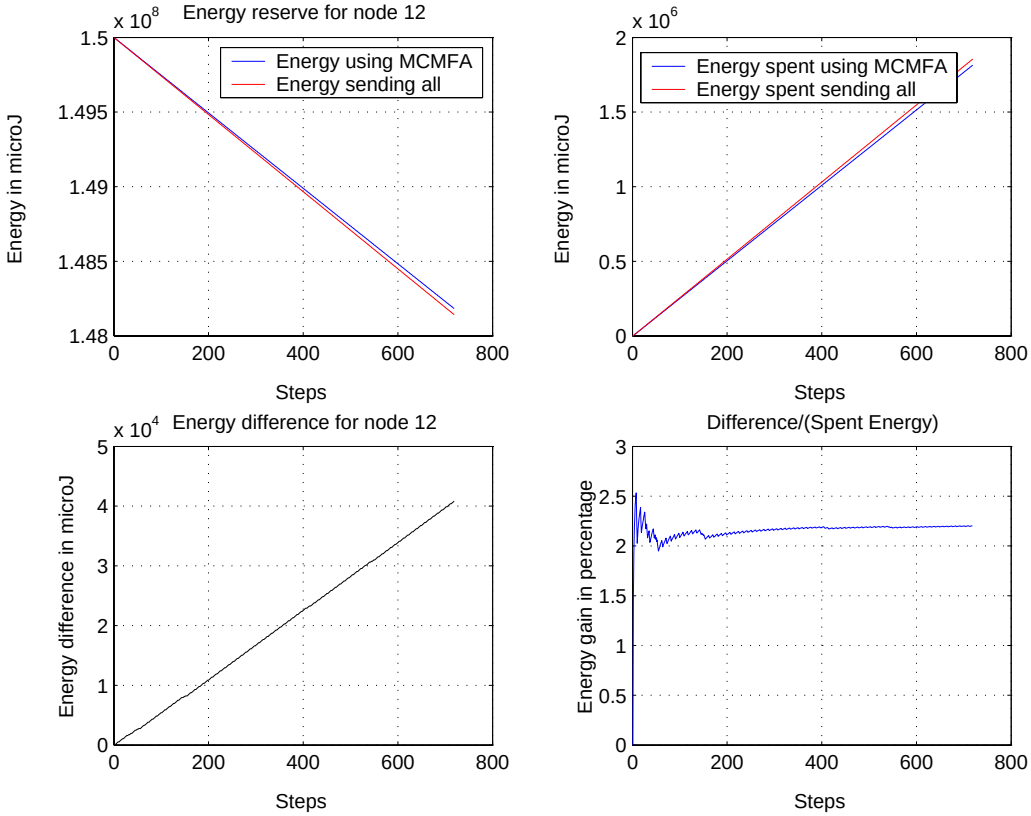
Συμπερασματικά, μπορούμε να πούμε ότι παρατηρείται μια βελτίωση του ποσοστιαίου ενεργειακού κέρδους στο δεύτερο πείραμα έναντι του πρώτου αφού η υπόθεση που χρησιμοποιούμε στην υλοποίηση οδηγεί σε μείωση της κατανάλωσης ενέργειας λόγο κακής διαχείρισης του χρόνου κατά τον οποίο ένας κόμβος παραμένει αδρανής.



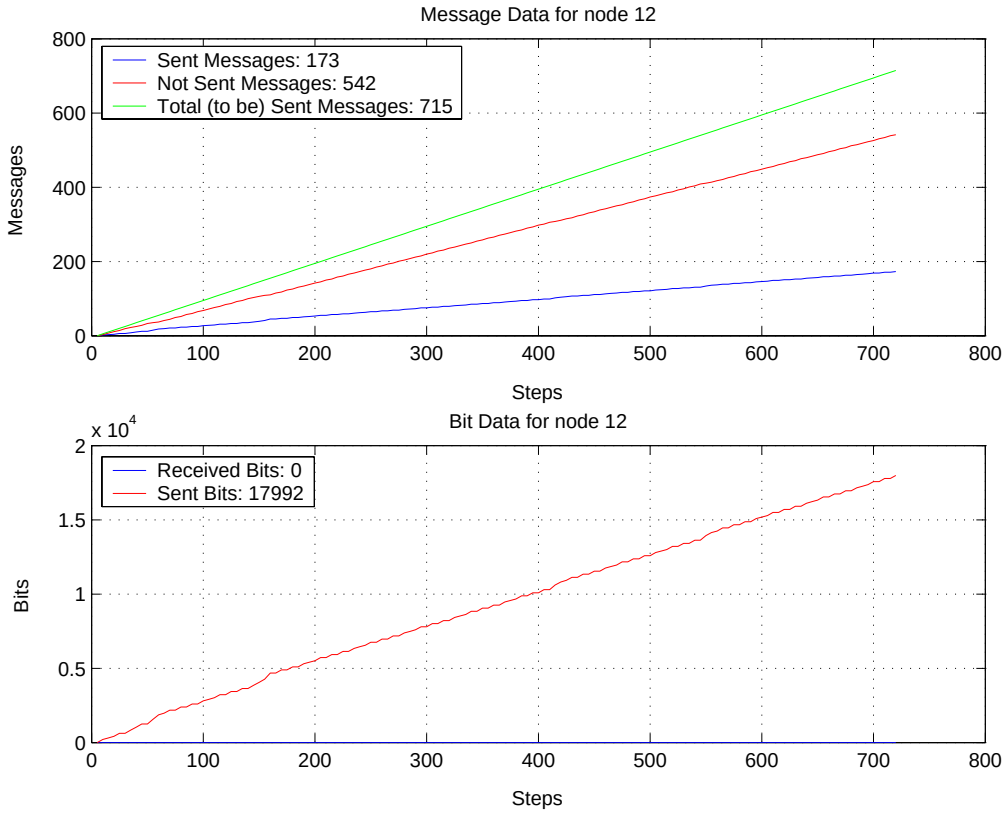
Σχήμα 6.23: Ενεργειακή κατάσταση κόμβου 7



Σχήμα 6.24: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 7



Σχήμα 6.25: Ενεργειακή κατάσταση κόμβου 12



Σχήμα 6.26: Αποστολές/λήψεις μηνυμάτων και bit κόμβου 12

ΚΕΦΑΛΑΙΟ 7

ΣΥΜΠΕΡΑΣΜΑΤΑ - ΠΡΟΤΑΣΕΙΣ

7.1 Συμπεράσματα

Στο προηγούμενο κεφάλαιο παρουσιάστηκαν κάποια αποτελέσματα από προσομοιώσεις της αρχιτεκτονικής που υλοποιήθηκε και προτάθηκε μέσα από την εργασία αυτή. Στο κεφάλαιο 4 παρουσιάστηκε η λογική της αρχιτεκτονικής και στο κεφάλαιο 5 σχολιάστηκε η υλοποίηση και οι σχεδιαστικές επιλογές. Από αυτά οδηγηθήκαμε σε κάποια συμπεράσματα ως προς την προτεινόμενη αρχιτεκτονική αλλά και γενικότερα.

Μια πρώτη, πολύ σημαντική, παρατήρηση είναι ότι σε εφαρμογές δικτύων αισθητήρων μας ενδιαφέρει να χρησιμοποιήσουμε αλγορίθμους που αποσκοπούν στην εξοικονόμηση ενέργειας. Προκειμένου να κατασκευαστούν αλγόριθμοι και τεχνικές δρομολόγησης αυτής της μορφής πρέπει να μελετηθεί πώς μπορεί να γίνει εξοικονόμηση ενέργειας, ποιές λειτουργίες καταναλώνουν πολλή ενέργεια και ποιές από τις λειτουργίες αυτές μπορούμε να αποφύγουμε. Είδαμε ότι ένας κόμβος πως καταναλώνει ενέργεια με αρκετούς τρόπους. Συγκεκριμένα, καταναλώνει ενέργεια εκετελώντας εντολές, στέλνοντας και λαμβάνοντας μηνύματα αλλά και καθ'όλη τη διάρκεια που παραμένει ανοιχτός, αφού κάποιο ρεύμα, έστω μικρής έντασης, τον διαπερνά. Η κατανάλωση ενέργειας είναι ο σημαντικότερος παράγοντας που επηρεάζει τη συμπεριφορά ενός δικτύου αισθητήρων και, όπως παρατηρήθηκε στις προσομοιώσεις, την αποτελεσματικότητα της αρχιτεκτονικής.

Μια σημαντική λεπτομέρεια είναι ότι ένας κόμβος μπορεί να παραμένει σε λειτουργία σε διάφορες καταστάσεις. Μπορεί να είναι αδρανής, σε κατάσταση αναμονής ή σε κατάσταση εξοικονόμησης ενέργειας. Αυτές οι τρεις καταστάσεις έχουν διαφορετική κατανάλωση ενέργειας και, μάλιστα, τη μεγαλύτερη έχει η κατάσταση αδράνειας και τη μικρότερη η κατάσταση εξοικονόμησης ενέργειας. Όπως έγινε αντιληπτό είναι σημαντικό να κατανέμεται ο χρόνος στις διάφορες καταστάσεις έξυπνα.

Ο πυρήνας της προτεινόμενης αρχιτεκτονικής είναι η αποφυγή περιττών αποστολών. Αυτό έγινε, επειδή σε όλα τα συστήματα ασυρμάτων αισθητήριων κόμβων η αποστολή μηνυμάτων είναι δαπανηρή και συγκεκριμένα η αποστολή ενός bit κοστίζει από 200 έως 1000 φορές περισσότερες από την εκτέλεση μιας εντολής. Έτσι, με χρήση μιας συνάρτησης που ονομάζεται συνάρτηση αξιολόγησης αποστολής, αποφασίζουμε για κάθε πακέτο που πρόκειται να προωθηθεί, εάν κάτι τέτοιο κρίνεται σκόπιμο. Η απόφαση αυτή βασίζεται σε τρεις μετρικές, που αφορούν το ενεργειακό απόθεμα, το σφάλμα που εισάγεται από τη μη προώθηση και το χρόνο. Μετά από τις προσομοιώσεις βρέθηκε ότι αυτή η αρχιτεκτονική μπορεί να οδηγήσει σε μη αμελητέο ενεργειακό κέρδος εάν συνδυαστεί με την έξυπνη διαχείριση του χρόνου παραμονής στις διάφορες καταστάσεις για τους κόμβους του δικτύου.

7.2 Προτάσεις

Τα αποτελέσματα που παρουσιάστηκαν αφήνουν υποσχέσεις για καλύτερη απόδοση. Ανωτέρω, παρουσιάστηκε περιληπτικά και ένα μοντέλο επαναφόρτισης. Αυτό το μοντέλο μπορεί να συνεισφέρει σημαντικά στη δημιουργία πραγματικών δεδομένων αλλά και στην ενσωμάτωση μιας λογικής που δεν έχει ακόμα κυριαρχήσει στα δίκτυα αισθητήρων. Η επαναφόρτιση των συσσωρευτών των κόμβων είναι μια ιδέα που θα

βοηθήσει στην επιμήκυνση της ζωής των δικτύων και στη δημιουργία πιο εύρωστων δικτύων. Η πρόοδος της τεχνολογίας, μας επιτρέπει να προσθέσουμε σε έναν αισθητήρα μεγέθους ενός κέρματος (του ενός δολλαρίου) έναν μικρό ηλιακό συσσωρευτή εμβαδού, τάξης μεγέθους, ενός τετραγωνικού εκατοστού. Επίσης, μπορούν να μελετηθούν και άλλοι πιθανοί τρόποι επαναφόρτισης, όπως χρήση χημικής και κινητικής ενέργειας.

Εκτός από το μοντέλο επαναφόρτισης που αναφέρθηκε παραπάνω, η αρχιτεκτονική που παρουσιάστηκε μπορεί να δοκιμαστεί με διαφορετικές δικτυακές τοπολογίες και με δυναμικά πρωτόκολλα δρομολόγησης, που να επιλέγουν το μονοπάτι το οποίο θα ακολουθήσουν οι ροές, ανάλογα με την τρέχουσα κατάσταση του δικτύου. Ακόμη, στον κώδικα θα μπορούσε να προστεθεί καθολικός μηχανισμός συντονισμού της συμπεριφοράς του δικτύου. Πιο συγκεκριμένα, ο κόμβος-καταβόθρα, εκτός από τη συλλογή των ροών δεδομένων θα μπορούσε να διαδραματίζει και το ρόλο του συντονιστή του δικτύου και ανάλογα με τα δεδομένα που δέχεται να αραιώνει ή να πυκνώνει το ρυθμό αποστολής μηνυμάτων από τους κόμβους που πραγματοποιούν τις μετρήσεις. Ένα τέτοιο χαρακτηριστικό, θα μπορούσε να επιμηκύνει τη ζωή ενός δικτύου αισθητήρων, αφού οι αισθητήρες δε θα είναι απαραίτητο να λειτουργούν συνέχεια, αλλά μόνο όταν κρίνεται αναγκαίο από την εφαρμογή με δυναμικό τρόπο.

Τέλος, η αρχιτεκτονική που συζητήθηκε παραπάνω εφαρμόστηκε σε στατικά δίκτυα αισθητήρων. Πολύ ενδιαφέρουσα θα ήταν η προσπάθεια να προσαρμοστεί κατάλληλα, ώστε να είναι δυνατή η εφαρμογή του και σε κινητά δίκτυα αισθητήρων, όπου η έννοια της τοπολογίας αλλάζει με το πέρασμα του χρόνου και ο τρόπος μεταδόσεων και επικοινωνίας διαφέρει.

Παράρτημα Α: Ορισμοί δικτύων αισθητήρων

Η έρευνα που αφορά τα δίκτυα αισθητήρων δέχεται επιρροές από πολλούς άλλους επιστημονικούς τομείς όπως η επεξεργασία σήματος, τα δίκτυα και τα πρωτόκολλα επικοινωνίας, η διαχείριση βάσεων δεδομένων, οι αλγόριθμοι, οι αρχιτεκτονικές και τα ενσωματωμένα συστήματα. Στη συνέχεια, δίνονται ορισμοί εννοιών-κλειδιά που αφορούν τα δίκτυα αισθητήρων.

- *Αισθητήρας (Sensor):* Ένας μετατροπέας, ο οποίος μετασχηματίζει ένα φυσικό φαινόμενο όπως ζέστη, φως, ήχος, ή κίνηση σε ηλεκτρικό ή άλλο σήμα, το οποίο με τη σειρά του μπορεί να χρησιμοποιηθεί από άλλες συσκευές.
- *Αισθητήριος κόμβος (Sensor node):* Η βασική μονάδα σε ένα δίκτυο αισθητήρων, με ενσωματωμένο αισθητήρα, επεξεργαστή, μνήμη και πηγή ενέργειας. Πολλές φορές αναφερόμαστε σε αυτόν και ως κόμβο. Όταν ο κόμβος έχει ένα μεμονωμένο αισθητήρα τότε ο κόμβος μερικές φορές αναφέρεται και ως αισθητήρας γεγονός που μπορεί να οδηγήσει σε σύγχυση.
- *Τοπολογία δικτύου (Network topology):* Ένας συνεκτικός γράφος στον οποίο κόμβοι είναι οι αισθητήριοι κόμβοι και ακμές είναι οι συνδέσεις για επικοινωνία. Σε ένα ασύρματο δίκτυο μια σύνδεση αναπαριστά μια one-hop διασύνδεση και ως γείτονες ενός κόμβου θεωρούνται όσοι κόμβοι βρίσκονται στο βεληνεκές εκπομπής και λήψης του.
- *Δρομολόγηση (Routing):* Η διαδικασία για τον προσδιορισμό ενός μονοπατιού για ένα πακέτο στο δίκτυο από τον κόμβο-πηγή στον προορισμό του.
- *Στο δίκτυο (In-network):* Ένας τύπος επεξεργασίας όπου τα δεδομένα είναι αντικείμενο επεξεργασίας και συνδυάζονται κοντά στο σημείο παραγωγής τους. Σε περιπτώσεις που χρησιμοποιείται ως επίθετο, η φράση αυτή αποδίδεται και ως “ενδοδικτυακός”.
- *Συνεργατική επεξεργασία (Collaborative processing):* Οι αισθητήρες επεξεργάζονται δεδομένα που έχουν προέλθει από πολλαπλές πηγές. Συνεργατικά, με σκοπό να βοηθήσουν υψηλότερα επίπεδα. Αυτή η επεξεργασία απαιτεί συνήθως επικοινωνία ανάμεσα σε ένα σύνολο από κόμβους.
- *Κατάσταση (State):* Ένα στιγμιότυπο ενός φυσικού περιβάλλοντος ή του ίδιου του συστήματος (π.χ. η κατάσταση του δικτύου).
- *Αβεβαιότητα (Uncertainty):* Η κατάσταση στην οποία υπεισέρχεται η πληροφορία λόγω θορύβου στις μετρήσεις των αισθητήρων ή λόγω έλλειψης στοιχείων στη μοντελοποίηση. Η αβεβαιότητα επηρεάζει την ικανότητα του συστήματος να εκτιμά με ακρίβεια την κατάσταση, και γι’αυτό πρέπει να μοντελοποιείται με προσοχή.
- *Έργο (Task):* Χωρίζονται σε δύο κατηγορίες, στα high level system tasks όπως μέτρηση, επικοινωνία, επεξεργασία και εντοπισμός πόρων και στα application tasks όπως ανίχνευση, ταξινόμηση ή εντοπισμός.
- *Ανίχνευση (Detection):* Η διαδικασία για την ανακάλυψη της ύπαρξης ενός φυσικού φαινομένου. Ένας φωρατής που βασίζεται σε κάποιο κατώφλι μπορεί να ενημερώνει, όταν εμφανίζονται ενδείξεις του φαινομένου συγκρίσιμες με την τιμή του κατωφλίου.

- *Ταξινόμηση (Classification):* Η ταξινόμηση σε συγκεκριμένη κατηγορία ενός συνόλου φυσικών φαινομένων.
- *Εντοπισμός και ιχνηλασία (Localization and tracking):* Η εκτίμηση της κατάστασης μιας φυσικής οντότητας, όπως ένα φυσικό φαινόμενο ή ένας αισθητήριος κόμβος από το σύνολο των μετρήσεων. Η ιχνηλάτηση παράγει σειρές αποτιμήσεων με την πάροδο του χρόνου.
- *Πόροι (Resources):* Οι πόροι περιλαμβάνουν αισθητήρες, συνδέσεις επικοινωνίας, επεξεργαστές, ενσωματωμένη μνήμη και ενεργειακά αποθέματα του κόμβου. Η διανομή πόρων εκχωρεί πόρους σε έργα βελτιστοποιώντας την απόδοση.
- *Αποστολή αισθητήρα (Sensor tasking):* Η ανάθεση ενός συγκεκριμένου έργου σε έναν αισθητήρα και ο έλεγχος της κατάστασης του αισθητήρα ώστε να διεκπεραιώσει το έργο.
- *Υπηρεσίες κόμβων (Node services):* Αναφέρεται σε υπηρεσίες όπως συγχρονισμός και εντοπισμός οι οποίες επιτρέπουν σε εφαρμογές να γνωρίζουν τις ιδιότητες των κόμβων, και στους κόμβους να οργανώνονται σε χρήσιμα δίκτυα.
- *Αποθήκευση δεδομένων (Data storage):* Οι πληροφορίες που προέρχονται από αισθητήρες αποθηκεύονται, ευρετηριάζονται και καθίστανται προσπελάσιμα από εφαρμογές. Η αποθήκευση μπορεί να είναι τοπική εκεί που παράγονται τα δεδομένα, κατανεμημένα στο δίκτυο ή συγκεντρωμένα σε λίγα σημεία (αποθήκες).
- *Ενσωματωμένο λειτουργικό σύστημα (Embedded operating system):* Είναι το σύστημα που υποστηρίζει τις εφαρμογές των δικτύων αισθητήρων κατά την εκτέλεση τους. Ένα ενσωματωμένο λειτουργικό σύστημα παρέχει ένα αφηρημένο στιγμιότυπο των πόρων του συστήματος και ένα σύνολο δυνατοτήτων.
- *Απόδοση συστήματος (System performance):* Είναι τα αφηρημένα χαρακτηριστικά των ιδιοτήτων ενός συστήματος όπως, κλιμάκωση, ευρωστία, μακροζωία του δικτύου, όπου η κάθε μια ιδιότητα υπολογίζεται με τη χρήση μιας διαφορετικής μετρικής.
- *Μετρική αξιολόγησης (Evaluation metric):* Είναι μια μετρήσιμη ποσότητα, η οποία προσδιορίζει πόσο καλά λειτουργεί ένα σύστημα σε απόλυτη κλίμακα, σε σχέση με τα πακέτα που χάνονται, το χρόνο αντίδρασης του δικτύου, τα λάθη εντοπισμού κ.ά. Μια μέθοδος αξιολόγησης είναι η διαδικασία σύγκρισης των τιμών της μετρικής που έχουν προέλθει από ένα σύνολο πειραμάτων, σε σχέση με αυτές που έχουν προέλθει από ένα άλλο σύστημα δοκιμών.

Παράρτημα Β: Ενεργειακή συμπεριφορά κόμβων

Κατανάλωση ενέργειας κατά την εκτέλεση εντολών και άλλων διεργασιών

Στην παρούσα εργασία έχουν χρησιμοποιηθεί κάποια δεδομένα που αφορούν στην ενεργειακή συμπεριφορά των κόμβων. Τα δεδομένα που έχουν χρησιμοποιηθεί αναφέρονται σε κόμβους τύπου mica2, που είναι και τα κλασσικά Berkeley Motes.

Η πρώτη βασική υπόθεση είναι ότι η τάση λειτουργίας των κόμβων είναι 3 V. Αυτό ισχύει αφού οι κόμβοι αυτοί είναι, γενικά, εφοδιασμένοι με δυο μπαταρίες AA, τάσης 1.5 V η κάθε μία, σε σειρά, άρα η συνολική τάση λειτουργίας είναι 3 V. Στην πραγματικότητα, όσο αποφορτίζονται οι μπαταρίες, η τάση μεταβάλλεται και, πιο συγκεκριμένα, μειώνεται, αλλά σε καμμία περίπτωση δεν πέφτει κάτω από 2.5 V. Σύμφωνα με στοιχεία που βρίσκουμε σε διάφορα άρθρα ([26], [27]), ισχύει ότι η κατανάλωση ενέργειας ανά εντολή επεξεργαστή είναι 4 nJ/instruction. Επίσης, η κατανάλωση στην περίπτωση της μετάδοσης και της λήψης είναι 720 nJ/bit και 110 nJ/bit αντίστοιχα. Σύμφωνα με τον πίνακα που μπορούμε να βρούμε στο διαδίκτυο[1] η κατανάλωση κατά την αποστολή ενός μηνύματος είναι:

$$E_{bit} = 62.4\ \mu\text{s/bit} \cdot 3.72\ \text{mA} \cdot 3\text{V} \simeq 700 \cdot 10^{-6} \cdot 10^{-3}\ \text{s} \cdot \text{W/bit} \Rightarrow E_{bit} \simeq 700\,\text{nJ/bit}$$

Με τον παραπάνω υπολογισμό επαληθεύεται σε μεγάλο βαθμό η εκτίμηση των προαναφερθέντων άρθρων. Αντίστοιχα μπορεί να υπολογιστεί και η κατανάλωση της λήψης ενός μηνύματος. Με την ίδια μεθοδολογία μπορούμε να υπολογίσουμε και την κατανάλωση ενέργειας που γίνεται σε κάποιον κόμβο όταν βρίσκεται σε μία εκ των τριών καταστάσεων που αναφέρονται στο κεφάλαιο 4 και συγκεκριμένα σε κατάσταση αδράνειας, αναμονής και εξοικονόμησης ενέργειας. Λαμβάνοντας υπόψη ότι το ρεύμα που διαπερνά τον κόμβο στις παραπάνω κατάστασεις είναι 3.2 mA, 216 μA και 110 μA αντίστοιχα, και υποθέτοντας ότι η τάση είναι ίση με 3 V, μπορούμε να συμπεράνουμε ότι η ισχύς που καταναλώνεται στις τρεις καταστάσεις αυτές είναι 9.6 mW, 648 μW και 330 μW αντίστοιχα.

Κέρδος ενέργειας κατά τη φόρτιση

Σύμφωνα με τη βιβλιογραφία μπορούμε να υποθέσουμε ότι ένας κόμβος ενός Α.Δ.Α. μπορεί να εφοδιαστεί με μικροσκοπικούς ηλιακούς συσσωρευτές εμβαδού 1 cm². Αυτοί οι ηλιακοί συσσωρευτές αναλόγως αν χρησιμοποιούνται σε εσωτερικό ή εξωτερικό χώρο μπορούν να αποδώσουν ισχύ στο συσσωρευτή του τρέχοντος κόμβου. Πιο συγκεκριμένα, σε μια ηλιόλουστη μέρα ένας τέτοιος συσσωρευτής μπορεί να αποδώσει 15 mW/cm² ενώ αν επικρατεί σκιά τότε θα αποδώσει 150 μW/cm². Αν ένας τέτοιος συσσωρευτής χρησιμοποιηθεί σε εσωτερικό χώρο θα αποδώσει από 6 μW/cm² έως 570 μW/cm² ανάλογα με το αν υπάρχει εσωτερικός φωτισμός με χρήση ηλεκτρικού λαμπτήρα ή όχι.

[1] http://www.eecs.harvard.edu/~shnayder/ptossim/mica2bench/summary.html

Μπαταρίες

Όπως σημειώνεται ανωτέρω οι κόμβοι στους οποίους αναφερόμαστε είναι εφοδιασμένοι με δύο AA μπαταρίες. Τέτοιες μπαταρίες εχουν γενικά αρκετή ενέργεια για να μπορεί να είναι λειτουργικός ένας κόμβος για μεγάλο χρονικό διαστημα. Σύμφωνα με τη βιβλιογραφία, μια μπαταρία AA έχει 1100 mAh έως 1800 mAh και άρα για δύο μπαταρίες AA, που έχουν τάση από 1.5 V η κάθε μια και συνολική τάση σε παράταξη σε σειρά 3 V, η ενέργεια είναι ίση με:

$$E = \begin{Bmatrix} 1100\,\text{mAh} \cdot 3600\,\text{s/h} \cdot 3\,\text{V} \\ \varepsilon\omega\varsigma \\ 1800\,\text{mAh} \cdot 3600\,\text{s/h} \cdot 3\,\text{V} \end{Bmatrix} = \begin{Bmatrix} 11880\,\text{J} \\ \varepsilon\omega\varsigma \\ 19440\,\text{J} \end{Bmatrix}$$

Σύμφωνα με άλλες πηγές μια μπαταρία ανάλογα με τη χημική της σύσταση έχει την εξής πυκνότητα ενέργειας:

Πρωτεύουσες μπαταρίες			
Χημική σύσταση	Zinc-air	Lithium	Alkaline
Ενέργεια (J/cm³)	3780	2880	1200
Δευτερεύουσες μπαταρίες			
Χημική σύσταση	Lithium	NiMHd	NiCd
Ενέργεια (J/cm³)	1080	860	650

Με βάση τον παραπάνω πίνακα φτάνουμε σε παρόμοια αποτελέσματα αφού μια μπαταρία AA έχει γύρω στα 4 cm³ όγκο και άρα η ενέργεια που υπάρχει σε δύο αλκαλικές μπαταρίες ή σε δύο μπαταρίες λιθίου είναι ίση με:

$$E = \begin{Bmatrix} 8\,\text{cm}^3 \cdot 1200\,\text{J/cm}^3 \\ \varepsilon\omega\varsigma \\ 8\,\text{cm}^3 \cdot 2880\,\text{J/cm}^3 \end{Bmatrix} = \begin{Bmatrix} 9600\,\text{J} \\ \varepsilon\omega\varsigma \\ 23040\,\text{J} \end{Bmatrix}$$

Παράρτημα Γ: Παράδειγμα Εκτέλεσης

Η εφαρμογή που αναπτύχθηκε στα πλαίσια αυτής της εργασίας είναι γραμμένη σε nesC και για να εκτελεστεί απαιτεί το λειτουργικό σύστημα TinyOS και τον προσομοιωτή κόμβων TOSSIM που βρίσκεται ενσωματωμένος σε αυτό. Το TinyOS είναι ένα λειτουργικό γραμμένο για ασύρματους αισθητήριους κόμβους αλλά για τις ανάγκες προσομοιώσεων μπορεί να εγκατασταθεί σε προσωπικό υπολογιστή, είτε σε λειτουργικό τύπου LINUX είτε σε λειτουργικό τύπου Microsoft Windows ® με τη χρήση του emulator κελύφους cygwin. Όλα τα απαραίτητα αρχεία μπορούν να βρεθούν στο δικτυακό τόπο http://www.tinyos.net/download.html.

Ο κώδικας της εφαρμογής πρέπει να τοποθετηθεί στον υποκατάλογο του καταλόγου του TinyOS που ονομάζεται apps και να γίνει compiled με τη χρήση της εντολής:

```
myserver:/opt/tinyos-1.x/apps/Mcmfa/> make pc
```

Αυτή η εντολή θα παράγει το εκτελέσιμο που μπορεί να τρέξει σε προσομοίωση σε προσωπικό υπολογιστή. Στη συνέχεια για να εκτελεστεί η εφαρμογή πρέπει να δοθεί η εντολή:

```
myserver:/opt/tinyos-1.x/apps/Mcmfa/> ./build/pc/main.exe –b=1 <n>
```

όπου το όρισμα –b=1 σημαίνει ότι όλοι οι κόμβοι θα αρχικοποιηθούν σε ένα δευτερόλεπτο και το δεύτερο όρισμα είναι μια τιμή, το πλήθος των προς προσομοίωση κόμβων.

Για τις διάφορες παραλλαγές εκτελέσεων ο χρήστης καλείται να επέμβει στον κώδικα, να πειραματιστεί με τις τιμές των διάφορων σταθερών και, ίσως, των αλγορίθμων παρεμβολής, αποφόρτισης και επαναφόρτισης.

Παράρτημα Δ: Κώδικας εφαρμογής

Mcmfa.nc

```
1   /*File name: Mcmfa.nc*/
2   /**
3   Πτυχιακή Εργασία - Υλοποίηση MCMFA
4   MCMFA - Multi-Criteria Message Forwarding Architecture
5   Αθανασούλης Μανούσος-Γαβριήλ     ΑΜ:018200100002
6   Αλαγιάννης Ιωάννης                ΑΜ:018200100011
7   **/
8
9   includes Mcmfa;
10
11  configuration Mcmfa {
12  }
13
14  implementation {
15      components Main, McmfaM, TimerC, GenericComm as Comm, TinyAlloc, RandomLFSR;
    /*Components που χρησιμοποιούνται*/
16      /*Συναρμολόγηση components συνδέοντας interfaces*/
17      Main.StdControl -> TimerC.StdControl;
18      Main.StdControl -> McmfaM.StdControl;
19      Main.StdControl -> Comm;
20
21      McmfaM.Timer -> TimerC.Timer[unique("Timer")];
22      McmfaM.MySend -> Comm.SendMsg[Msg];
23      McmfaM.MyReceive -> Comm.ReceiveMsg[Msg];
24      McmfaM.CommControl -> Comm;
25      McmfaM.MemAlloc -> TinyAlloc;
26
27      McmfaM.MyRand -> RandomLFSR;
28  }
29
30
31
32
33
```

```
1   /*File name: McmfaM.nc*/
2   /**
3   Πτυχιακή Εργασία - Υλοποίηση MCMFA
4   MCMFA - Multi-Criteria Message Forwarding Architecture
5   Αθανασούλης Μανούσος-Γαβριήλ     ΑΜ:018200100002
6   Αλαγιάννης Ιωάννης                ΑΜ:018200100011
7   **/
8   /**
9   Ο παρακάτω κώδικας έχει σχεδιαστεί για προσομοίωση με χρηση του TOSSIM, ώστε να ελεγχθεί η αποτελεσματικότητα
10  της αρχιτεκτονικής και να πραγματοποιηθεί συλλογή στατιστικών στοιχείων που αφορούν τις αποστολές, τις λήψεις την
11  κατανάλωση ενέργειας κ.ά. Ο κώδικας χρειάζεται τροποποιήσεις για να χρησιμοποιηθεί απευθείας σε κόμβους. Σε πρώτο
12  στάδιο, αρχικοποιείται μια τοπολογία δυαδικού δέντρου με στην οποία κάθε κόμβος γνωρίζει τον πατέρα του(στον οποίο
13  στέλνει) και το πλήθος των αισθητήριων κόμβων απο τους οποίους μπορεί να λάβει μετρήσεις. Στη συνέχεια, να αναφερθούμε
14  στα πιο σημαντικά τμήματα της υλοποίησης, ενώ και εκτενή σχόλια που αφορούν τη λογική αλλά και τις σχεδιαστικές επιλογές
15  υπάρχουν και στο μέρος Β' του κειμένου.
16   **/
17  includes Mcmfa;
18
19  module McmfaM
20  {
21      /*Το interface που παρέχει το Component*/
22      provides
23      {
24          interface StdControl;
25      }
26      /*Τα interfaces που χρησιμοποιεί το Component*/
27      uses
28      {
29          /*Interface το οποίο παρέχει έναν χρονομέτρη που χρησιμοποιείται για παράγονται events σε κανονικά διαστήματα*/
30          interface Timer;
31          /*Standard Control interface του TinyOS που χρησιμοποιείται για αρχικοποίηση*/
32          interface StdControl as CommControl;
33          /*Interface το οποίο χρησιμοποιείται για αποστολή AM (Active Messages) μηνυμάτων*/
34          interface SendMsg as MySend;
35          /*Interface για λήψη πακέτων*/
36          interface ReceiveMsg as MyReceive;
37          /*Interface για δυναμική δέσμευση μνήμης*/
38          interface MemAlloc;
39          /*Interface το οποίο παρέχει μια ψευδοτυχαία γεννήτρια αριθμών(δεν χρησιμοποιείται άμεσα, αποτελεί τμήμα του μοντέλου
    επαναφόρτισης)*/
40          interface Random as MyRand;
41      }
42  }
43
44  /*Υλοποίηση του Component Mcmfa*/
45  implementation
46  {
47  /*Στο πρώτο τμήμα του κώδικα ορίζονται μεταβλητές και σταθερές οι οποίες χρησιμοποιούνται καθολικά σε όλο το πρόγραμμα*/
48  /*************DEFINES*************/
49      /*Λογικές σταθερές*/
50      #define SUCCESS 1
51      #define FAIL    0
52      #define TRUE    1
53      #define FALSE   0
54
55      /*Σταθερές καθορισμού χρονικών στιγμών στην προσομοίωση*/
56      #define STEP                16   /*Αριθμός των clicks σε κάθε βήμα (step)*/
```

```
57        #define HEART_BEAT_FREQ      9   /*Συχνότητα ενεργοποίησης μηνυμάτων Heart-Beat - ανά πόσα steps*/
58        #define CLICK              500   /*Διάρκεια click, η τιμή είναι σε millisec και αφορά το χρόνο ενεργοποίησης του
      Timer στην προσομοίωση*/
59
60        #define SENSE_SEND           2 /*Click του step στο οποίο ξεκινά η διαδικασία για την αποστολή μέτρησης*/
61        #define SINK_REC_DONE        6 /*Click του step στο οποίο θεωρούμε πως ο κόμβος δεξαμενή έχει λάβει όσα μηνύματα
      έπρεπε να λάβει και ξεκινά η επεξεργασία τους*/
62        /*Οι παρακάτω σταθερές έχουν την ίδια λογική αλλά ανάλογα με το βάθος στο οποίο βρίσκεται ένας κόμβος
63        φροντίζουμε οι κόμβοι διαδοχικών επιπέδων να έχουν διαφορετικά παράθυρα αποστολής ώστε να επιτυγχάνεται
64        συγχρονιμός στις αποστολές και στις λήψεις μηνυμάτων.*/
65        #define COMM_REC_DONE_1      6 /*Click του step στο οποίο ο ενδιάμεσος κόμβος έχει λάβει όσα μηνύματα έπρεπε να λάβει
      και ξεκινά η επεξεργασία τους*/
66        #define COMM_SEND_1          8 /*Click του step στο οποίο ο ενδιάμεσος κόμβος προωθεί τα μηνύματα του (εφόσον
      πρόκειται να στείλει)*/
67        #define COMM_REC_DONE_2     12 /*Click του step στο οποίο ο ενδιάμεσος κόμβος έχει λάβει όσα μηνύματα έπρεπε να λάβει
      και ξεκινά η επεξεργασία τους*/
68        #define COMM_SEND_2         14 /*Click του step στο οποίο ο ενδιάμεσος κόμβος προωθεί τα μηνύματα του (εφόσον
      πρόκειται να στείλει)*/
69
70        /*Σταθερές που καθορίζουν την ενεργειακή κατανάλωση σε nJ*/
71        #define EPI             4 /*Κατανάλωση για την εκτέλεση μιας εντολής ενός κύκλου στη CPU(nJ/instruction)*/
72        #define EPBS          720 /*Κατανάλωση για την αποστολή ενός bit πληροφορίας(nJ/bit)*/
73        #define EPBR          110 /*Κατανάλωση για τη λήψη ενός bit πληροφορίας(nJ/bit)*/
74        #define EPSI  9600000 /*Κατανάλωση όταν το CPU βρίσκεται σε κατάσταση IDLE(nJ/sec)*/
75        #define EPSSB  330000 /*Κατανάλωση όταν το CPU βρίσκεται σε κατάσταση STANDBY(nJ/sec)*/
76        #define EPMC           62 /*Ενέργεια που καταναλώνεται για αλλαγή της κατάσταση του CPU(nJ/mode change)*/
77
78        /*Διαφορετικοί τύποι κατανάλωσης ενέργειας*/
79        #define T_CHARGE    4 /*Επαναφόρτιση (είναι απενεργοποιημένη)*/
80        #define T_IDLE      3 /*Κατάσταση IDLE*/
81        #define T_SEND      2 /*Περίπτωση αποστολής μηνύματος*/
82        #define T_RECEIVE   1 /*Περίπτωση λήψης μηνύματος*/
83        #define T_INSTR     0 /*Κατάσταση ACTIVE, υπολογισμός κατανάλωσης ανάλογα με τις εντολές που εκτελεί ο επεξεργαστής*/
84
85        /*Σταθερές που καθορίζουν το ενεργειακό κέρδος σε nJ/sec για το μοντέλο επαναφόρτισης(ΕΙΝΑΙ ΑΠΕΝΕΡΓΟΠΟΙΗΜΕΝΟ) */
86        /*Εξωτερικό χώρο*/
87        #define CHARGE_DIRECT_SUN      15000 /*Φόρτιση από απευθείας έκθεση στον ήλιο (μJ/s)*/
88        #define CHARGE_CLOUDY_DAY        150 /*Φόρτιση σε μια συννεφιασμένη μέρα (μJ/s)*/
89        /*Εσωτερικό χώρο*/
90        #define CHARGE_OFFICE_DESK         6 /*Φόρτιση πάνω σε ένα γραφείο(μJ/s)*/
91        #define CHARGE_DESK_LAMP         570 /*Φόρτιση από μια λάμπα γραφείου (μJ/s)*/
92        #define ENABLE_CHARGING            0 /*Flag που ενεργοποιεί(1) ή απενεργοποιεί(0) το μοντέλο επαναφόρτισης*/
93        /*Χρησιμεύει στο μοντέλο επαναφόρισης για να έχουμε μια καλή εκτίμηση του χρόνου που πρέπει να θεωρούμε πως φορτίζει ο κόμβος
      (είναι μέρα)
94        και για τον υπολογισμό του αντίστοιχου ποσού ενέργειας που χάνεται στο ίδιο χρονικό διάστημα*/
95        #define NUMOF_READINGS    150 /*Αριθμός μετρήσεων που πρόκειται να πραγματοποιηθούν*/
96        #define NUMOF_MINUTES      20  /*Αριθμός πραγματικού χρόνου προσομοίωσης*/
97
98
99        #define OVERHEAD    7    /*Επιπλέον κόστος σε bytes για την αποστολή κάθε πακέτου*/
100       #define HB_ALL      0    /*Flag με το οποίο ενεργοποιημένο(1) όλα τα μηνύματα θεωρούνται heart beat*/
101
102   /***********END DEFINES***********/
103
104   /************VARIABLES************/
105   /*Στη συνέχεια ορίζονται οι μεταβλητές που προσπελαύνονται καθολικά*/
106
```

```
107         Buffer* Data; /*Δείκτης για να χειριζόμαστε τη δεσμευμένη μνήμη σε τύπο Buffer για επεξεργασία στους communication
    κόμβους και στη sink*/
108         Handle hand;   /*Για δέσμευση της παραπάνω μνήμης με MemAlloc.allocate()*/
109
110         Extr_Data* History; /*Δείκτης για να χειριζόμαστε τη δεσμευμένη μνήμη σε τύπο Extr_Data για τους sense κόμβους*/
111         Handle HED;          /*Για δέσμευση της παραπάνω μνήμης με MemAlloc.allocate()*/
112
113
114         NodeInfo myNode; /*Struct με πληροφορίες για κάθε κόμβο*/
115         uint8_t BufCnt;  /*Μετρητής*/
116         char filename[30];
117         int clk_ticks; /*Μετρητής των ticks του ρολογιού*/
118
119         TOS_Msg ToSend;     /*Struct για την αποστολή μηνυμάτων*/
120         TOS_MsgPtr ptrRec; /*Δείκτης σε struct για την αποστολή μηνυμάτων*/
121         /*Κάθε κόμβος ανοίγει ή δημιουργεί αρχείο που αντιστοιχεί σε αυτόν*/
122         FILE *data_fp;      /*Δείκτης σε αρχείο για ανάγνωση δεδομένων*/
123         FILE *rec_data_fp; /*Δείκτης σε αρχείο για αποθήκευση των αποτελεσμάτων*/
124         FILE *energy_fp;    /*Δείκτης σε αρχείο για αποθήκευση της κατανάλωσης ενέργειας*/
125         FILE *stats_fp;     /*Δείκτης σε αρχείο για αποθήκευση στατιστικών που αφορούν την αποστολή και λήψη μηνυμάτων*/
126
127
128     /*Λογικά flags για την επίτευξη συγχρονισμού και την αποφυγή συνθηκών ανταγωνισμού (με TRUE είναι ενεργοποιημένα, με FALSE
    απενεργοποιημένα)*/
129         uint8_t senseSend;          /*Καθορίζει πως είναι δυνατόν να πραγματοποιηθεί μια μέτρηση αφού έχει ολοκληρωθεί η
    διαδικασία για την προηγούμενη*/
130         uint8_t commReceiveMode; /*Είμαι ενδιάμεσος κόμβος και μπορώ να λάβω μήνυμα(δεν επεξεργάζομαι κάποιο
    προηγούμενο)*/
131         uint8_t sinkReceiveMode; /*Είμαι η δεξαμενή και μπορώ να λάβω μήνυμα(δεν επεξεργάζομαι κάποιο προηγούμενο)*/
132         uint8_t commFirstRec;      /*Είμαι ενδιάμεσος κόμβος και  έχω λάβει τουλάχιστον ένα μήνυμα*/
133         uint8_t sinkFirstRec;      /*Είμαι η δεξαμενή και  έχω λάβει τουλάχιστον ένα μήνυμα*/
134         uint8_t extr_start;         /*Έχω λάβει κάποια αρχικά μηνύματα(ανάλογα με το βαθμό του extrapolation) και μπορώ να
    ξεκινήσω να χρησιμοποιώ την utility function*/
135         uint8_t messageReady;      /*Το μήνυμα είναι έτοιμο για προώθηση*/
136
137         uint8_t FORWARD_MSG;     /*Είναι TRUE αν πρόκειται να προωθήσουμε μήνυμα*/
138         uint8_t HBM_REC;          /*Είναι TRUE αν το μήνυμα που θα προωθήσουμε ή που λάβαμε είναι Heart Beat*/
139
140         uint8_t init_extr_data;   /*Μετρητής των πρώτων τιμών που εισάγονται στον πίνακα με τις τιμές για extrapolation,
141                                     χρησιμεύει ώστε να έχουμε είσαγει στα πρώτα βήματα ενεργοποίησης του πρωτοκόλλου όσες
142                                     τιμές χρειάζεται για να κάνουμε extrapolation*/
143         uint8_t messageLength;    /*Κρατάμε το μέγεθος του νηνύματος που πρόκειται να στείλουμε*/
144
145         //uint8_t just_sent; /*Μεταβλητές που χρησιμοποιούνται για debug*/
146         //double lastE;
147
148         double IdlePC; /*Ποσοστό του step που είναι idle ο επεξεργαστής*/
149         double SBPC;    /*Ποσοστό του step που είναι stand by ο επεξεργαστής*/
150
151         uint8_t COMM_REC_DONE; /* Παίρνει τιμή COMM_REC_DONE_1 ή COMM_REC_DONE_2 ανάλογα με το βάθος που
    βρίσκεται στο δέντρο*/
152         uint8_t COMM_SEND;       /*Παίρνει τιμή COMM_SEND_1 ή COMM_SEND_2 ανάλογα με το βάθος που βρίσκεται στο
    δέντρο*/
153
154     /*Μεταβλητές που χρησιμοποιούνται για τον καθορισμό των χαρακτηριστικών της utility function*/
155         float ERR_THRESHOLD; /*Το error threshold*/
156         float UT_THRESHOLD;   /*Το threshold για τη utility function*/
```

```
157         float W1,W2,W3;         /*Τα βάρη της utility function*/
158         double E_MAX,CUR_E;     /*Η μεταβλητή E_MAX καθορίζει τη μέγιστη ενέργεια που έχει ένας κόμβος και η CUR_E η ενέργεια
        που έχουμε κάθε στιγμή*/
159
160         Statistics stats; /*Struct που χρησιμοποιείται για τη συλλογή στατιστικών στοιχείων*/
161
162
163     /*********END VARIABLES**********/
164
165
166     /********LOCAL FUNCTIONS********/
167
168
169     /*Η παρακάτω συνάρτηση εκτελείται μια φορά κατα την αρχικοποίηση κάθε κόμβου και καθορίζει τα χαρακτηριστικά του.
170     Η τοπολογία που προκύπτει μετά από την εκτέλεση αυτής της συνάρτησης είναι ένα πλήρες δυαδικό δέντρο και κάθε κόμβος
171     ανάλογα με το ID του λαμβάνει και τη θέση του στο δέντρο. Ο κόμβος 0 είναι η δεξαμενή και ανάλογα με το πλήθος των κόμβων
172     που προσομοιώνονται έχουμε και διαφορετικό πλήθος επιπέδων στο δέντρο. Ο αλγόριθμος αρχικοποίησης έχει σχεδιαστεί
173     ώστε να δημιουργεί πλήρες δυαδικό δέντρο και γι'αυτό κατά την εκτέλεση πρέπει οι χρήστες να δίνουν αριθμό της μορφής
174     (2^κ - 1) με κ=1,2,3,.. για να προκύπτει τέτοιο δέντρο. Η αρχικοποίηση αυτή χρησιμεύει ώστε κάθε κομβος να ξέρει το ρόλο
175     του σε αυτή τη δομή (που πρέπει να στείλει, πόσες τιμές πρέπει να αποθηκεύσει για extrapolation κ.ά). Εφόσον, θέλουμε να
176     έχουμε άλλη τοπολογία αρκεί αυτή η συνάρτηση να τροποποιηθεί κατάλληλα ώστε να παρέχονται οι πληροφορίες που αποθηκεύουμε
177     στην  struct myNode τύπου NodeInfo.*/
178         void initialize_tree()
179         {
180             uint8_t FirstLeaf;
181             uint8_t Depth,i;
182             myNode.NumOfNodes=tos_state.num_nodes;
183             myNode.myID=NODE_NUM;
184             FirstLeaf=(myNode.NumOfNodes-1)/2;
185             if(myNode.myID==0)/*sink κόμβος*/
186             {
187                 myNode.myType=SINK_NODE;/*Καθορισμός τύπου*/
188                 myNode.myChildren=2;     /*Αριθμός παιδιών*/
189                 Depth=((int)(log10(myNode.NumOfNodes+1)/log10(2)))-1;
190                 myNode.myDepth=(int)(log10(myNode.myID+1)/log10(2));
191                 myNode.myLeafs=pow(2,Depth-myNode.myDepth); /*Υπλογισμός πλήθους φύλλων με τα οποία
        συνδέεται ο κάθο κόμβοςι*/
192                 myNode.myFirstLeaf=FirstLeaf;/*Το πιο αριστερό φύλλο με το οποίο συνδέομαι*/
193             }
194             else if(myNode.myID > 0 && myNode.myID<FirstLeaf)/*Communication κόμβος*/
195             {
196                 myNode.myType=COMMUNICATION_NODE;
197                 myNode.myChildren=2;
198                 Depth=((int)(log10(myNode.NumOfNodes+1)/log10(2)))-1;
199                 myNode.myDepth=(int)(log10(myNode.myID+1)/log10(2));
200                 myNode.myLeafs=pow(2,Depth-myNode.myDepth);
201                 myNode.myFirstLeaf=myNode.myID;
202                 for(i=0;i<(Depth-myNode.myDepth);i++)
203                     myNode.myFirstLeaf=2*myNode.myFirstLeaf+1;
204             }
205             else{/*Sense κόμβος*/
206                 myNode.myType=SENSE_NODE;
207                 myNode.myChildren=0;
208                 myNode.myDepth=(int)(log10(myNode.myID+1)/log10(2));
209                 myNode.myLeafs=0;
210                 myNode.myFirstLeaf=0;
211             }
```

```
212
213             /*Καθορισμός του κόμβου πατέρα(εκεί που στέλνουμε) για τους ενδιάμεσους και τους αισθητήριους κόμβους*/
214             if(myNode.myID!=0)
215             {
216                 if(myNode.myID%2==0)
217                 {
218                     myNode.myParent=(myNode.myID/2)-1;
219                 }
220                 else
221                 {
222                     myNode.myParent=(myNode.myID-1)/2;
223                 }
224             }
225             else/*Τυπικά, τιμή μηδέν για τον κόμβο δεξαμενή*/
226             {
227                 myNode.myParent=0;
228             }
229         }
230
231     /*Βοηθητική συνάρτηση που τυπώνει τα χαρακτηριστικά κάθε κόμβου(υπάρχει για λόγους debug)*/
232         void print_node()
233         {
234             printf("I am node %d(myType=%d),i have %d childrean, my parent is %d, i have
        %d leafs and my depth is %d\n",myNode.myID,myNode.myType,myNode.myChildren,myNode.
        myParent,myNode.myLeafs,myNode.myDepth);
235         }
236
237     /*Η παρακάτω συνάρτηση χρησιμοποιείται για να ενημερώνει την καθολική μεταβλητή CUR_E με την ενέργεια
238     που διαθέτει ο κόμβος κάθε στιγμή μετά από την εκτέλεση πράξεων που καταναλώνουν ενέργεια. Δέχεται δύο
239     ορίσματα έναν int τον amount που μας βοηθά να προσδιορίζουμε την ποσότητα της ενέργειας που καταναλώθηκε
240     και ένας uint8_t τον type που ανάλογα με την τιμή που έχει (T_CHARGE, T_IDLE, T_SEND, T_RECEIVE, T_INSTR)
241     γνωρίζουμε την αιτία που καταναλώνει την ενέργεια και αντίστοιχα αφαιρούμε την κατάλληλη ποσότητα. */
242         void update_energy(int amount,uint8_t type)
243         {
244             double difference,time_factor;
245             double recharging_ratio;
246             double randnum;
247             double elapsed_hours;
248
249
250             if (type==T_INSTR)
251                 difference=((double)amount*(double)EPI)/1000;/*Το 1000 στη διαίρεση χρησιμοποιείται για την
        αναγωγή στις σωστές μονάδες*/
252             else if (type==T_RECEIVE)
253                 difference=((double)(amount+OVERHEAD)*(double)EPBR*8)/1000;/*Το 8 μετατρέπει να
        bytes που δεχόμαστε στην είσοδο σε bits*/
254             else if (type==T_SEND)
255                 difference=((double)(amount+OVERHEAD)*(double)EPBS*8)/1000;
256             else if (type==T_IDLE)
257             {
258                 /*Με βάση τον παράγοντα κατανάλωσης και τη διάρκεια που το CPU είναι idle ή stand by υπολογίζουμε την κατανάλωση*/
259                 time_factor=(double)(60000*NUMOF_MINUTES)/(double)(NUMOF_READINGS*STEP*
        CLICK);
260                 difference=((double)EPMC+(time_factor*((double)amount/1000))*(SBPC*EPSSB+
        IdlePC*EPSI))/1000;
261             }
262             else if (type==T_CHARGE)
```

```
263             {
264                 randnum=(double)(call MyRand.rand())/65535;
265                 time_factor=(double)(60000*NUMOF_MINUTES)/(double)(NUMOF_READINGS*STEP*
    CLICK);
266                 elapsed_hours=(time_factor*clk_ticks*(CLICK/1000))/3600;/*Ώρες που πέρασαν από την
    αρχή της προσομοίωσης*/
267                 recharging_ratio=0;
268                 /*Θεωρούμε πως 10 ώρες είναι μέρα και μπορούμε να έχουμε φόρτιση*/
269                 if (((long)elapsed_hours%24)<10)
270                 {
271                     if (randnum<0.25)
272                         recharging_ratio=CHARGE_DIRECT_SUN;
273                     else if (randnum<0.95)
274                         recharging_ratio=CHARGE_CLOUDY_DAY;
275                     else
276                         recharging_ratio=0;
277                 }
278                 difference=-(time_factor*((double)amount/1000)*recharging_ratio);
279             }
280             else
281             {
282                 dbg(DBG_USR3, "Wrong type of energy consumption.\n");
283             }
284     /*Με χρήση του atomic είμαστε βέβαιοι πως όσο εκτελείται η ενημέρωση της ενέργειας δε θα συμβεί άλλος υπολογισμός όσο γίνεται αυτή η
    ενημέρωση*/
285             atomic CUR_E=CUR_E-difference;
286
287             if (CUR_E>E_MAX)
288                 CUR_E=E_MAX;
289         }
290
291     /*Η παρακάτω συνάρτηση υλοποιεί την αντιγραφεί size από το src στο dest*/
292         int memorycpy(int8_t* dest, int8_t* src, int16_t size)
293         {
294             int16_t cnt = 0;
295             update_energy((int)(3*size),T_INSTR);
296             do
297             {
298                 dest[cnt] = src[cnt];
299                 cnt++;
300             } while (cnt<size);
301             return cnt;
302         }
303
304     /*Η συνάρτηση υπολογίζει τη γραμμική παρεμβολή (ανάλογα με το σχήμα που χρησιμοποιείται για προσέγγιση
305     η συνάρτηση αυτή μπορεί να τροποποιηθεί)*/
306         double extrap_val(Extr_Data data)
307         {
308             double result;
309             result = 2 * data.values[0] - data.values[1];
310             update_energy((int)(2*EXTR_DEGREE),T_INSTR);
311             return result;
312         }
313
314     /*Η παρακάτω συνάρτηση υπολογίζει κάθε component της Utility function και ανάλογα επιστρέφει TRUE ή FALSE
315     επηρεάζοντας το επόμενο βήμα μας. Το err που δέχεται ως όρισμα χρησιμοποιείται για να συγκριθεί με το κατώφλι
316     σφάλματος του error component.*/
```

```
317         uint8_t eval_utility(float err)
318         {
319             float Uk,Uerr=0,Uen=0,Utime=0;
320             float DT,Dt;
321             update_energy((int)26,T_INSTR);
322 
323             /*error component*/
324             if (err<ERR_THRESHOLD)
325             {
326                 Uerr=(err*err)/(ERR_THRESHOLD*ERR_THRESHOLD);
327             }
328             else
329             {
330                 Uk=1.0;/*Αν το error είναι μεγάλο στέλνουμε*/
331                 return (Uk>=UT_THRESHOLD);
332             }
333 
334             /*energy component*/
335             atomic Uen=1-(float)exp((double)(-10*((double)CUR_E/(double)E_MAX)));
336 
337             /*time component*/
338             DT=(float)(STEP*HEART_BEAT_FREQ);
339             Dt=(float)(clk_ticks%(int)DT);
340             if (Dt<(DT/2))
341                 Utime=2*(Dt/DT);
342             else
343                 Utime=-2*(Dt/DT)+2;
344 
345             /*Συνολικός υπολογισμός utility*/
346             Uk=W1*Uen+W2*Uerr+W3*Utime;
347 
348             return (Uk>=UT_THRESHOLD);
349         }
350 
351     /*Η παρακάτω συνάρτηση χρησιμοποιείται ενημέρωση των τιμών που κρατάμε για extrapolation σε κάθε βήμα
352     σε αισθητήριους κόμβους*/
353         void shift_insert_sense(float newval)
354         {
355             uint8_t i;
356             for (i=EXTR_DEGREE-1;i>0;i--)
357                 History->values[i]=History->values[i-1];
358             History->values[i]=newval;
359             update_energy((int)(2*EXTR_DEGREE+1),T_INSTR);
360         }
361 
362     /*Η παρακάτω συνάρτηση χρησιμοποιείται ενημέρωση των τιμών που κρατάμε για extrapolation σε κάθε βήμα
363     σε ενδιάμεσους κόμβους και στη δεξαμενή*/
364         void shift_insert(uint8_t pos,float newval)
365         {
366             uint8_t i;
367             for (i=EXTR_DEGREE-1;i>0;i--)
368                 Data[pos].old_data.values[i]=Data[pos].old_data.values[i-1];
369             Data[pos].old_data.values[i]=newval;
370             update_energy((int)(2*EXTR_DEGREE+1),T_INSTR);
371         }
372 
373     /*****END LOCAL FUNCTIONS******/
```

```
374
375
376
377
378     /*Στη συνέχεια υλοποιούνται οι commands του interface που παρέχει το Mcmfa (εδώ StdControl).*/
379
380     /*Υλοποίηση της init του interface StdControl*/
381         command result_t StdControl.init()
382         {
383
384             call MyRand.init();/*Ενεργοποίηση για το interface Random*/
385
386             initialize_tree();/*Αρχικοποίηση της τοπολογίας*/
387             /*print_node();*/
388
389         /*Αρχικοποίηση σταθερών*/
390             stats.msgs_rec=0;
391             stats.msgs_sent=0;
392             stats.msgs_not_sent=0;
393             stats.bits_rec=0;
394             stats.bits_sent=0;
395
396             clk_ticks=-1;
397             BufCnt=0;
398             init_extr_data=0;
399
400             extr_start=FALSE;
401             sinkFirstRec=FALSE;
402             commFirstRec=FALSE;
403             FORWARD_MSG=FALSE;
404
405         /*Αρχικοποίηση σφαλμάτων, βαρών και ενέργειας σε κάυε κόμβο*/
406             ERR_THRESHOLD=0.1;
407             UT_THRESHOLD=0.85;
408             W1=0.35;
409             W2=0.35;
410             W3=0.30;
411             E_MAX=1.5e+8;/*Η αρχική ενέργεια είναι μετρημένη σε μJ (η τιμή μπορεί να φτάσει μέχρι και 23μJ)*/
412
413             CUR_E=E_MAX;/*Αρχικά, η ενέργεια είναι ίση με τη μέγιστη*/
414
415             HBM_REC=FALSE;
416             /*Ανάλογα με το βάθος κάθε κόμβος επιλέγει το παράθυρο στο οποίο θα λειτουργήσει ώστε να έχουμε τον κατάλληλο συγχρονισμό*/
417             COMM_REC_DONE=(myNode.myDepth%2)?COMM_REC_DONE_2:COMM_REC_DONE_1;
418             COMM_SEND=(myNode.myDepth%2)?COMM_SEND_2:COMM_SEND_1;
419
420             //lastE=E_MAX;
421             /*Ποσοστά λειτουργίας σε IDLE και Stand by κατάσταση του κόμβου, ανάλογα με τον τύπο του.*/
422             if (myNode.myType==SENSE_NODE)
423             {
424                 IdlePC=0.2/10.0;
425                 SBPC=9.3/10.0;
426             }
427             else
428             {
429                 IdlePC=0.5/10.0;
430                 SBPC=9.0/10.0;
```

```
431             }
432
433
434             /*Άνοιγμα αρχείου για ανάγνωση μετρήσεων*/
435             if(myNode.myType==SENSE_NODE)
436             {
437                 sprintf(filename,"data%d.txt",myNode.myID);
438                 if((data_fp=fopen(filename,"r"))==NULL)
439                 {
440                     dbg(DBG_USR3, "Error opening data file.\n");
441                 }
442             }
443             /*Δημιουργία αρχείου όπου σώζει η δεξαμενή τα τελικά αποτελέσματα*/
444             if(myNode.myType==SINK_NODE)
445             {
446                 if((rec_data_fp=fopen("recdata.txt","w"))==NULL)
447                 {
448                     dbg(DBG_USR3, "Error creating results file.\n");
449                 }
450             }
451             /*Δημιουργία αρχείου όπου κάθε κόμβος σώζει στοιχεία για την κατανάλωση ενέργειας του*/
452             sprintf(filename,"energy%d.txt",myNode.myID);
453             if((energy_fp=fopen(filename,"w"))==NULL)
454             {
455                 dbg(DBG_USR3, "Error creating energy file.\n");
456             }
457             /*Δημιουργία αρχείου όπου κάθε κόμβος αποθηκεύει στατιστικά στοιχεία για της αποστολές και τις λήψεις του*/
458             sprintf(filename,"stats%d.txt",myNode.myID);
459             if((stats_fp=fopen(filename,"w"))==NULL)
460             {
461                 dbg(DBG_USR3, "Error creating statistics file.\n");
462             }
463
464             switch(myNode.myType)
465             {
466                 case COMMUNICATION_NODE:
467                     commReceiveMode=TRUE;
468                     messageReady=FALSE;
469                     /*Για τους ενδιάμεσους κόμβους δεσμεύουμε μνήμη ίση με myNode.myLeafs*sizeof(Buffer). Χρειαζόμαστε για κάθε
    φύλλο
470                     που επικοινωνεί με τον κόμβο μας μια struct τύπου Buffer για να διατηρούμε τις τιμές που χρειάζονται για
    extrapolation*/
471                     if (call MemAlloc.allocate(&hand,myNode.myLeafs*sizeof(Buffer)) ==
    FAIL)
472                     {
473                         dbg(DBG_USR3, "Error allocating memory for communication node.\n");
474                     }
475                     break;
476                 case SENSE_NODE:
477                     senseSend=TRUE;
478                     /*Δέσμευση μνήμης για EXTR_DEGREE τιμές*/
479                     if (call MemAlloc.allocate(&HED,sizeof(Extr_Data))==FAIL)
480                     {
481                         dbg(DBG_USR3, "Error allocating memory for sense node.\n");
482                     }
483                     break;
484                 case SINK_NODE:
```

```
485                     sinkReceiveMode=TRUE;
486                     /*Για τον κόμβο δεξαμενή δεσμεύουμε χώρο για όλα τα φύλλα του δέντρου, ώστε να μπορούμε να αναπαράγουμε την τιμή
487                     που θα έπρεπε να έχουμε αν η ροή από ένα αισθητήρα δε φτάσει μέχρι τη δεξαμενή.*/
488                     if (call MemAlloc.allocate(&hand,myNode.myLeafs*sizeof(Buffer)) == FAIL)
489                     {
490                         dbg(DBG_USR3, "Error allocating memory for sink node.\n");
491                     }
492                     break;
493                 default:
494                         dbg(DBG_USR3, "Wrong type of node.\n");
495                     break;
496             }
497             return SUCCESS;
498         }
499
500     /*Υλοποίηση της start του interface StdControl*/
501         command result_t StdControl.start()
502         {/*Εκκίνηση χρονομέτρη που προκαλεί event ανά CLICK*/
503             call Timer.start(TIMER_REPEAT,CLICK);
504             return SUCCESS;
505         }
506
507     /*Υλοποίηση της stop του interface StdControl*/
508         command result_t StdControl.stop()
509         {
510             return call Timer.stop();
511         }
512
513
514     /*Στη συνέχεια υλοποιούνται τα events των interfaces που χρησιμοποιεί το Mcmfa.*/
515
516     /*Το event αυτό σηματοδοτείται όταν ο χρονομέτρης ενεργοποιηθεί*/
517         event result_t Timer.fired()
518         {
519             float newval;
520             double extrapval;
521             uint8_t HBmessage=FALSE;
522             uint8_t bufmax,i;
523             uint8_t position;
524             int8_t count=1;
525
526
527             clk_ticks++;/*Αύξηση του καθολικού αθροιστή των ticks του ρολογιού*/
528             /*if (myNode.myID==0)
529             {
530                 printf("Step %d, Tick %d\n",(int)clk_ticks/STEP,clk_ticks);
531             }*/
532
533             ///changed by manos start
534             //if (clk_ticks%STEP==0)
535             //    just_sent=FALSE;
536             /*Ανά 15 step αποθηκεύουμε στο κατάλληλο αρχείο στοιχεία για την υπολοιπόμενη ενέργεια*/
537             if (clk_ticks%STEP==15)
538             {
539                 //printf("\tEnergy Report for %d: Current energy: %.4lf μJ Msg sent:%d
```

```
Diff:%.4lf\n",myNode.myID,CUR_E,just_sent,lastE-CUR_E);
540                 fprintf(energy_fp,"%lf\n",CUR_E);
541                 //lastE=CUR_E;
542             }
543             /*Αποθήκευση στο κατάλληλο αρχείο στατιστικά και τις αποστολές και τις λήψεις*/
544             if (clk_ticks%(5*STEP)==1)
545             {
546                 //printf("Messages for %d: received: %d, sent: %d, not sent (due to utility)
%d\n",myNode.myID,stats.msgs_rec,stats.msgs_sent,stats.msgs_not_sent);
547                 fprintf(stats_fp,"%d %d %d %d %d\n",stats.msgs_rec,stats.msgs_sent,stats.
msgs_not_sent,stats.bits_rec,stats.bits_sent);
548             }
549
550             if (CUR_E<50)
551             {/*Διακόπτεται η προσομοίωση αν μείνει από ενέργεια ο κόμβος*/
552                 call Timer.stop();
553                 dbg(DBG_USR3, "Simulation stopped. Out of energy\n");
554             }
555
556             if (myNode.myType==SENSE_NODE)
557             {/*Για τους αισθητήριους κόμβους*/
558                 update_energy((int)2,T_INSTR);
559                 if (clk_ticks%STEP==SENSE_SEND && senseSend)
560                 {/*Αν είμαστε στο κατάλληλο χρονικό παράθυρο και δεν υπάρχει σε εξέλιξη άλλη αποστολή ξεκινάμε τη διαδικασία
μέτρησης*/
561                     atomic senseSend=FALSE;
562                     fscanf(data_fp,"%f\n",&(newval));
563                     update_energy((int)5,T_INSTR);
564                     /*Έλεγχος αν το μήνυμα είναι Heart Beat*/
565                     if (((((int)clk_ticks/STEP)%HEART_BEAT_FREQ)>=0 && (((int)clk_ticks/
STEP)%HEART_BEAT_FREQ)<EXTR_DEGREE) || init_extr_data<EXTR_DEGREE || HB_ALL)
566                     {
567                         HBmessage=TRUE;
568                     }
569                     if (extr_start)
570                     {/*Αν μπορούμε να κάνουμε extrapolation*/
571                         update_energy((int)8,T_INSTR);
572                         if (!HBmessage)
573                         {/*αν το μήνυμα δεν είναι Heart Beat πραγματοποιούμε extrapolation*/
574                             extrapval=extrap_val(*History);
575
//printf("ID=%d,extr=%f,new=%f,util=%d\n",myNode.myID,extrapval,newval,eval_utility(fabs(extrapval-newval)));
576                         }
577                         if (HBmessage)
578                         {/*Αποστολή Heart Beat μηνύματος*/
579                             count=(-count);/*Σύμβαση για να αναγνωρίζουμε στα παραπάνω επίπεδα τα Heart Beat
μηνύματα*/
580                             /*Δημιουργία κατάλληλου μηνύματος*/
581                             memorycpy((uint8_t*)&(ToSend.data),(uint8_t*)&(count),sizeof(
int8_t));
582                             memorycpy((uint8_t*)&(ToSend.data[sizeof(uint8_t)]),(uint8_t
*)&(myNode.myID),sizeof(uint8_t));
583                             memorycpy((uint8_t*)&(ToSend.data[2*sizeof(uint8_t)]),((
uint8_t*)&newval),sizeof(float));
584                             shift_insert_sense(newval);/*Ανανέωση των τιμών που κρατάμε για extrapolation*/
585                         }
586                         else if (eval_utility(fabs((extrapval-newval)/newval)))
```

```
587                         {/*Αν δεν έχουμε Heart Beat μήνυμα η utility function αποφασίζει αν θα στείλουμε ή όχι*/
588                             memorycpy((uint8_t*)&(ToSend.data),(uint8_t*)&(count),sizeof(
    int8_t));
589                             memorycpy((uint8_t*)&(ToSend.data[sizeof(uint8_t)]),(uint8_t
    *)&(myNode.myID),sizeof(uint8_t));
590                             memorycpy((uint8_t*)&(ToSend.data[2*sizeof(uint8_t)]),((
    uint8_t*)&newval),sizeof(float));
591                             shift_insert_sense(newval);/*Ανανέωση των τιμών που κρατάμε για extrapolation*/
592                         }
593                         else
594                         {/*Δεν χρειάζεται να στείλουμε λόγω utility*/
595                             shift_insert_sense(extrapval);
596                             atomic senseSend=TRUE;
597                         }
598                     }
599                     else/*διαφορετικά στέλνουμε για να έχουμε EXTR_DEGREE τιμές για extrapolation*/
600                     {
601                         update_energy((int)4,T_INSTR);
602                         init_extr_data++;
603                         count=(-count);
604                         memorycpy((uint8_t*)&(ToSend.data),(uint8_t*)&(count),sizeof(
    int8_t));
605                         memorycpy((uint8_t*)&(ToSend.data[sizeof(uint8_t)]),(uint8_t*)&(
    myNode.myID),sizeof(uint8_t));
606                         memorycpy((uint8_t*)&(ToSend.data[2*sizeof(uint8_t)]),((uint8_t*)&
    newval),sizeof(float));
607                         shift_insert_sense(newval);
608                         if (init_extr_data==EXTR_DEGREE)
609                             atomic extr_start=TRUE;
610                     }
611                     if (!senseSend)
612                     {/**/
613                         update_energy((int)2,T_INSTR);
614                         //just_sent=TRUE;
615                         /*Αποστολή του μηνύματος*/
616                         if(call MySend.send(myNode.myParent,2*sizeof(uint8_t)+sizeof(float
    ),&ToSend)==FAIL)
617                         {
618                             dbg(DBG_USR3, "Error calling MySend.send().\n");
619                         }
620                     }
621                     else
622                         stats.msgs_not_sent++;
623 
624                     atomic senseSend=TRUE; /*ενεργοποίηση του flag*/
625                     update_energy((int)2,T_INSTR);
626                 }
627             }
628             else if (myNode.myType==COMMUNICATION_NODE && commFirstRec)
629             {/*αν έχουμε communication κόμβο*/
630                 atomic bufmax=BufCnt;
631                 update_energy((int)2,T_INSTR);
632                 if (clk_ticks%STEP==COMM_REC_DONE)
633                 {/*αν έχουμε το κατάλληλο παράθυρο*/
634                     atomic commReceiveMode=FALSE;
635                     update_energy((int)4,T_INSTR);
636                     if (bufmax<myNode.myLeafs)
```

```
637                     {/*Δεν έχω λάβει όλες τις τιμές και όσες δεν έλαβα τις προσεγγίζω με extrapolation*/
638                         for (i=0;i<myNode.myLeafs;i++)
639                         {
640                             if (Data[i].valid==FALSE)
641                             {
642                                 Data[i].valid=TRUE;
643                                 Data[i].myValue=extrap_val(Data[i].old_data);
644                             }
645                         }
646                         update_energy((int)(myNode.myLeafs*2),T_INSTR);
647                     }
648 
649                     if (HBM_REC)
650                     {/*Αν έχω λάβει Heart Beat μήνυμα το προωθώ ως Heart Beat στον επόμενο κόμβο*/
651                         count=-myNode.myLeafs;
652                         memorycpy((uint8_t*)&(ToSend.data),(uint8_t*)&(count),sizeof(
    int8_t));
653                         position=sizeof(int8_t);
654                         /*Δημιουργία μηνύματος*/
655                         for (i=0;i<myNode.myLeafs;i++)
656                         {
657                             memorycpy((uint8_t*)&(ToSend.data[position]),(uint8_t*)&(Data[
    i].myID),sizeof(uint8_t));
658                             position+=sizeof(uint8_t);
659                             memorycpy((uint8_t*)&(ToSend.data[position]),(uint8_t*)&(Data[
    i].myValue),sizeof(float));
660                             position+=sizeof(float);
661                         }
662                         messageLength=position;
663                         atomic messageReady=TRUE;
664                         update_energy((int)(5+myNode.myLeafs*2),T_INSTR);
665                     }
666                     else
667                     {/*Αν δεν έχω Heart Beat μήνυμα με χρήση της utility function αποφασίζω ποιες τιμές θα στείλω και ποιες
    έρχομαι*/
668                         count=0;
669                         position=sizeof(int8_t);
670                         for (i=0;i<myNode.myLeafs;i++)
671                         {
672                             extrapval=extrap_val(Data[i].old_data);
673                             if (eval_utility(fabs((extrapval-Data[i].myValue)/Data[i].
    myValue)))
674                             {
675                                 count++;
676                                 memorycpy((uint8_t*)&(ToSend.data[position]),(uint8_t*)&(
    Data[i].myID),sizeof(uint8_t));
677                                 position+=sizeof(uint8_t);
678                                 memorycpy((uint8_t*)&(ToSend.data[position]),(uint8_t*)&(
    Data[i].myValue),sizeof(float));
679                                 position+=sizeof(float);
680                             }
681                         }
682                         memorycpy((uint8_t*)&(ToSend.data),(uint8_t*)&(count),sizeof(
    int8_t));
683                         messageLength=position;
684                         atomic messageReady=TRUE;
685                         update_energy((int)(6+myNode.myLeafs*3),T_INSTR);
```

```
686                     }
687                     /*Ενημέρωση των τιμών που θα χρησιμοποιηθούν στη συνέχεια για extrapolation*/
688                     for (i=0;i<myNode.myLeafs;i++)
689                     {
690                         shift_insert(i,Data[i].myValue);
691                     }
692                     if (count!=0)
693                         atomic FORWARD_MSG=TRUE; /*Θα στείλω μήνυμα*/
694                     else
695                     {
696                         for (i=0;i<myNode.myLeafs;i++)
697                             Data[i].valid=FALSE;
698                         atomic HBM_REC=FALSE;
699                         atomic messageReady=FALSE;
700                         atomic FORWARD_MSG=FALSE;
701                         update_energy((int)(3+myNode.myLeafs),T_INSTR);
702                         stats.msgs_not_sent++;
703                     }
704                     atomic commReceiveMode=TRUE;
705                     atomic BufCnt=0;
706                     update_energy((int)2,T_INSTR);
707                 }
708                 else if (clk_ticks%STEP==COMM_SEND && messageReady && FORWARD_MSG)
709                 {/*Αν είναι το παράθυρο για να στείλω μήνυμα και το μήνυμα είναι έτοιμο (FORWARD_MSG=TRUE)*/
710                     update_energy((int)4,T_INSTR);
711                     //just_sent=TRUE;
712                     if(call MySend.send(myNode.myParent,messageLength,&ToSend))
713                     {/*Αποστολή και ενημέρωση των flags*/
714                         for (i=0;i<myNode.myLeafs;i++)
715                             Data[i].valid=FALSE;
716                         atomic HBM_REC=FALSE;
717                         atomic messageReady=FALSE;
718                         atomic FORWARD_MSG=FALSE;
719                         update_energy((int)(myNode.myLeafs+3),T_INSTR);
720                     }
721                     else
722                     {
723                         dbg(DBG_USR3, "Error allocating memory for communication node.\n");
724                     }
725                 }
726             }
727             else if (myNode.myType==SINK_NODE)
728             {/*Αν είμαι κόμβος δεξαμενή*/
729                 atomic bufmax=BufCnt;
730                 update_energy((int)2,T_INSTR);
731                 if (clk_ticks%STEP==SINK_REC_DONE && sinkFirstRec)
732                 {/*Αν έχει φτάσει το παράθυρο για να λάβω και έχω λάβει*/
733                     atomic sinkReceiveMode=FALSE;
734                     update_energy((int)5,T_INSTR);
735                     if (bufmax<myNode.myLeafs)
736                     {/*Δεν έχω λάβει όλες τις τιμές και όσες δεν έλαβα τις προσεγγίζω με extrapolation*/
737                         for (i=0;i<myNode.myLeafs;i++)
738                         {
739                             if (Data[i].valid==FALSE)
740                             {
741                                 Data[i].valid=TRUE;
742                                 Data[i].myValue=extrap_val(Data[i].old_data);
```

```
743                             }
744                         }
745                         update_energy((int)(myNode.myLeafs*2),T_INSTR);
746                     }
747                     /*Ενημέρωση των τιμών που θα χρησιμοποιηθούν στα επόμενα βήματα για extrapolation*/
748                     for (i=0;i<myNode.myLeafs;i++)
749                     {
750                         shift_insert(i,Data[i].myValue);
751                     }
752 
753                     //printf("I am %d and I received %d values and the %d set of real and extrapolated data
        is:\n",myNode.myID,bufmax,myNode.myLeafs);
754                     for (i=0;i<myNode.myLeafs;i++)
755                     {
756                         //printf("{ID=%d,",Data[i].myID);
757                         //printf("val=%6.4f} ",Data[i].myValue);
758                         fprintf(rec_data_fp,"%d,%6.4f ",Data[i].myID,Data[i].myValue);
759                     }
760                     //printf("\n");
761                     fprintf(rec_data_fp,"\n");
762                     atomic sinkReceiveMode=TRUE;
763                     for (i=0;i<myNode.myLeafs;i++)
764                         Data[i].valid=FALSE;
765                     atomic HBM_REC=FALSE;
766                     atomic BufCnt=0;
767                     update_energy((int)(3+myNode.myLeafs),T_INSTR);
768                 }
769             }
770             /*Ενημέρωση της ενέργειας ανά tick*/
771             update_energy(CLICK,T_IDLE);
772             if (clk_ticks%STEP==1 && ENABLE_CHARGING)
773                 update_energy(CLICK*STEP,T_CHARGE);
774             return SUCCESS;
775         }
776 
777     /*Επιβεβαίωση αποστολής μηνύματος*/
778         event result_t MySend.sendDone(TOS_MsgPtr sent,result_t success)
779         {/*Ενημέρωση στατιστικών και ενέργειας*/
780             stats.msgs_sent++;
781             stats.bits_sent+=(((sent->length)+OVERHEAD)*8);
782             update_energy((int)(sent->length),T_SEND);
783 
784             return SUCCESS;
785         }
786 
787     /*Event που εκτελείται όταν λαμβάνεται μήνυμα*/
788         event TOS_MsgPtr MyReceive.receive(TOS_MsgPtr m)
789         {
790             MyMsg reading;
791             TOS_MsgPtr temp;
792             int8_t count;
793             uint8_t i,j;
794             uint8_t position;
795             temp=ptrRec;
796             ptrRec=m;
797             update_energy((int)(ptrRec->length),T_RECEIVE);
798             update_energy((int)4,T_INSTR);
```

```
799
800             stats.bits_rec+=(((ptrRec->length)+OVERHEAD)*8);
801             stats.msgs_rec++;
802
803             if (myNode.myType==COMMUNICATION_NODE && commReceiveMode)
804             {/*Λήψη μηνύματος σε ενδιάμεσο κόμβο αν το παράθυρο είναι ανοικτό για λήψη*/
805                 atomic commFirstRec=TRUE;
806                 memorycpy((uint8_t*)&(count),(uint8_t*)&(ptrRec->data),sizeof(int8_t));
807                 position=sizeof(int8_t);
808                 update_energy((int)2,T_INSTR);
809                 if (count<0)
810                 {/*Έλεγχος αν το μήνυμα είναι Heart Beat*/
811                     count=(-count);
812                     HBM_REC=TRUE;
813                 }
814                 for (i=0;i<count;i++)
815                 {/*Επεξεργασία μηνύματος*/
816                     update_energy((int)2,T_INSTR);
817                     memorycpy((uint8_t*)&(reading.myID),(uint8_t*)&(ptrRec->data[position
    ]),sizeof(uint8_t));
818                     position+=sizeof(uint8_t);
819                     memorycpy((uint8_t*)&(reading.myValue),(uint8_t*)&(ptrRec->data[
    position]),sizeof(float));
820                     position+=sizeof(float);
821                     //printf("I am %d and I received from %d values(%d):%f\n",myNode.myID,reading.myID,count,reading.myValue);
822                     update_energy((int)(myNode.myLeafs/2+3),T_INSTR);
823                     for (j=0;j<myNode.myLeafs;j++)
824                     {/*"Ταξινόμηση" τιμών για πιο εύκολη επεξεργασία*/
825                         if (Data[j].myID==reading.myID)
826                         {
827                             atomic
828                             {
829                                 Data[j].valid=TRUE;
830                                 Data[j].myValue=reading.myValue;
831                                 BufCnt++;
832                             }
833                             break;
834                         }
835                     }
836                 }
837             }
838             else if (myNode.myType==SINK_NODE && sinkReceiveMode)
839             {/*Λήψη μηνύματος σε  κόμβοδεξαμενή αν το παράθυρο είναι ανοικτό για λήψη*/
840                 atomic sinkFirstRec=TRUE;
841                 memorycpy((uint8_t*)&(count),(uint8_t*)&(ptrRec->data),sizeof(int8_t));
842                 position=sizeof(int8_t);
843                 update_energy((int)2,T_INSTR);
844                 if (count<0)
845                 {/*Έλεγχος αν το μήνυμα είναι Heart Beat*/
846                     count=(-count);
847                     HBM_REC=TRUE;
848                 }
849                 for (i=0;i<count;i++)
850                 {/*Επεξεργασία μηνύματος*/
851                     update_energy((int)2,T_INSTR);
852                     memorycpy((uint8_t*)&(reading.myID),(uint8_t*)&(ptrRec->data[position
    ]),sizeof(uint8_t));
```

```
853                     position+=sizeof(uint8_t);
854                     memorycpy((uint8_t*)&(reading.myValue),(uint8_t*)&(ptrRec->data[
    position]),sizeof(float));
855                     position+=sizeof(float);
856         //          printf("I am %d and I received from %d %d values:%f\n",myNode.myID,reading.myID,count,reading.myValue);
857                     update_energy((int)(myNode.myLeafs/2+3),T_INSTR);
858                     for (j=0;j<myNode.myLeafs;j++)
859                     {/*"Ταξινόμηση" τιμών για πιο εύκολη επεξεργασία*/
860                         if (Data[j].myID==reading.myID)
861                         {
862                             atomic
863                             {
864                                 Data[j].valid=TRUE;
865                                 Data[j].myValue=reading.myValue;
866                                 BufCnt++;
867                             }
868                             break;
869                         }
870                     }
871                 }
872             }
873             return temp;
874         }
875
876     /*Event που πυροδοτείται όταν ολοκληρωθεί η δυναμική δέδμευση μνήμης*/
877         event result_t MemAlloc.allocComplete(HandlePtr handle, result_t success)
878         {
879             uint8_t i,j;
880
881             if (myNode.myType==SENSE_NODE)
882             {/*Αρχικοποίηση*/
883                 History=(Extr_Data*)*HED; /*Ανάθεση σε δείκτη του handler για πιο εύκολο χειρισμό της δεσμευμένης
    μνήμης*/
884                 update_energy((int)(3+EXTR_DEGREE),T_INSTR);
885                 for (j=0;j<EXTR_DEGREE;j++) /*Αρχικοποίηση*/
886                     History->values[j]=0;
887             }
888             else
889             {
890                 Data=(Buffer*)*hand; /*Ανάθεση σε δείκτη του handler για πιο εύκολο χειρισμό της δεσμευμένης μνήμης*/
891                 update_energy((int)(myNode.myLeafs*(2+EXTR_DEGREE)+1),T_INSTR);
892                 for (i=0;i<myNode.myLeafs;i++)
893                 {/*Αρχικοποίηση*/
894                     Data[i].myID=myNode.myFirstLeaf+i;
895                     Data[i].valid=FALSE;
896                     for (j=0;j<EXTR_DEGREE;j++)
897                         Data[i].old_data.values[j]=0;
898                 }
899             }
900
901             return SUCCESS;
902         }
903
904     /*Τα δύο παρακάτω events από το Interface MemAlloc δεν χρησιμοποιούνται και γι' αυτό δεν υλοποιείται και ο κώδικας τους*/
905         event result_t MemAlloc.reallocComplete(Handle handle, result_t success)
906         {
907             return SUCCESS;
```

```
908         }
909
910         event result_t MemAlloc.compactComplete()
911         {
912             return SUCCESS;
913         }
914     }
915
916
917
```

```
1   /*File name: Mcmfa.h*/
2   /**
3   Πτυχιακή Εργασία - Υλοποίηση MCMFA
4   MCMFA - Multi-Criteria Message Forwarding Architecture
5   Αθανασούλης Μανούσος-Γαβριήλ    ΑΜ:018200100002
6   Αλαγιάννης Ιωάννης               ΑΜ:018200100011
7   **/
8
9   #define EXTR_DEGREE 2 /*Ο βαθμός του extrapolation είναι EXTR_DEGREE-1*/
10
11  /*Τύποι κόμβου*/
12  enum {
13      SINK_NODE=0,
14      COMMUNICATION_NODE=1,
15      SENSE_NODE=2
16  };
17
18  enum
19  {
20      Msg=10
21  };
22
23  /*Struct μηνύματος*/
24  typedef struct Sensor_Message{
25      uint8_t myID;   /*ID αισθητήρα από τον οποίο προέρχεται η μέτρηση*/
26      float myValue; /*Μέτρηση*/
27      uint8_t valid; /*Λογική τιμή που δείχνει αν είναι valid η τιμή που έχουμε στο myValue*/
28  }MyMsg;
29
30  /*Struct όπου διατηρούνται στοιχεία που αφορούν κάθε κόμβο*/
31  typedef struct node_info
32  {
33      uint8_t NumOfNodes;   /*Πλήθος κόμβων δέντρου*/
34      uint8_t myID;         /*ID αισθητήρα*/
35      uint8_t myType;       /*Τύπος κόμβου*/
36      uint8_t myParent;     /*Πατέρας κόμβου*/
37      uint8_t myChildren;   /*Πλήθος παιδιών*/
38      uint8_t myFirstLeaf; /*Αριστερότερο φύλλο από τα φύλλα που συνδέεται (άμεσα ή έμμεσα) ο κόμβος*/
39      uint8_t myLeafs;      /*Πλήθος φύλλων με τα οποία συνδέεται ένας κόμβος άμεσα ή έμμεσα*/
40      uint8_t myDepth;      /*Βάθος στο δέντρο*/
41  }NodeInfo;
42
43
44  /*Τιμές για extrapolation*/
45  typedef struct Extrapolation_Data{
46      float values[EXTR_DEGREE];
47  }Extr_Data;
48
49  /*Struct για επεξεργασία*/
50  typedef struct DATA
51  {
52      uint8_t myID;        /*ID αισθητήρα από τον οποίο προέρχεται η μέτρηση*/
53      float myValue;       /*Μέτρηση*/
54      uint8_t valid;       /*Λογική τιμή που δείχνει αν είναι valid η τιμή που έχουμε στο myValue*/
55      Extr_Data old_data; /*Struct με τις τιμές για extrapolation*/
56  }Buffer;
57
```

```
58    /*Συλλογή στατιστικών στοιχείων*/
59    typedef struct STATS
60    {
61        int msgs_rec;       /*Μηνύματα που έλαβε ο κόμβος*/
62        int msgs_sent;      /*Μηνύματα που έστειλε ο κόμβος*/
63        int msgs_not_sent; /*Μηνύματα που δεν έστειλε ο κόμβος*/
64        int bits_rec;       /*Bits που έλαβε ο κόμβος*/
65        int bits_sent;      /*Bits που έστειλε ο κόμβος*/
66    }Statistics;
67
68
69
```

Makefile

```
1   COMPONENT=Mcmfa
2   PFLAGS=-I%T/lib/Util
3   include ../Makerules
4
5
```

Αναφορές

1. M. Athanassoulis, I. Alagiannis and S. Hadjiefthymiades, “A Multi-Criteria Message Forwarding Architecture for Wireless Sensor Networks” , PCI 2005
2. I. F. Akyildiz et al., “Wireless sensor networks: a survey”, *Computer Networks*, Vol. 38, pp. 393-422, March 2002.
3. J. N. Al-Karaki and A.E. Kamal, “Routing Techniques in Wireless Sensor Networks: a survey”, *IEEE Wireless Commnications*
4. K. Akkaya and M. Younis, “A survey on Routing Protocols for Wireless Sensor Networks”
5. V. Shnayder et al., “Simulating the Power Consumption of Large Scale Sensor Network Applications”, *ACM Conference on Embedded Network Sensor Systems (SenSys), November 2004*
6. P. Levis et al. “TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications”, *ACM Conference on Embedded Network Sensor Systems (SenSys), November 2003*
7. P. Levis and N. Lee “TOSSIM: A Simulator for TinyOS Networks”, *from TinyOS Documentation*
8. A. Sridharan, M. Zuniga and B. Krishnamachari, “Integrating Environment Simulators with Network Simulators”
9. D. Gay, P. Levis, D. Culler and E. Brewer, “nesC 1.1 Language Reference Manual”
10. D. Gay et al., “The nesC Language: A Holistic Approach to Networked Embedded Systems”, *PLDI ’03*
11. www.tinyos.net
12. C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," Proceedings of ACM MobiCom '00, Boston, MA, 2000, pp. 56-67.
13. Y. Yao and J. Gehrke, “The cougar approach to in-network query processing in sensor networks”, in SIGMOD Record, September 2002.
14. R. C. Shah and J. Rabaey, “Energy Aware Routing for Low Energy Ad Hoc Sensor Networks”, IEEE Wireless Communications and Networking Conference (WCNC), March 17-21, 2002, Orlando, FL.
15. F. Zhao and L. Guibas “Wireless Sensor Networks: An Information Processing Approach”, *Elsevier, Morgan Kaufmann Publishers*
16. J. Hill, “System Architecture for Wireless Sensor Networks”, *PhD Thesis*
17. nescc.sourceforge.net
18. www.xbow.com
19. Q. Li and J. Aslam and D. Rus, “Hierarchical Power-aware Routing in Sensor Networks", In Proceedings of the DIMACS Workshop on Pervasive Networking, May, 2001.
20. J.-H. Chang and L. Tassiulas, “Maximum Lifetime Routing in Wireless Sensor Networks", Proc. Advanced Telecommunications and Information Distribution Research Program (ATIRP2000), College Park, MD, Mar. 2000.
21. C. Rahul, J. Rabaey, “Energy Aware Routing for Low Energy Ad Hoc Sensor Networks", IEEE Wireless Communications and Networking Conference (WCNC), vol.1, March 17-21, 2002, Orlando, FL, pp. 350-355.
22. S. Dulman, T. Nieberg, J. Wu, P. Havinga, “Trade-Off between Traffic Overhead and Reliability in Multipath Routing for Wireless Sensor Networks", WCNC Workshop, New Orleans, Louisiana, USA, March 2003.
23. http://www.stormwatch.com/index.asp
24. http://iqup.ifa.hawaii.edu/domeenv
25. http://sensorwebs.jpl.nasa.gov/
26. M. Vieira et al, “Survey on Wireless Sensor Network Devices”
27. Mani Srivastava, “Wireless Sensor & Actuator Networks - Tutorial”, Mobicom 2005
28. http://www.eecs.harvard.edu/~shnayder/ptossim/mica2bench/summary.html