# Περιεχόμενα

# Μέρος Ι

# Κεφάλαιο 1

# Εισαγωγή Στα Ασύρματα Δίκτυα Αισθητήρων

## 1.1 Γενικά

> "In 2010 infants will not die of SIDs, or suffocate, or drown, without an alert being sent to the parents. How will society change when your neighbors pool calls your cell phone to tell you that Johnny is drowning and you're the closest adult that could be located?
>
> In 2020 there will be no unanticipated illness. Chronic sensor implants will monitor all of the major circulator systems in the human body, and provide you with early warning of an impending flu, or save your life by catching cancer early enough that it can be completely removed surgically." , Kris Pister

Αυτές ήταν οι προβλέψεις ενός από τους δημιουργούς των ασύρματων δικτύων αισθητήρων, που πολλές φορές αναφέρονται και ως "έξυπνη σκόνη". Ο ανερχόμενος χώρος των ασύρματων δικτύων αισθητήρων ενσωματώνει αισθητήρες, υπολογιστή και επικοινωνία σε μία μόνο μικροσκοπική συσκευή. Απο μόνη της η κάθε συσκευή δεν αποτελει κατι χρήσιμο για την επιστημονική κοινότητα και κατ' επέκταση για την κοινωνία. Η σύνθεση όμως εκατοντάδων απο αυτών μέσω ενός συνόλου δικτυακών πρωτοκόλλων σχηματίζουν μία τεχνολογική θάλασσα, η οποία προσφέρει αναπάντεχες τεχνολογικές δυνατότητες. Η σπουδαιότητα και το μεγαλείο των συσκευών αυτών έγκειται στη δυνατότητα τους να μας παρέχουν ποικίλες πληροφορίες για το χώρο που έχουν απλώθει χωρίς να χρειάζονται επίβλεψη και συντήρηση. Απο μία πρώτη εικόνα μπορεί να φαίνονται ίδια με τα δίκτυα Ad hoc δίκτυα, τα οποία είναι ειδικά διαμορφωμένα δίκτυα με την πρωτοποριακή ικανότητα να οργανώνονται με απουσία σταθερής δομής. Αν και παρουσιάζουν ομοιότητες με τα κλασσικά Ad hoc δίκτυα υπάρχουν κάποιες ιδιότητες και χαρακτηριστικά που δεν τους επιτρέπουν να ταυτιστούν μαζί τους. Παρακάτω θα αναφέρουμε τις κύριες διαφορές μεταξύ δύο τέτοιων δικτύων. Όσο θα εξελίσσεται η τεχνολογία τόσο τα δίκτυα ασύρματων αισθητήρων θα

φαίνονται όλο και πιο επαναστατικά, χρήσιμα και έτοιμα να προσφέρουν τις υπηρεσίες τους στους ανθρώπους.

## 1.2 Ιστορικό

Η έρευνα άρχισε να καρποφορεί στα μέσα της δεκαετίας του 90' με τη μορφή δικτυακών κόμβων στο μέγεθος ενός γραμματοσήμου. Αποκαλούμενες συχνά "έξυπνη σκόνη" ή motes οι μικροσκοπικές αυτές συσκευές μπορούν να ενσωματώσουν αισθητήρες και να μεταφέρουν τις πληροφορίες που συλλέγουν στα άτομα που τις χρειάζονται. Η έρευνα άρχισε πριν κάποια χρόνια σε διάφορα πανεπιστήμια. Το 1998 όμως ο καθηγητής Kris Pister και η ομαδα του στο πανεπιστήμιο Berkeley της Καλιφόρνιας κατάφεραν να κατασκευάσουν μία συσκευή με ενα αισθητήρα, μία συσκευή επικοινωνίας και ενα μικρό υπολογιστή ενσωματωμένα σε μία μόνο μονάδα. Μία από τις πρώτες συσκευές φαίνεται στο παρακάτω σχήμα.

Σχήμα 1.1: Ένας πρώιμος κόμβος "έξυπνης σκόνης" φτιαγμένος απο εμπορικά διαθέσιμα υλικά. Η συσκευή συγκρίνεται με ένα κέρμα. (Πηγή Εικόνας: Seth Hollar, (Hollar 2000))

Η συσκευή αυτή ονομάζεται "RF Mote" και έχει αισθητήρες θερμοκρασίας, υγρασίας, βαρομετρικής πίεσης, φωτός, επιτάνυνσης και μαγνητισμού. Μπορεί να επικοινωνήσει από αποστάσεις περίπου 18 μέτρων με ραδιομετάδοση και αν δουλεύει συνεχώς η μπαταρία της διαρκεί περίπου μία εβδομάδα. Ένα από τα προβλήματα, που έπρεπε να αντιμετωπίσει η ομάδα του πανεπιστημίου για την κατασκευή μικρότερων συσκευών ήταν η κατανάλωση ενέργειας. Μικρότερες μπαταρίες βοηθούν στον περιορισμό του μεγέθους του της παραγόμενης συσκευής, αλλά προσφέρουν λιγότερη ενέργεια από τις παραδοσιακά μεγάλες μπαταρίες. Παρόλαυτά οι μεγάλης διάρκειας μπαταρίες είναι σημαντικές σε εφαρμογές που η συλλογή της συσκευής και η αλλαγή μπαταρίας είναι δαπανηρή, μη συμφέρουσα και πολλές φορές ανέφικτη. Για παράδειγμα για συσκευες που μετράνε τη θερμοκρασία και την υγρασία και τοποθετούνται μέσα σε τοίχους κτιρίων κατά τη διάρκεια της κατασκευής τους. Αντιμετωπίζοντας λοιπόν το δίλημμα μεταξύ μεγάλης διάρκειας και μικρού μεγέθους οι αρχικοί κατασκευαστές τέτοιων συσκευών επέλεξαν το μικρό μέγεθος. Γι'άυτό κατέβαλαν προσπάθειες να εξοικονομήσουν ενέργεια αυξάνοντας τη δίαρκεια ζωής της συσκευής. Μία προσέγγιση από τον Δρ. David Culler ήταν η σχεδίαση λογισμικού που επιτρέπει τις συσκευές να είναι αδρανής την περισσοτερη ώρα και κατά τακτά χρονικά διαστήματα να πέρνουνε μετρήσεις και να επικοινωνούν. Έτσι επιτυγχάνεται εξοικονόνιση την περίοδο αδρανίας. Μία άλλη προσέγγιση της ομάδας του πανεπιστημίου του Berkley είναι ο εφοδιασμός κάποιων συσκεύων με οπτικά επικοινωνιακά συστήματα με στόχο τη μείωση της ενεργειακής κατανάλωσης(2001). Με αυτή την προσέγγιση μία συσκευή που σχεδιάζεται σαν "ενεργή" εφοδιάζεται με συσκευή μετάδοσης όμοια με αυτη που βρίσκεται στους laser προβολείς. Μία άλλη προσέγγιση ήταν η "ανενεργή" μετάδοση. Μία συσκευή με τέτοιο επικοινωνιακό σύστημα ήταν εφοδιασμένη με μία σειρά καθρεφτών, οι οποίοι έστελναν μηνύματα κωδικοποιημένα σε παλμούς φωτός όταν ένα laser κατευθυνόταν προς την κατεύθυνση της συσκευής. Η προσέγγιση αυτή όμως είχε ένα σοβαρό μειονέκτημα, εξαρτιόταν από ένα κεντρικό σταθμό, που ήταν εφοδιασμένος με τη δέσμη φωτός(laser). Η προσέγγιση που έχει όμως επικρατήσει σταδιακά, ως προς το ζήτημα της επικοινωνίας είναι η ραδιομεταδόσεις. Και αυτό διότι επιτρέπουν στο σχεδιαστή να ελαχιστοποιήσει το μέγεθος και την κατανάλωση ενέργειας. Κατά τη διάρκεια της έρευνας ο καθηγητής David Culler και η ομάδα των ερευνητών του πανεπιστηνίου του Berkeley δημιούργησαν το λειτουργικό σύστημα TinyOS(2003). Όταν χρησιμοποιηθεί σε μία συσκευή μπορούμε να ελέγχουμε τις λειτουργίες της συσκευής, την κατανάλωση της ενέργειας της και την επικοινωνία με άλλες συσκευές. Σε επόμενο κεφάλαιο θα ανεφερθούμε με περισσοτερες λεπτομέρειες στο συγκεκριμένο λειτουργικό σύστημα.

## 1.3 Διαφορές των δικτύων έξυπνης σκόνης με τα κλασσικά ad-hoc δίκτυα

Τα δίκτυα έξυπνης σκόνης αποτελούν σημαντική βελτίωση σε σχέση με τους παραδοσιακούς αισθητήρες, οι οποίοι χρησιμοποιούνται με τους παρακάτω δύο τρόπους:

- Τοποθετούνται σχετικά μακριά από το φαινόμενο προς παρατήρηση, και επομένως απαιτούνται ακριβοί αισθητήρες, οι οποίοι έχουν τη δυνατότητα να φιλτράρουν το θόρυβο από το περιβάλλον, ο οποίος υπεισέρχεται ακριβώς λόγω της απόστασης από το φαινόμενο προς μέτρηση, ώστε να δίνουν αξιόπιστα αποτελέσματα.

- Τοποθετούνται σε μικρούς αριθμούς με κάποιο προσχεδιασμένο τρόπο και μεταδίδουν συνεχώς μετρήσεις ανά τακτά διαστήματα προς κάποιο κεντρικό σταθμό.

Όπως αναφέραμε και στην εισαγωγή θα μπορούσε κάποιος να αναρωτηθεί: γιατί δεν εφαρμόζουμε απλά τις μεθόδους και τα πρωτόκολλα που έχουμε αναπτύξει τα τελευταία χρόνια; . Η απάντηση είναι ότι τα δίκτυα έξυπνης σκόνης, αν και παρουσιάζουν ομοιότητες με τα κλασσικά ad hoc δίκτυα, δεν ταυτίζονται μαζί τους. Επομένως, οι ήδη υπάρχουσες μέθοδοι για ad hoc δεν επαρκούν για τα δίκτυα έξυπνης σκόνης και παραθέτουμε τους κυριότερους λόγους για αυτό:

- Ο αριθμός των κόμβων σε ένα δίκτυο έξυπνης σκόνης είναι πολύ μεγαλύτερος από ότι σε ένα ad hoc δίκτυο.

- Η πυκνότητα των κόμβων στο πεδίο είναι πολύ μεγαλύτερη.

- Οι κόμβοι είναι επιρρεπείς σε αστοχίες (failures), κυρίως σε επίπεδο υλικού (hardware), και επομένως και η τοπολογία μπορεί να αλλάζει με μεγαλύτερη συχνότητα από ότι στα ad hoc δίκτυα.

- Χρησιμοποιούνται ως επί το πλείστον τεχνικές broadcast μετάδοσης, ενώ στα ad hoc δίκτυα χρησιμοποιούνται συνδέσεις σημείο-προς-σημείο.

- Υπάρχουν κυρίαρχοι περιορισμοί σε ισχύ, μνήμη και ενέργεια.

- Μία πολύ σημαντική διαφορά είναι ότι η επικοινωνία κόμβων και βασικού σταθμού είναι δεδομενο-κεντρική (data-centric). Αυτό σημαίνει ότι ο κεντρικός σταθμός ενδιαφέρεται να μάθει πληροφορία για ορισμένες περιοχές του δικτύου, και όχι από συγκεκριμένους κόμβους. Δηλαδή, το πιο πιθανό είναι να στέλνει αιτήσεις του στυλ οι κόμβοι που βρίσκονται στην περιοχή x να στείλουν μετρήσεις , και όχι ο κόμβος 1380 να στείλει μετρήσεις .

Όπως είναι φανερό τα ήδη υπάρχοντα πρωτόκολλα και οι τεχνικές για τα Ad hoc δίκτυα δεν μπορούν να εφαρμοστούν στα δίκτυα ασύρματων αισθητήρων, όχι τουλάχιστον χωρίς σημαντικές προσθήκες και προσαρμογές στη αρχιτεκτονική και φιλοσοφία τους.

## 1.4 Εφαρμογές Στα Ασύρματα Δίκτυα Αισθητήρων

Τα ασύρματα δίκτυα αισθητήρων προορίζονται να χρησιμοποιηθούν σε μία μεγάλη ποικιλία εφαρμογών. Οι κόμβοι αυτών των δικτύων μπορούν να εξοπλιστούν με αισθητήρες θερμοκρασίας, υγρασίας, πίεσης, φωτός, επιτάχυνσης(δονήσεων), μαγνητισμού, ήχου(Crossbow 2004a). Μία γενική κατηγοριοποιήση των εφαρμογών των δικτύων αισθητήρων ειναι σύμφωνα με το [**?**] η παρακάτω :

- Στρατιωτικές
- Περιβαλλοντικές
- Περιθαλψη και γενικότερα οτι αφορά τον τομέα της υγείας
- Εφαρμογες που αφορούν την κατοικία("έξυπνη κατοικία")
- Άλλες εμπορικές εφαρμογές

Το ασυνήθηστο μικρό μέγεθος τους σε συνδυασμό με το χαμηλό κόστος και την αυτάρκια τους, επιτρέπει να χρησιμοποιηθούν σε καίριες εφαρμογές περιλαμβανομένης της προστασίας του περιβάλλοντος
(ανίχνευση και παρακολούθηση μολύνσεων), της παρακολούθησης του φυσικού περιβάλλοντος(παρατήρηση της συμπεριφοράς των ζώων στη φυσική τους κατοικία), καθώς και σε στρατιωτικά συστήματα(παρακολούθηση δραστηριότητας σε απαγορευμένες περιοχές, εντοπισμός ελεύθερων σκοπευτών[**?**], παρακολούθηση εχθρικών αλλά και συμμαχικών στρατευμάτων). Λόγω του μικροσκοπικού μεγέθους τους αναμένεται να έχουν εφαρμογή ακόμα και σε ασυνήθηστα και πρωτότυπα πεδία. Για παράδειγμα προβλέπεται οτι στο μέλλον οι κόμβοι των δικτύων αυτών θα μπόρουν να μετακινηθούν από τον αέρα ή και να παραμείνουν στον αέρα έτσι ώστε να μπόρουν να παρέχουν καλύτερη και ποιοτικότερη παρακολούθηση των καιρικών συνθηκών,της μόλυνσης του αέρα και πολλών άλλων φαινομένων. Παράλληλα γίνεται δύσκολη η ανίχνευση της παρουσίας τους και ακόμα δυσκολότερη η απομάκρυνση τους απο το χώρο που έχουν τοποθετηθεί. Κάτι που μπορεί να είναι πολύ χρήσιμο για ευαίσθητες περιοχές. Εξοπλίζοντας τα με ανιχνευτές ζωτικών σημάτων όπως η κίνηση μπορούν να ειδοποιούν τους γιατρούς και τις νοσοκόμες για την κατάσταση των ασθενών αλλά και για τη θέση των γιατρων και του υπόλοιπου προσωπικού στο νοσοκομείο. Ανιχνέυοντας τα επίπεδα νερού και χημικών ουσιών μπορούν να προειδοποιούν τους αγρότες για μελλοντικά προβλήματα στις καλλιέργιες τους. Παράλληλα θα ήταν χρήσιμα

στην πρόληψη των πυρκαγιώνΘα μπορούσαν να βελτιώσουν ακομα την θέρμανση και την ποιότητα του αέρα σε ένα κτίριο, παρακολουθόντας τη θερμοκρασία και τα ρεύματα αέρος. Τα ασύρματα δίκτυα αισθητήρων μπορούν να έχουν σημαντικό ρολο στην παρακολούθηση της κατάστασης κτιρίων δομών όπως κτίρια, γέφυρες, πλοία και αεροπλάνα. Μπορουν παραδείγματος χαριν να εντοπίσουν ζημιες μετα απο ένα καταστροφικό σεισμό. Δεν θα πρέπει να παραλείψουμε τη συμμετοχή των ασύρματων δικτύων αισθητήρων στην υλοποιηση των "έξυπνων κατοικιών"(smart homes). Πλέον κάποιοι αισθητήρες μπορούν να καθορίσουν την τοποθεσία τους μεσω δορυφόρου και GPS(Global Positioning System ). Ετσι πλεον μπορούμε να εντοπισουμε οχήματα καθώς και τους επίδοξους κλέφτες τους.
Πολλά από τα παραπάνω αναφέρπνται εκτενέστερα και λεπτομερέστερα στις διπλωματικές Αντωνίου[?] και Μυλωνά[?] γιάυτό και εμείς επίλέξαμε να παρουσιάσουμε και κάποια επίλεγμένα παραδείγματα απτών εφαρμογών, που έχουν υλοποιηθεί και δοκιμαστεί τα δύο τελευταία χρόνια. Έτσι λοιπόν παρακατω παρουσιάζουμε κάποιες εφαρμογές που ήδη υλοποιούνται και μέσω αυτων μπορούμε να αξιολογήσουμε την μεγάλη χρησιμότητα των δικτύων ασύρματων αισθητήρων.

## Παραδείγματα Υλοποιημένων Εφαρμογών

Η εταιρεία BP πειραματίζεται με motes σε ένα πετρέλαιοφορο για να καθορίσει αν είναι χρήσιμα στην πρόβλεψη λαθών στον εξοπλισμό του καταστρώματος (Economist 2004).Έτσι λοιπόν τοποθέτησε 160 motes κοντά στον εξοπλισμό του πλοίου για να μετράει "*δονήσεις στις αντλίες του πλοίου,στους συμπιεστές και στις μηχανές*". Με αυτόν τον τρόπο μπορεί να ανιχνέυσει τυχόν μελλοντικό πρόβλημα και να το επισκευάσει πριν οι συνέπειες γίνουν σοβαρότερες. Αν τα motes λειτουργήσουν όπως αναμένεται και "επιζήσουν" από το σκληρό θαλάσσιο περιβάλλον τότε μάλλον η εταιρεία θα εξοπλίσει περισσότερα πλοία της με motes.

Το 2002 ο βιολόγος John Anderson προσπάθησε να παρακολουθήσει τις ανάγκες ενός είδους θαλάσσιων πουλιών. Γι'αυτό το λόγο τοποθέτησε κόμβους έξυπνης σκόνης στις φωλιές των πουλιών για να ανιχνεύουν την παρουσία τους καθώς και να παρατηρούν τις γύρω περιβαλλοντικές συνθήκες(θερμοκρασία, υγρασία, βαρομετρική πίεση).Έτσι εξαιτίας των ασύρματων δικτύων αισθητήρων οι ερευνητές μπορούν να συλλέγουν πληροφορίες πραγματικού χρόνου για το περιβάλλον χωρίς να εισβάλλουν στις φωλιές των πουλιών.

Ο Steve Glaser, επίκουρος καθηγητής στο UC Berkeley ερευνά τη χρήση της έξυπνης σκόνης στην αξιολόγηση του δομικού θορύβου των κτιρίων και άλλων κατασκευών. Τοποθέτησε motes με αισθητήρες σεισμικών επιταχυνσιομέτρων για να ανιχνέυσει μικρές κινήσεις στα δοκάρια και τις κολώνες του κτιρίου. Έτσι μέτα από ένα σεισμό και με τη βοήθεια των motes θα μπορούν να κρίνουν

αν το κτίριο παραμένει ασφαλές για επανείσοδο των ατόμων.

Στις Η.Π.Α. μία εταιρεία (*Science Applications International Corp.*) που συνεργάζεται με την κυβέρνηση χρησιμοποιεί την έξυπνη σκόνη για την ανάπτυξη ενος ειδικού περιμετρικού συστήματος ασφαλείας για τον στρατό. Το σύστημα θα παρέχει χαμηλού κόστους συνεχή επίβλεψη κάποιου χώρου.Το σύστημα θα είναι εύκολο στην εγκατάσταση και την συντήρηση του ακόμα και από τους στρατιώτες. Παρόμοιες τεχολογίες θα μπορούν να χρησιμοποιηθούν για την κατασκευή φθηνών συστημάτων ασφαλείας για το σπίτι και το γραφείο, τα οποία θα εγκαθίστανται εύκολα και γρήγορα.

Σε μία συνέντευξη του ο Mike Horton, CEO της Crossbow είχε πει "Πιστεύουμε οτι όσον αφορά το κόστος των κλιματιστικών, τα motes μπορούν να μειώσουν κατά 20% τους λογαριασμούς του ηλεκτρικού ρεύματος, και είναι κατά 50% με 80% φθηνότερα να τα εγκαταστήσεις από οτι οι ασύρματοι αισθητήρες". ΄Ετσι σύμφωνα και με το παραπάνω θα μπορούσε ενα κτίριο με πολλά γραφεία να εξοπλιστεί με εκατοντάδες motes που θα ανίχνευαν το επίπεδο του φωτός και της θερμοκρασίας καθως και την παρουσία των ατόμων στο δωμάτιο. Το ασύρματο δίκτυο αισθητήρων τότε θα μπορούσε να παρέχει πληροφορίες σε ένα κεντρικό σύστημα, το οποίο θα έσβηνε τα φώτα αν το δωμάτιο ήταν άδειο και θα αυξομείωνε κατάλληλα τον κλιματισμό

Τα motes σύντομα θα χρησιμοποιηθούν για απομακρυσμένη περιβαλλοντική παρακολούθηση σε μία προσπάθεια να διατηρηθούν οι ζωγραφισμένοι τοίχοι και οι οροφές Βουδιστικών ναών που βρίσκονται σε σπηλιές δίπλα στην κινεζικη πόλη Dunhuang(Pescovitz 2003, cf. Weintruab 2004). Αυτή τη στιγμή οι συντηρητές παρακολουθούν το περιβάλλον των σπηλιών για να προσδιορίσουν τις συνθήκες που προκαλούν τη διάβρωση των τοίχων. Το θέμα είναι οτι χρειάζεται να μπούν στις σπηλιές για να συλλέξουν πληροφορίες και έτσι εντείνεται η διάβρωση. ΄Ετσι ο Steven Glaser, επίκουρος καθηγητής στο UC Berkeley και η ομάδα του θα τοποθετήσουν motes που θα παρακολουθούν την υγρασία, τη θερμοκρασία, τη βαρομετρική πίεση καθώς και τους κραδασμούς. ΄Ετσι οι συντηρητές πλεον θα μπορούν να συλλέγουν στοιχεία χωρίς να εισέρχονται στις σπηλιές.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η βιοτεχνολογική προσέγγιση των ασύρματων δικτύων αισθητήρων από επιστήμονες στο πανεπιστήμιο της California στο San Diego. Κατάφεραν να φτιάξουν motes με χημικές ενώσεις και οχι με ολοκληρωμένα κυκλώματα. Παρόλο που δεν έχουν βγει στην αγορά ακόμα έχουν αποδειχθεί πολύ χρήσιμα στα εργαστήρια τους. Σε ένα πείραμα τους μάλιστα κατάφεραν να ανιχνεύσουν την παρουσία αέριου υδρογονάνθρακα από απόσταση περίπου 20 μέτρων.

## 1.5 Μειονεκτήματα - Ιδιαιτερότητες

Μολονότι το μικροσκοπικο μέγεθος των κομβων των δικτυων αισθητήρων ειναι υπευθυνο για το πληθος των εφαρμογων που περιγράψαμε παραπάνω γίνεται αιτία και για αρκετά "προβλήματα".

Αρχικά οι φυσικές ζημιές που μπορούν να προκληθούν απο την ρίψη τον κόμβων ή από τις συνθήκες του φυσικού περιβάλλοντος είναι ένα πιθανό σενάριο. Η ρίψη συσκευών μπορεί να γίνει από αεροπλάνο (η συγκεκριμένη μέθοδος έχει ήδη δοκιμαστεί), από καταπέλτη, με τα χέρια, κτλ. Όπως είναι ευνόητο κάποιες συσκευές θα πάθουν ανεπανόρθωτες ζημιές. Παράλληλα η τοπολογία του δικτύου που θα σχηματιστεί μπορεί να είναι εντελώς τυχαία. Ακόμα χειρότερα όμως μπορεί η τοπολογία να μην επιτρέπει σε κάποιες συσκεύες να επικοινωνούν με τις γειτονικές τους. Όλα αυτά τα ζητήματα πρέπει ο σχεδιαστής και ο χρήστης τέτοιων δικτύων να τα λάβει σοβαρά υπόψιν του. Παρόλο αυτά έχουν υλοποιηθεί αλγόριθμοι και πρακτικές που καταφέρνουν να υπερκεράσουν αυτές τις δυσκολίες ή έστω να τις κάνουν να φαίνονται ασήμαντες μπροστά στις λειτουργίες τέτοιων δικτύων.

Το θέμα της κατανάλωσης ενέργειας παίζει επίσης σπουδαίο ρόλο για τα δίκτυα αισθητήρων. Πολλές φορές η επαναφόρτιση της μπαταρίας τους είναι αδύνατη. Ετσι η διάρκεια ζωης τους εξαρτάται απο την διάρκεια της μπαταρίας. Η πλέον δαπανηρή πράξη φαίνεται να είναι η επικοινωνία(όχι για όλες τις πλαρφόρμες, όπως θα δούμε σε παρακάτω κεφάλαιο). Στην επικοινωνία περιλαμβάνεται τόσο η μετάδοση όσο και η λήψη δεδομένων. Γενικά, για μικρές αποστάσεις και για τις δύο περιπτώσεις καταναλώνεται περίπου η ίδια ενέργεια. Επίσης, λαμβάνουμε υπόψη μας και την ενέργεια για την αρχικοποίηση του πομποδέκτη. Επομένως, πρέπει να βρούμε έναν έξυπνο τρόπο να διαχειριστούμε τον πομποδέκτη και να ελαχιστοποιήσουμε μεταδόσεις και λήψεις μηνυμάτων. Τυπικές τιμές για αυτήν την περιοχή είναι 50 nJ/bit. Έχουν προταθεί κάποια πρωτόκολλα που εξοικονομούν ενέργεια αλλά αυτό δεν φαίνεται να επαρκεί. Έχουν γίνει προσπάθειες για ενσωμάτωση ηλιακών κυψελιδών , οι οποίες παράγουν ενέργεια από το φώς. Το φως αυτό μπορεί να προέρχεται είτε από τον ήλιο, είτε από κάποια άλλη φωτεινή πηγή, όπως ο εσωτερικός φωτισμός. Ακόμα όμως δεν έχει καρποφορήσει η λύση αυτή κυρίως λόγω κόστους. Γενικότερα πάντως έχουν κατά καιρους υπάρξει πολύ πρωτότυπες ιδέες, οι οποίες όμως είτε δεν είναι συμφέρουσες σε κάποιον άλλον τομέα είτε δεν έχουν τύχει κατάλληλης προσοχής και αξιοποίησης.

Ένα ακόμα θέμα που θα πρέπει να ληφθεί υπόψιν από τους κατασκευαστές είναι αυτό του κόστους. Δηλαδή θα πρέπει το κόστος κάθε συσκευής να είναι πολύ μικρότερο από ενα παραδοσιακό κόμβο δικτύου αισθητήρων. Τη συγκεκριμένη στιγμή που γράφουμε μία τέτοια συσκευή κοστίζει γύρω στα 130$. Μία τιμή που ίσως να μην φαινεται τόσο προσιτή τώρα. Παρόλο αυτά

οι προβλέψεις του καθηγητή Pister, είναι αισιόδοξες και όπως χαρακτηριστικά αναφέρει το 2010 οι ράδιοσυσκευές θα κοστίζουν 0,10$.

Αυτά είναι τα κύρια προβλήματα που καλούνται να αντιμετωπίσουν οι σχεδιαστές συσκευών δικτύων ασύρματων αισθητήρων. Επειδή όμως ο χώρος των εφαρμογών που καλούνται να καλύψουν τα δίκτυα αυτα είναι μεγάλος και ανομοιόμορφος τα προβήματα δεν θα είναι ίδια για κάθε εφαρμογή και όσο αυξάνουν οι εφαρμογές που θα θέλουν να ενσωματώσουν τέτοιες συσκευές στην υλοποίηση τους, θα παρουσιάζονται μοιραία περισσότερα και πιο εξειδικευμένα προβλήματα. Έχοντας όμως ως σύμμαχο την ραγδαία εξέλιξη της τεχνολογίας και την εφευρετικότητα του ανθρώπου θα υπάρχει πάντα μία λυση για κάθε πρόβλημα.

# Κεφάλαιο 2

# Πλατφόρμες για Ασύρματα Δίκτυα Αισθητήρων

## 2.1 Γενικά

Ο στόχος των σχεδιαστών των ασύρματων δικτύων αισθητήρων επικεντρώνεται στο να φτιάξουν πλατφόρμες διαστάσεων κυβικών χιλιοστών που θα σχηματίζουν κατανεμημένα δίκτυα αισθητήρων και θα μπορούν να παρατηρούν διάφορα φαινόμενα. Κάθε πλατφόρμα αποτελείται απο διαφορετικά υποσυστήματα κατασκευασμένα με διαφορετικές πολλές φορές τεχνολογίες. Κάθε τέτοια πλατφόρμα μπορεί να διαχωριστεί σε 4 κύρια υποσυστήματα : τους αισθητήρες και τα αναλογικά σήματα, το τροφοδοτικό σύστημα, το σύστημα επικονωνίας και τον πυρήνα, ο οποίος ουσιαστικά περιέχει όλα τα ψηφιακά κυκλώματα περιλαμβανομένων των κυκλωμάτων για τους αισθητήρες, για την ασύρματη επικοινωνία, για τους υπολογισμούς και για τη μνήμη.

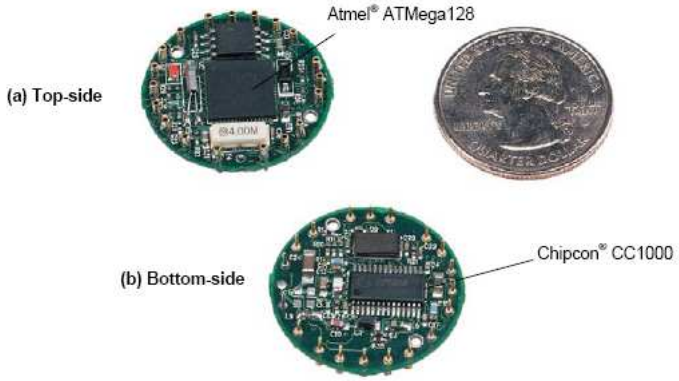
## 2.2 Berkeley Motes ( Crossbow )

Διάφορες εταιρίες έχουν ασχοληθεί με την παραγωγή τέτοιων πλατφορμών. Μία απο τις κύριες είναι η Crossbow[?] η οποία παράγει τα Berkeley Motes. Η Crossbow αυτή τη στιγμή παράγει 4 διαφορετικές πλατφόρμες.

- **Mica2Dot**

  Η πρώτη πλατφόρμα της Crossbow ειναι τα Mica2Dot motes. Βγήκε στην αγορά το 2002 και αποτελεί το παλαιότερο μοντέλο που παράγει αυτή τη στιγμή η Crossbow. Έχει σχεδιαστεί για εφαρμογές στις οποίες παίζει ρόλο το μέγεθος των κόμβων. Κατατάσσεται στις πλατφόρμες τρίτης γενιάς και έχει μέγεθος που δεν ξεπερνά τα 25 χιλιοστά. Το Mica2Dot mote παράγεται σε τρία διαφορετικά μοντέλα ανάλογα με τη ραδιοσυχνότητα που χρησιμοποιεί στην επικοινωνία : το MPR400(915MHz), το MPR410 (433 MHz), και το MPR420 (315 MHz) μοντέλο. Χρησιμοποιεί τον Chipcon CC1000, ο οποίος είναι ένας UHF RF πομποδέκτης που χρησιμοποιεί

διαμόρφωση σήματος κατά συχνότητα και μπορεί να επικοινωνεί σε ρυθμούς μέχρι 38,4 Kbaud. Όλα τα μοντέλα χρησιμοποιούν έναν ισχυρό 8-bit Atmega128L μικροελεγκτή με ταχύτητα διάυλου 4 MHz. Ό Atmega128L έχει ενσωματωμένη 4kb RAM και 128kb ROM. Παράλληλα η συγκεκριμένη πλατφόρμα διαθέτει και 512kb μνήμη flash για αποθήκευση εξωτερικών πληροφοριών και κώδικα. Έχει ενσωματωμένο αισθητήρα θερμοκρασίας και δυνατότητα παρακολούθησης της "ζωής" της μπαταριάς. Τέλος η συγκεκριμένη πλατφόρμα παρέχει τη δυνατότητα απομακρυσμένου ασύρματου επαναπρογραμματισμού της.




Σχήμα 2.1: *Φωτογραφία ενός Mica2Dot(MPR4x0) δίπλα σε ένα νόμισμα. Κανονικά κάθε Mica2Dot έχει μία 3V μπαταρία σε σχήμα νομίσματος τοποθετημένη στο πίσω μέρος αλλά εδώ παραλείπεται για να φάνουν οι λεπτομέρειές.( Πηγή Εικόνας: www.crossbow.com )*

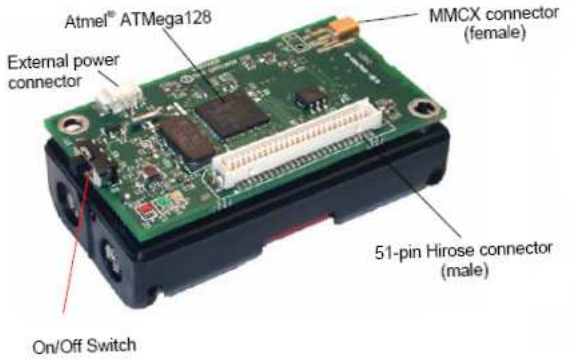
Αναλυτικότερα οι διαφορές και τα χαρακτηριστικά των 3 μοντέλων Mica2Dot φαίνονται στον παρακάτω πίνακα ανάλογα με τη συχνότητα των ραδιοπομπών τους.

| Processor/Radio Board | MPR500CA | MPR510CA | MPR520CA |
|---|---|---|---|
| **Processor Performance** | | | |
| Program Flash Memory | 128K bytes | 128K bytes | 128K bytes |
| Measurement (Serial) Flash | 512K bytes | 512K bytes | 512K bytes |
| Configuration EEPROM | 4 K bytes | 4 K bytes | 4 K bytes |
| Serial Communications | UART | UART | UART |
| Analog to Digital Converter | 10 bit ADC | 10 bit ADC | 10 bit ADC |
| Other Interfaces | DIO | DIO | DIO |
| Current Draw | 8 mA | 8 mA | 8 mA |
| | < 15 uA | < 15 uA | < 15 uA |
| **Multi-Channel Radio** | | | |
| Center Frequency | 868/916 MHz | 433 MHz | 315 MHz |
| Number of Channels | 4 / 50 | 4 | 5 |
| Data Rate | 38.4 Kbaud | 38.4 Kbaud | 38.4 Kbaud |
| RF Power | -20 - +5 dBm | -20 - +10 dBm | -20 - +10 dBm |
| Receive Sensitivty | -98 dBm | -101 dBm | -101 dBm |
| Outdoor Range | 500 ft | 1000 ft | 1000 ft |
| Current Draw | 27 mA | 25 mA | 25 mA |
| | 10 mA | 8 mA | 8 mA |
| | < 1 uA | < 1 uA | < 1 uA |
| **Electromechanical** | | | |
| Battery | 3V Coin Cell | 3V Coin Cell | 3V Coin Cell |
| External Power | 2.7 - 3.3 V | 2.7 - 3.3 V | 2.7 - 3.3 V |
| User Interface | 1 LED | 1 LED | 1 LED |
| Size (in) | 1.0 x 0.25 | 1.0 x 0.25 | 1.0 x 0.25 |
| (mm) | 25 x 6 | 25 x 6 | 25 x 6 |
| Weight (oz) | 0.11 | 0.11 | 0.11 |
| (grams) | 3 | 3 | 3 |
| Expansion Connector | 18 pins | 18 pins | 18 pins |

Σχήμα 2.2: *Συκριτικός πίνακας των τριών μοντέλων Mica2Dot. ( Πηγή Εικόνας: www.crossbow.com )*

- **Mica2**

Η επόμενη πλατφόρμα της Crossbow είναι τα λεγόμενα Mica2 motes, τα οποία βγήκαν στην αγορά το 2002. Το Mica2 mote είναι μία τρίτης γενιάς πλατφόρμα που έχει σχεδιαστεί ειδικά για δίκτυα αισθητήρων ειδικού σκοπού και χρησιμοποιείται σε χαμηλής κατανάλωσης ασύρματα δίκτυα αισθητήρων.Το Mica2 mote παράγεται σε τρία διαφορετικά μοντέλα όπως ακριβως και το Mica2Dot : το MPR400(915MHz), το MPR410 (433 MHz), και το MPR420 (315 MHz) μοντέλο. Χρησιμοποιεί όπως και το Mica2Dot τον μεταβαλλόμενης συχνότητας πομποδέκτη Chipcon CC1000, και μπορεί να επικοινωνεί σε ρυθμούς μέχρι 38,4 Kbaud. Όλα τα μοντέλα χρησιμοποιούν έναν ισχυρό 8-bit Atmega128L μικροελεγκτή με ταχύτητα διάυλου 7,37 MHz. Όπως ακριβώς στις Mica2Dot πλατφόρμες, ο Atmega128L έχει ενσωματωμένη 4kb RAM και 128kb ROM. Παράλληλα η συγκεκριμένη πλατφόρμα διαθέτει και 512kb μνήμη flash για αποθήκευση πληροφοριών. Έχει χρόνο ζωής περισσότερο του ενός χρόνου με AA μπαταρίες(λειτουργώντας σε Sleep Modes) και έχει υποδοχή για πλακέτες αισθητήρων της Crossbow(αισθητήρες φωτός, θερμοκρασίας, βαρομετρικής πίεσης, επιτάχυνσης, ήχου, μαγνητικού πεδίου, υγρασίας και άλλων). Και αυτή η πλατφόρμα παρέχει δυνατότητα απομακρυσμένου ασύρματου επαναπρογραμματισμού της. Τέλος θα πρέπει να αναφέρουμε ότι οι πλατφόρμες Mica2 είναι συμβατές με τις Mica2Dot, κάτι που επιτρέπει την επικοινωνία μεταξύ τους και συνεπώς τον συνδυασμό τους για την δημιουργία ενός δικτύου ασύρματων αισθητήρων.




Σχήμα 2.3: *Φωτογραφία ενός Mica2(MPR4x0) χωρίς κεραία. ( Πηγή Εικόνας: www.crossbow.com )*

Αναλυτικότερα οι διαφορές και τα χαρακτηριστικά των 3 μοντέλων mica2 φαίνονται στον παρακάτω πίνακα ανάλογα με τη συχνότητα των ραδιοπομπών τους.

| Processor/Radio Board | MPR400CB | MPR410CB | MPR420CB[1] |
|---|---|---|---|
| **Processor Performance** | | | |
| Program Flash Memory | 128K bytes | 128K bytes | 128K bytes |
| Measurement (Serial) Flash | 512K bytes | 512K bytes | 512K bytes |
| Configuration EEPROM | 4 K bytes | 4 K bytes | 4 K bytes |
| Serial Communications | UART | UART | UART |
| Analog to Digital Converter | 10 bit ADC | 10 bit ADC | 10 bit ADC |
| Other Interfaces | DIO,I2C,SPI | DIO,I2C,SPI | DIO,I2C,SPI |
| Current Draw | 8 mA | 8 mA | 8 mA |
| | < 15uA | < 15 uA | < 15 uA |
| **Multi-Channel Radio** | | | |
| Center Frequency | 868/916 MHz | 433 MHz | 315 MHz |
| Number of Channels | 4 / 50 | 4 | 5 |
| Data Rate | 38.4 Kbaud | 38.4 Kbaud | 38.4 Kbaud |
| RF Power | -20 to +5 dBm | -20 to +10 dBm | -20 to +10 dBm |
| Receive Sensitivty | -98 dBm | -101 dBm | -101 dBm |
| Outdoor Range | 500 ft | 1000 ft | 1000 ft |
| Current Draw | 27 mA | 25 mA | 25 mA |
| | 10 mA | 8 mA | 8 mA |
| | < 1 uA | < 1 uA | < 1 uA |
| **Electromechanical** | | | |
| Battery | 2X AA batteries | 2X AA batteries | 2X AA batteries |
| External Power | 2.7 - 3.3 V | 2.7 - 3.3 V | 2.7 - 3.3 V |
| User Interface | 3 LEDs | 3 LEDs | 3 LEDs |
| Size (in) | 2.25 x 1.25 x 0.25 | 2.25 x 1.25 x 0.25 | 2.25 x 1.25 x 0.25 |
| (mm) | 58 x 32 x 7 | 58 x 32 x 7 | 58 x 32 x 7 |
| Weight (oz) | 0.7 | 0.7 | 0.7 |
| (grams) | 18 | 18 | 18 |
| Expansion Connector | 51 pin | 51 pin | 51 pin |

Σχήμα 2.4: *Συκριτικός πίνακας των τριών μοντέλων Mica2. ( Πηγή Εικόνας: www.crossbow.com )*

- **Micaz**

  Μία από τις τελευταίες πλατφόρμες της Crossbow είναι τα Micaz motes, τα οποία βγήκαν στην αγορά το 2005. Η καινοτομία των micaz σε σχέση με τα mica2 και τα mica2dot είναι η υιοθέτηση του IEEE 802.15.4 προτύπου και των ZigBee προδιαγραφών. Κάτι που παρέχει αυτόματα μεγαλύτερη ταχύτητα μεταφοράς καθώς και ασφάλεια στις επικοινωνίες. Χρησιμοποιεί τον Chipcon CC2420, ο οποίος εκμπέμπει σε συχνότητα 2,4Ghz και μπορεί να επικοινωνεί σε ρυθμούς των 250Kbps. Ό συγκεκριμένος ραδιοπομπός είναι, όπως αναφέραμε και παραπάνω, συμβατός με τις ZigBee προδιαγραφές και το IEEE 802.15.4 πρότυπο. Όλα τα μοντέλα χρησιμοποιούν έναν ισχυρό 8-bit Atmega128L μικροελεγκτή με ταχύτητα διάυλου 7,37 MHz, που ενσωματώνει 4kb RAM και 128kb ROM. Παράλληλα διαθέτει 512kb μνήμη flash για αποθήκευση πληροφοριών, όπως ακριβώς και οι δύο προηγούμενες πλατφόρμες. Έχει την ίδια υποδοχή για πλακέτες αισθητήρων της Crossbow με την Mica2 πλατφόρμα(αισθητήρες φωτός, θερμοκρασίας, βαρομετρικής πίεσης, επιτάχυνσης, ήχου, μαγνητικού πεδίου, υγρασίας και άλλων). Και αυτή η πλατφόρμα παρέχει δυνατότητα απομακρυσμένου ασύρματου επαναπρογραμματισμού της.

Σχήμα 2.5: *Φωτογραφία ενός Micaz με κεραία. ( Πηγή Εικόνας: www.crossbow.com )*

| Processor/Radio Board | MPR2400CA |
|---|---|
| **Processor Performance** | |
| Program Flash Memory | 128K bytes |
| Measurement Serial Flash | 512K bytes |
| Configuration EEPROM | 4 K bytes |
| Serial Communications | UART |
| Analog to Digital Converter | 10 bit ADC |
| Other Interfaces | Digital I/O,I2C,SPI |
| Current Draw | 8 mA |
| | < 15uA |
| **RF Transceiver** | |
| Frequency band[1] | 2400 MHz to 2483.5 MHz |
| Transmit (TX) data rate | 250 kbps |
| RF power | -24 dBm to 0 dBm |
| Receive Sensitivity | -90 dBm (min), -94 dBm (typ) |
| Adjacent channel rejection | 47 dB |
| | 38 dB |
| Outdoor Range | 75 m to 100 m |
| Indoor Range | 20 m to 30 m |
| Current Draw | 19.7 mA |
| | 11 mA |
| | 14 uA |
| | 17.4 mA |
| | 20 µA |
| | 1 µA |
| **Electromechanical** | |
| Battery | 2X AA batteries |
| External Power | 2.7 V - 3.3 V |
| User Interface | 3 LEDs |
| Size (in) | 2.25 x 1.25 x 0.25 |
| (mm) | 58 x 32 x 7 |
| Weight (oz) | 0.7 |
| (grams) | 18 |
| Expansion Connector | 51 pin |

Σχήμα 2.6: *Πίνακας με τα χαρακτηριστικά ενος Micaz. ( Πηγή Εικόνας: www.crossbow.com )*

- **Telos-B**

Πριν αρχίσουμε να αναφέρουμε χαρακτηριστικά για τη συγκεκριμένη πλατφόρμα θα πρέπει, ως ένδειξη της ρευστότητας που χαρακτηρίζει το χώρο, να αναφέρουμε ότι όταν πρωτογράφαμε(Αύγουστος 2005) το παρόν κεφάλαιο η συγκεκριμένη πλατφόρμα άνηκε στην εταιρεία MoteIV και συνεπώς την είχαμε ταξινομήσει στην παρακάτω παράγραφο. Παρόλ' αυτά αργότερα διαπιστώσαμε ότι η συσκευή "πήρε μεταγραφή" και ανήκει πλεόν στην Crossbow υπό την "ταμπέλα" της ερευνητικής συσκευής, η οποία προορίζεται να χρησιμοποιηθεί για πειράματα με ασύρματα δίκτυα αισθητήρων και γενικότερα στην έρευνα.
Όπως αναφέρει και η ίδια η Crossbow η Telos-B συσκευή είναι μια open source πλατφόρμα σχεδιασμένη έτσι ώστε να επιτρέπει πειραματισμούς στην ερευνητική κοινότητα. Υποστηρίζει και αυτή ,όπως και η Micaz, το IEEE 802.15.4 πρότυπο και τις ZigBee προδιαγραφές. Χρησιμοποιεί τον 16-bit TI MSP430 μικροελεγκτή της Texas Instruments , ο οποίος δουλεύει στα 8Mhz και έχει 10kb RAM και 48kb ROM. Χρησιμοποιεί τον Chip-con CC2420, ο οποίος εκμπέμπει σε συχνότητα 2,4-2,4835Ghz και μπορεί να επικοινωνεί σε ρυθμούς των 250Kbps. Παράλληλα έχει εξωτερική μνήμη flash 1Mb. Η Telos-B αντίθετα με τις προηγούμενς συσκευές της Crossbow έχει ενσωματωμένη USB υποδοχή, την οποία μπορεί κάποιος να συνδέσει απευθείας στην αντίστοιχη USB θύρα του υπολογιστή και να προγραμματίσει την συσκευή. Τέλος παράγεται σε δύο εκδοχές, μία χωρις αισθήρες και μία με αισθητήρες της Crossbow ενσωματωμένους.

Σχήμα 2.7: *Φωτογραφία ενός Telos-B με κεραία. ( Πηγή Εικόνας: www.crossbow.com )*

| Specifications | TPR2400CA |
|---|---|
| **Module** | |
| Processor Performance | 16-bit RISC |
| Program Flash Memory | 48K bytes |
| Measurement Serial Flash | 1024K bytes |
| RAM | 10K bytes |
| Configuration EEPROM | 16K bytes |
| Serial Communications | UART |
| Analog to Digital Converter | 12 bit ADC |
| Digital to Analog Converter | 12 bit DAC |
| Other Interfaces | Digital I/O,I2C,SPI |
| Current Draw | 1.8 mA |
| | 5.1 uA |
| **RF Transceiver** | |
| Frequency band[1] | 2400 MHz to 2483.5 MHz |
| Transmit (TX) data rate | 250 kbps |
| RF power | -24 dBm to 0 dBm |
| Receive Sensitivity | -90 dBm (min), -94 dBm (typ) |
| Adjacent channel rejection | 47 dB |
| | 38 dB |
| Outdoor Range | 75 m to 100 m |
| Indoor Range | 20 m to 30 m |
| Current Draw | 23 mA |
| | 21 µA |
| | 1µA |

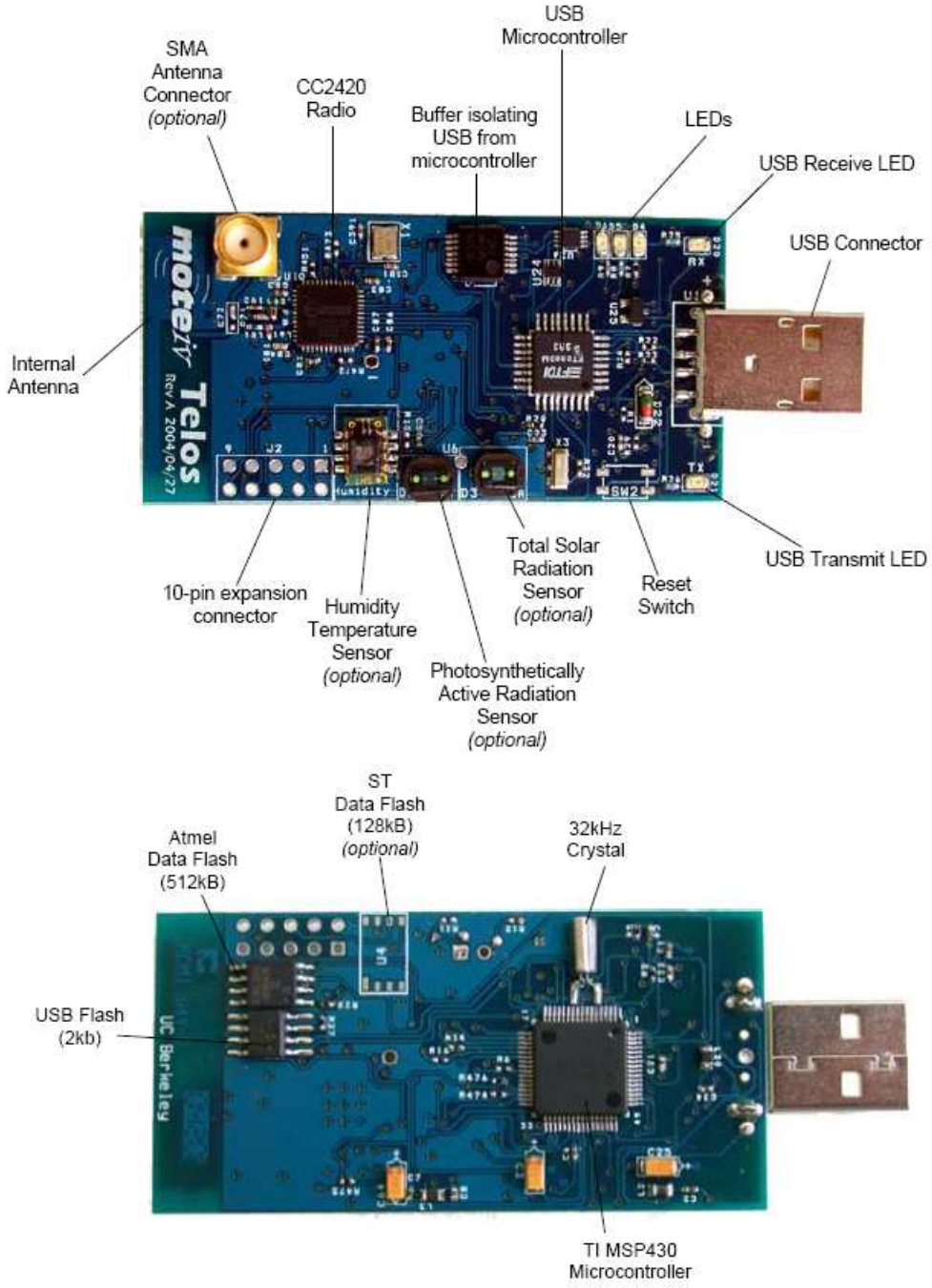
Σχήμα 2.8: *Πίνακας με τα χαρακτηριστικά ενος Telos-B. ( Πηγή Εικόνας: www.crossbow.com )*

## 2.3 Moteiv

Η εταιρεία moteiv[?] ιδρύθηκε το 2003. Από τότε έχει κυκλοφορήσει δύο πλατφόρμες. Την Telos και την Tmote sky. Η Telos κυκλοφόρησε το 2004 αλλά πρόσφατα αποσύρθηκε για να πάρει τη θέση της η Tmote sky. Η Telos βγήκε σε δύο μοντέλα, το Telos Revision A και το Telos Revision B. Η Tmote sky, η οποία κυκλοφόρησε το 2005, είναι ουσιαστικά η συνέχεια του Telos Revision B αφού όπως θα δούμε παρακάτω έχουν πολλά κοινά χαρακτηριστικά. Πρόσφατα όπως αναφέραμε και στην προηγούμενη ενότητα η Telos αγοράστηκε από την Crossbow και πλέον δεν παράγεται καμία Telos πλατφόρμα από την MoteIV.

- **Telos Revision A**

  Αν και πλέον δεν παράγεται, και στην επίσημη ιστοσελίδα της εταιρείας, μας παραπέμπουν στην πλατφόρμα Tmote sky, επιλέξαμε να παρουσιάσουμε συντόμως την Telos Revision A αφού πάνω σε αυτή στηρίχτηκε και η Telos-B και η Tmote sky. Η πλατφόρμα Telos Revision A , λοιπόν, χρησιμοποιεί τον MSP430 F149, ένα 16-bit Risc μικροελεγκτή της εταιρείας Texas Instruments, ο οποίος δουλεύει στα 8Mhz και έχει 2kb RAM και 60kb ROM. Παράλληλα παρέχει και 256kb εξωτερική μνήμη flash. Για την ασύρματη επικοινωνία χρησιμοποιεί τον Chipcon CC2420, ο οποίος δουλέυει σύμφωνα με το IEEE 802.15.4 πρότυπο και μεταδίδει στα 2,4Ghz μέχρι 250kbps. Έχει ενσωματωμένους αισθητήρες θερμοκρασίας και υγρασίας και υλοποιεί, μέσω hardware, επίπεδο πιστοποίησης και απόκρυψης δεδομένων. Τέλος θα πρέπει να αναφέρουμε ότι αυτή η πλατφορμα έχει βεληνεκές γύρω στα 125m σε εξωτερικό περιβάλλον.



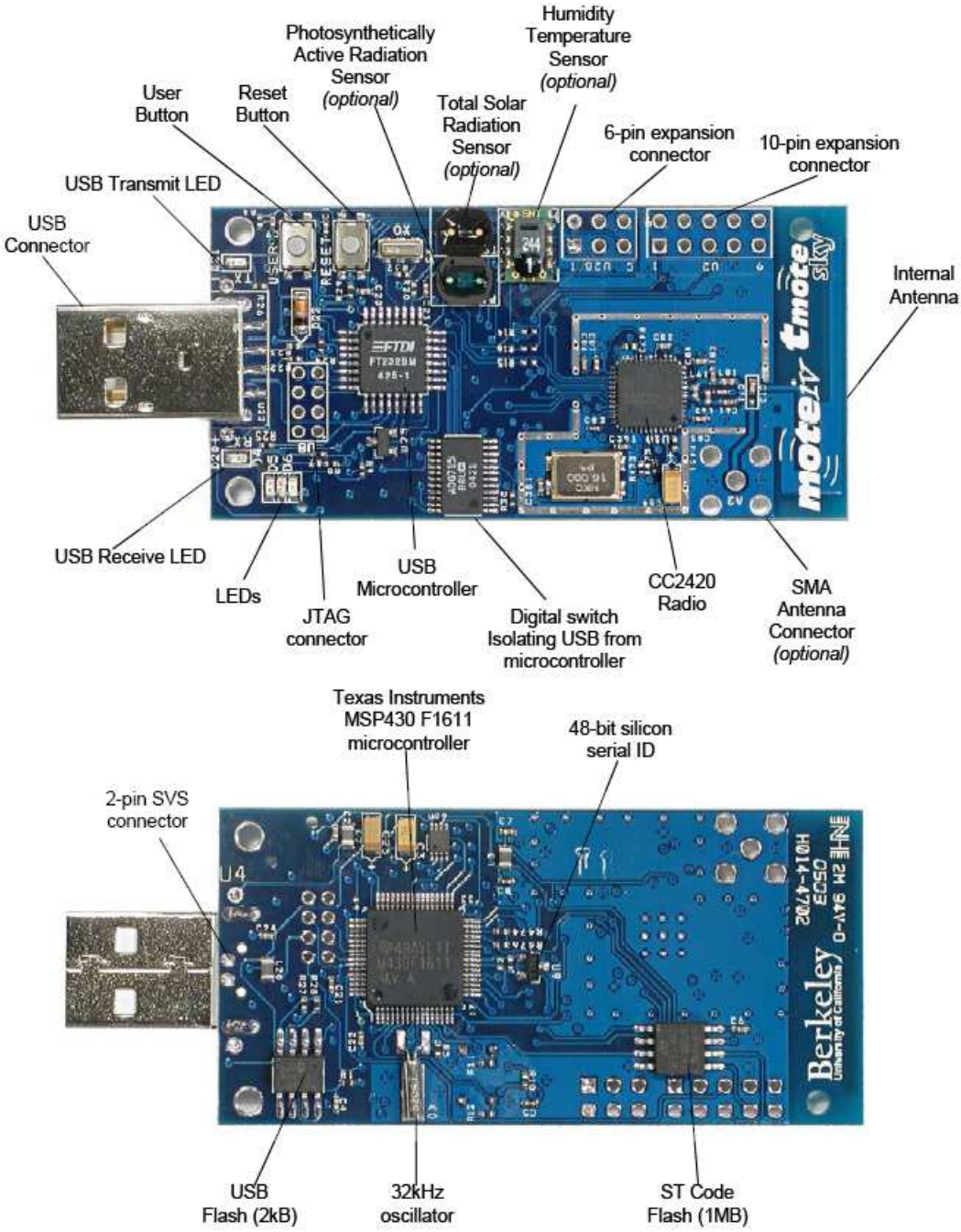

Σχήμα 2.9: *Εμπρός και πίσω όψη ενος Telos Revision A(χωρίς μπαταρία). ( Πηγή Εικόνας: www.moteiv.com )*

| | |
|---|---|
| **CPU** | |
| Bus Speed | 8 MHz |
| RAM | 2 Kb |
| Program Space | 60 Kb |
| External Flash | 256 Kb |
| Serial Communications | DIO,SPI,I2C,UART |
| Current (active w/ Radio on) | 19 mA |
| Current (sleep) | 2.4 uA |
| Voltage | 1.8-3.6 V |
| **Radio** | |
| Frequency | 2400-2483 MHz |
| Data rate | 250 kbps |
| Output Power | -25 to 0 dBm |
| Antenna Type | Inverted-F or SMA Coax |
| **Humidity Sensor** | |
| Humidity Accuracy | 3.5% RH |
| Temperature Accuracy | 0.5 $^{o}$C |
| Sampling Rate | 90 Hz |
| **Electromechanical** | |
| Battery | AA, 2/3A |

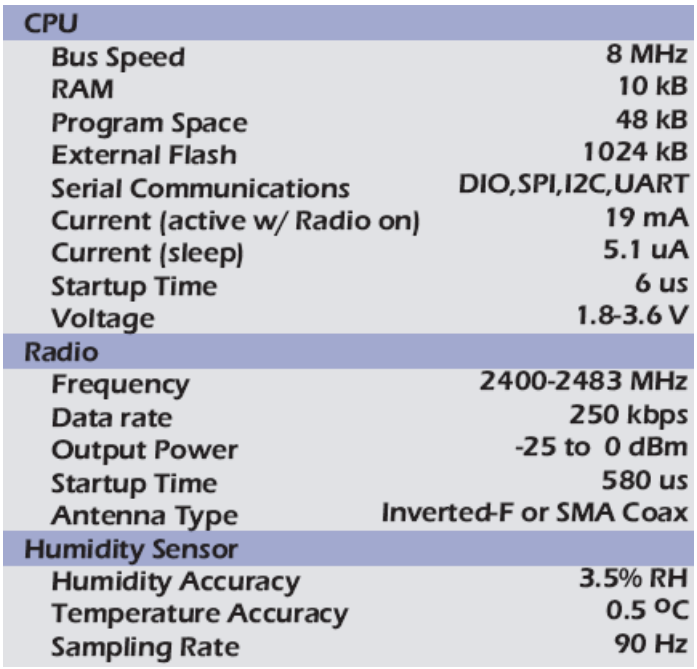
Σχήμα 2.10: *Χαρακτηριστικά ενός Telos Revision A. ( Πηγή Εικόνας: www.moteiv.com )*

- **Tmote Sky**

  Η πλατφόρμα Tmote Sky χρησιμοποιεί τον MSP430 F1611, ένα 16-bit Risc μικροελεγκτή της Texas Instruments, ο οποίος δουλέυει στα 8Mhz και έχει 10kb RAM, 48kb ROM και 128b για αποθήκευση πληροφοριών. Παράλληλα η εξωτερική μνήμη flash που είχε και η Telos αυξάνεται στα 1024kb. Για την ασύρματη επικοινωνία χρησιμοποιεί και αυτή τον Chipcon CC2420, ο οποίος δουλέυει σύμφωνα με το IEEE 802.15.4 πρότυπο και μεταδίδει στα 2,4Ghz μέχρι 250kbps. Η μόνη διαφορά με την Telos είναι ότι η Tmote Sky χρησιμοποιεί ένα μεγαλύτερο 16Mhz κρυσταλλικό ταλαντωτή για τον Chipcon CC2420 και έτσι επιτυγχάνει καλύτερους χρόνους εκκινησης στην μετάδοση(λιγότερο από 580μs). Έχει ενσωματωμένους αισθητήρες θερμοκρασίας και υγρασίας και υλοποιεί, μέσω hardware, επίπεδο πιστοποίησης και απόκρυψης δεδομένων. Όπως παρατηρουμε οι πλατφόρμες Tmote Sky και Telos-B είναι σχεδόν ίδιες. Η διαφορά τους έγκειται σε μία ιδιότητα του Tmote Sky να φορτίζει τις μπαταρίες του όταν είναι συνδεδεμένο με τον υπολογιστή, αν βέβαια αυτές είναι επαναφορτιζόμενες.




Σχήμα 2.11: *Εμπρός και πίσω όψη ενός Tmote Sky(χωρίς μπαταρία). ( Πηγή Εικόνας: www.moteiv.com )*

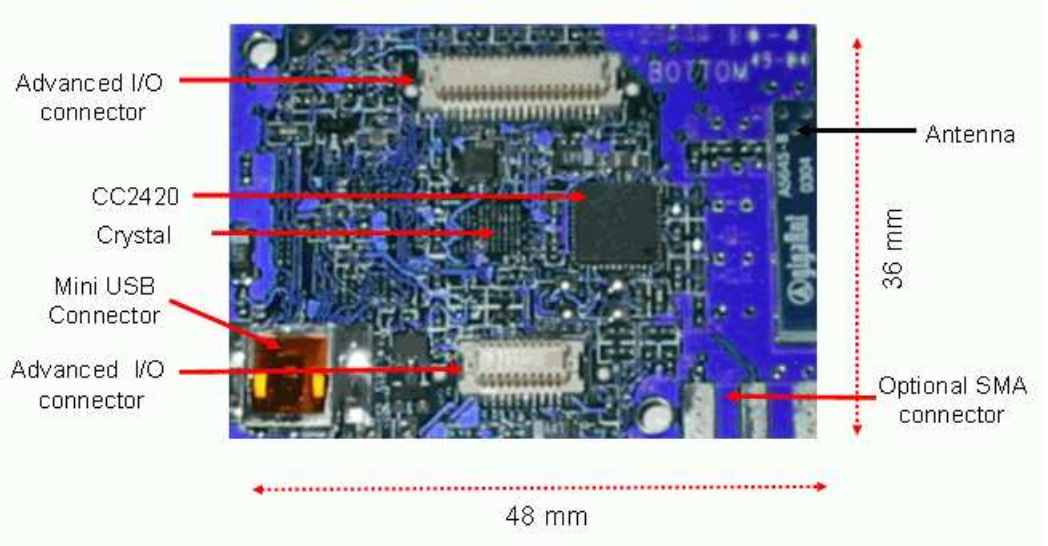


| CPU | |
|---|---|
| Bus Speed | 8 MHz |
| RAM | 10 kB |
| Program Space | 48 kB |
| External Flash | 1024 kB |
| Serial Communications | DIO,SPI,I2C,UART |
| Current (active w/ Radio on) | 19 mA |
| Current (sleep) | 5.1 uA |
| Startup Time | 6 us |
| Voltage | 1.8-3.6 V |
| **Radio** | |
| Frequency | 2400-2483 MHz |
| Data rate | 250 kbps |
| Output Power | -25 to 0 dBm |
| Startup Time | 580 us |
| Antenna Type | Inverted-F or SMA Coax |
| **Humidity Sensor** | |
| Humidity Accuracy | 3.5% RH |
| Temperature Accuracy | 0.5 °C |
| Sampling Rate | 90 Hz |

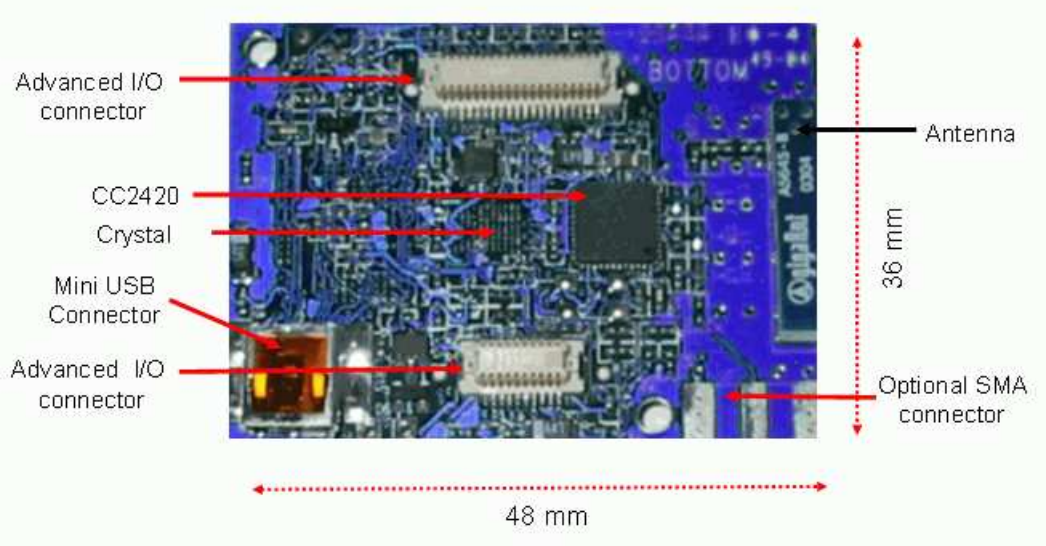
Σχήμα 2.12: *Χαρακτηριστικά ενός Tmote Sky. ( Πηγή Εικόνας: www.moteiv.com )*

## 2.4 Intel

Μία ακόμα εταιρεία που έχει ασχοληθεί με την έρευνα και την παραγωγή motes είναι η Intel. Η έρευνα άρχισε γύρω στα μέσα του 2002 και η πρώτη έκδοση του iMote έκανε την εμφάνιση της μέσα στο 2003. Η τωρινή έκδοση του iMote(Intel Mote) χρησιμοποεί τον PXA273 μικροεπεξεργαστή της ίδιας της Intel, ο οποίος τρέχει στα 13Mhz. Παράλληλα έχει 256kb SRAM, και επιλογή για 32Mb FLASH/SDRAM μνήμης. Για την ασύρματη επικοινωνία χρησιμοποιεί τον Chipcon CC2420(με 16Mhz κρυσταλλικό ταλαντωτή), ο οποίος δουλέυει σύμφωνα με το IEEE 802.15.4 πρότυπο και μεταδίδει στα 2,4Ghz μέχρι 250kbps. Εκτός απο τον Chipcon CC2420 έχει την δυνατότητα να επικοινωνεί και με άλλα πρωτόκολλα όπως το 802.11b και Bluetooth μέσω ειδικών καρτών που μπορούν να ενσωματωθούν πάνω στην πλατφόρμα αλλά και μέσω των UART/USB υποδοχών. Τέλος χρησιμοποιεί εξωτερικούς αισθητήρες και έχει χρόνο ζωής 6 μήνες με ένα έτος. Κάτι που δείχνει να είναι μειονέκτημα για τη συγκεκριμένη πλατφόρμα σε σχέση με τις όλες παραπάνω. Θα πρέπει να αναφερθεί οτι η Intel φαίνεται να είναι ακόμα σε ερευνητικό επίπεδο σε ότι αφορά τα iMotes αφού δεν είδαμε κάπου στην επίσημη ιστοσελίδα της να τα προωθεί εμπορικά.




Σχήμα 2.13: *Εμπρός όψη ενός iMote. ( Πηγή Εικόνας: Lama Nachman Intel Corporation Research Santa Clara, CA )*

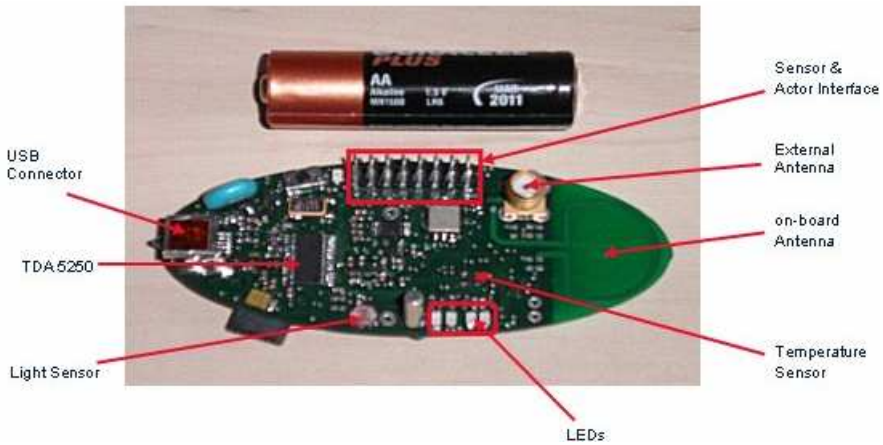



Σχήμα 2.14: *Πίσω όψη ενός iMote. ( Πηγή Εικόνας: Lama Nachman Intel Corporation Research Santa Clara, CA )*

## 2.5 Eyes Project

Το Eyes είναι ένα Ευρωπαϊκό ερευνητικό έργο πάνω στην οργάνωση και συνεργασία ενεργειακά αποδοτικών δικτύων αισθητήρων. Στα πλαίσια αυτού του έργου έχουν αναπτυχθεί δύο πλατφόρμες που βασίζονται στον μικροελεγκτή TI MSP430, τον οποίο περιγράψαμε παραπάνω αφού χρησιμοποιείται και από την πλατφόρμα Telos-B.

- Η πρώτη ονομάζεται EyesIFX και αναπτύχθηκε απο την εταιρεία Infineon[**?**]. Χρησιμοποιεί τον TDA5250 ραδιοπομποδέκτη της ίδιας εταιρείας, ο οποίος μεταδίδει σε συχνότητα 868Mhz μέχρι 64kbps και υποστηρίζει διαμόρφωση κατά πλάτος και κατά συχνότητα. Τέλος έχει ενσωματωμένους αισθητήρες θερμοκρασίας και φωτός.




Σχήμα 2.15: *Φωτογραφία ενός EyesIFX.*

- Η δεύτερη πλατφόρμα που αναπτύχθηκε στα πλαίσια του έργου αυτού είναι η EyesNEDAP, η οποία είναι παρόμοια με την αρχική Mica της crossbow. Αναπτύχθηκε απο την εταιρεία NEDAP και χρησιμοποιεί τον RFM TR1001 ραδιοπομποδέκτη της RF Monolithics[**?**], ο οποίος μεταδίδει σε συχνότητα 868Mhz μέχρι 64kbps και υποστηρίζει διαμόρφωση κατά συχνότητα. Λόγω της επιλογής του TR1001 η πλατφόρμα αυτή, παρα την ταχύτητα της στην επικοινωνία(115,2kbps), παρουσιάζει αυξημένο θόρυβο και είναι ευαίσθητη σε παρεμβολές. Γι' αυτό η EyesIFX αποτελεί καλύτερη επιλογή για απαιτητικά περιβάλλοντα εργασίας. Τέλος πρέπει να αναφέρουμε ότι έχει βγει και μία δεύτερη έκδοση της EyesIFX, η EyesIFXv2

## 2.6 Το IEEE 802.15.4 Πρότυπο και οι ZigBee Προδιαγραφές

Πριν αναφέρθούμε στη σύγκριση των παραπάνω πλατφορμών θεωρούμε απαραίτητο να αναφέρουμε κάποια πράγματα για το πρότυπο IEEE 802.15.4 και τις ZigBee προδιαγραφές, καθώς φαίνεται ότι θα διαδραματίσουν σπουδαίο ρόλο στην μεπέπειτα πορεία των ασύρματων δικτύων αισθητήρων. Το IEEE 802.15.4 πρότυπο καθορίζει αφενός τα πρωτόκολλα του φυσικού και του MAC επιπέδου για περισσότερο έλεγχο και απομακρυσμένη παρακολούθηση και αφετέρου τις εφαρμογές των ασύρματων δικτύων αισθητήρων. Η ZigBee είναι μία εμπορική συνεργασία με στόχο την προώθηση του IEEE 802.15.4 προτύπου. Η ZigBee διασφαλίζει συμβατότητα ορίζοντας υψηλότερα δικτυακά επίπεδα και διεπαφές εφαρμογών. Τα χαμηλού κόστους και ενέργειας χαρακτηριστικά του IEEE 802.15.4 πρόκειται να καταστήσουν δυνατό, ευρείας διασποράς ασύρματα δίκτυα να μπορούν να τρέχουν για χρόνια με μία τυπική εφαρμογή παρακολούθησης. Το πρότυπο αυτό είναι βελτιστοποιημένο για χαμηλού ρυθμού μεταδοσή πληροφορίας(115.1Kb/s), με απλή ή καθόλου υποστήριξη ποιότητας υπηρεσιών(QoS). Υποστηρίζονται τοπολογίες αστεριού αλλά και peer-to-peer. Αντίθετα με το S-MAC, το IEEE 802.15.4 οφείλει τη αποτελεσματικότητα του τόσο στο φυσικό όσο και στο MAC επίπεδο. Ό χρόνος λειτουργίας σε ένα 802.15.4 δίκτυο αναμένεται να είναι μόνο περίπου 1%, αποφέροντας έτσι πολύ μικρή μέση κατανάλωση ενέργειας. Παρολ' αυτά η διατήρηση ενός τόσο μικρού χρόνου λειτουργίας εξαρτάται και από τα πρωτόκολλα υψηλότερου επιπέδου.

## 2.7 Σύγκριση Επιλεγμένων Πλατφορμών

Στην παρούσα ενότητα θα παρουσιάσουμε ένα συγκριτικό πίνακα που θα παρουσιάζει τις διαφορετικές τιμές μεταξύ των πλατφορμών Mica, Mica2Dot, Mica2, Micaz, Tmote sky, Telos-B για κάποια χαρακτηριστικά γνωρίσματα τους. Παρακάτω φαίνεται ο συγκεκριμένος πίνακας:

| | Mica (Atmega 103L) | Mica2Dot | Mica2 | Micaz | Tmote sky | Telos-B |
|---|---|---|---|---|---|---|
| **Processor** | 8-bit, 4Mhz | 8-bit, 4Mhz | 8-bit, 7.37Mhz | 8-bit, 7.37Mhz | 16-bit, 8Mhz | 16-bit, 8Mhz |
| **RAM(Kb)** | 4 | 4 | 4 | 4 | 10 | 10 |
| **ROM(Kb)** | 128 | 128 | 128 | 128 | 48 | 48 |
| **Flash Memory(Kb)** | 4Mbit | 512 | 512 | 512 | 1024 | 1024 |
| **Data Rate(Kbps)** | 40 | 38.4Kbaud (Manchester encoded) | 38.4Kbaud (Manchester encoded) | 250 | 250 | 250 |
| **IEEE 802.15.4 /ZigBee** | No/No | No/No | No/No | Yes/Yes | Yes/No | Yes/Yes |
| **Radio(Mhz)** | 433 | 433(Europe) | 433(Europe) | 2400-2483.5 | 2400-2483 | 2400-2483.5 |
| **Maximum Range(m)** | 30 | 305 | 305 | 100 | 125 | 100 |
| **Wakeup Time(μs)** | 180 | 180 | 180 | 6 | 6 | 6 |

Σχήμα 2.16: *Συκριτικός πίνακας των Mica, Mica2Dot, Mica2, Micaz, Tmote sky, Telos-B πλατφορμών.*

Πρέπει να αναφέρουμε ότι η επιλογή των συγκεκριμένων πλατφορμών έγινε με βάση την διαθεσιμότητα τους στην αγορά. Είναι οι μόνες, που υπάρχουν στην αγορά τη συγκεκριμένη στιγμή. Η μόνη, που έχει αποσυρθεί είναι η Mica και την συμπεριλάβαμε επείδη οι μετρήσεις που περιγράφουμε σε παρακάτω κεφάλαιο έγιναν με αυτές. Στο επόμενο κεφάλαιο περιγράφουμε λεπτομερέστερα τα χαρακτηριστικά της.
Όσον αφορά τη σύγκριση μπορούμε να δούμε ότι οι πλατφόρμες που κυριαρχούν με βάση τα χαρακτηριστικά τους είναι οι Tmote sky και η Telos-B. Ακολουθεί η Micaz και έπειτα οι Mica2, Mica2Dot, Mica. Παρατηρούμε ότι οι Mica2 και Mica2Dot δεν έχουν διαφορές και όπως είδαμε και στις παραπάνω περιγραφές τους είναι παρόμοιες και συμβατές μεταξύ τους. Κάτι που ίσως να τους δίνει ένα ελαφρή προβάδισμα για κάποιες εφαρμογές. Η ειδοποιός διαφορά τους, έγκειται στο μέγεθος τους. Συνεπώς το κριτήριο για να επιλέξει κάποιος μία από τις δύο είναι οι ανάγκες της εφαρμογής που θέλει να πλαισιώσει με τέτοιες συσκευές. Εντύπωση κάνει η μεγάλη διαφορά της εμβέλειας των Mica2 και Mica2Dot σε σχέση με τις υπόλοιπες. Βλέπουμε επίσης ότι μόνο οι τρεις τελευταίες πλατφόρμες υποστηρίζουν το πρότυπο IEEE 802.15.4 και από αυτές μόνο όσες ανήκουν στην Crossbow(Micaz, Telos-B) υποστηρίζουν τις ZigBee προδιαγραφές. Η υιοθέτηση των παραπάνω προτύπων κάθως και η επιλογή καλύτερου ραδιοπομπού(Chipcon CC2420) είναι και η αιτία που οι συγκεκριμένες πλατφόρμες παρουσιάζουν αφενός μεγαλύτερους ρυθμούς μεταφοράς από τις υπόλοιπες και αφετέρου καλύτερους χρόνους μετάβασης από την ενεργή στην ανενεργή κατάσταση(wakeup time). Ώς αφορά τέλος τον μικροελεγκτή βλέπουμε ότι οι δύο τελευταίες πλατφόρμες υπερέχουν σε σχέση με τις υπόλοιπες αν εξαιρέσουμε τη μνήμη ROM.

# Κεφάλαιο 3

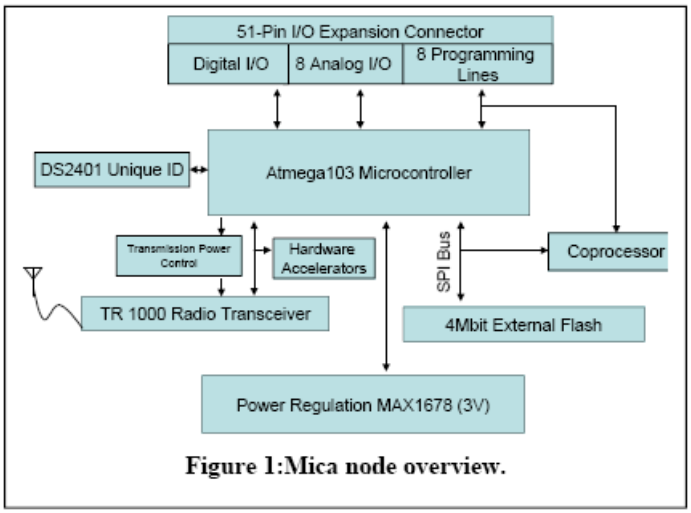
# Αρχιτεκτονική Mica Motes

## 3.1 Εισαγωγή

Στα πλαίσια της διπλωματικής αυτής είχαμε την ευκαιρία να ασχοληθούμε εκτενώς και να κατανοήσουμε τη λειτουργία των Mica motes, της Crossbow. Το Mica mote είναι δεύτερης γενιάς mote, που χρησιμοποιήθηκε σε πολλές προσπάθειες ανάπτυξης εφαρμογών καθώς και στην έρευνα. Πριν από αυτό είχαν αναπτυχθεί τέσσερεις πλατφόρμες τύπου Berkley Motes:

1. COTS dust prototypes, από τον Seth Hollar
2. weC Mote
3. Rene Mote, από την Crossbow
4. Dot mote, που χρησιμοποιήθηκε στο IDF( Intel Developers Forum)[**?**] και έχει τα ίδια χαρακτηρισιτκά με το Rene2 mote

Η αρχιτεκτονική των Mica motes αποτελείται από πέντε βασικά μέρη: υπολογισμοί, RF επικοινωνία, διαχείρηση ενέργειας, επέκταση υποδοχών Ειδόδου/Εξοδου, και δευτερεύουσα μνήμη. Παρακάτω φαίνεται η αρχιτεκτονική ενός Mica mote.




Σχήμα 3.1: *Αρχιτεκτονική ενός Mica mote. ( Πηγή Εικόνας: A wireless embedded sensor architecture for system-level optimization, Jason Hill and David Culler jhill, culler@cs.berkeley.edu )*

## 3.2 Επεξεργαστής

Η μονάδα του μικροελεγκτή είναι υπεύθυνη για τον έλεγχο των αισθητήρων και την εκτέλεση επικοινωνιακών πρωτοκόλλων και υπολογιστικών αλγορίθμων στις συλλεγμένες πληροφορίες από τους αισθητήρες. Στο παραπάνω σχήμα ο κύριος μικροελεγκτής είναι ένας Atmega103L, που τρέχει στα 4Mhz[**?**, **?**, **?**].Παρόλ' αυτά τα μετέπειτα μοντέλα χρησιμοποιούσαν τον νεότερο ATmega128L. Έμεις εδώ θα περιγράψουμε το μοντέλο με τον Atmega103L καθώς με αυτόν πειραματιστήκαμε. Ο Atmega103L είναι ένας χαμηλής ισχύος CMOS 8-bit μικροελεγκτής. Εκτελώντας απαιτητικούς υπολογισμούς σε ένα κύκλο ρολογιού, ο Atmega103L πετυχαίνει αποτελέσματα που πλησιάζουν το 1 MIPS ανα Mhz, επιτρέποντας έτσι στο σχεδιαστή να βελτιστοποιήσει την κατανάλωση ενέργειας σε σχέση με την υπολογιστική ταχύτητα. Ο AVR πυρήνας συνδυάζει ένα πλούσιο σύνολο εντολών(133) με 32 καταχωτητές γενικού σκοπού. Και οι 32 συνδέονται άμεσα με την Αριθμητική Λογική Μονάδα(ALU), επιτρέποντας 2 ανεξαρτητους καταχωρητές να χρησιμοποιούνται σε μία μόνο εντολή σε ένα κύκλο ρολογιού. Η αρχιτεκτονική αυτή είναι πιο αποδοτική ως προς το θέμα του κώδικα, καθώς πετυχαίνει 10 φορές πιο υψηλές τατχύτητες από ένα κοινό CISC μικροελεγκτή. Ο Atmega103L ενσωματώνει τα παρακάτω χαρακτηριστικά: 128Kbytes επαναπρογραμματιζόμενη flash μνήμη, 4Kbytes EEPROM, 4Kbytes SRAM, 32 γενικού σκοπού γραμμές Εισόδου/Εξόδου, 8 γραμμές εισόδου, 8 γραμμές εξόδου, ένα εσωτερικό κανάλι 8 γραμμών, 32 γενικού σκοπού καταχωρητές, 3 hardware χρονομετρητές, 10-bit ADC(μετατροπέα αναλογικού σε ψηφιακό), και 3 διαφορετικά επιλεγόμενα προγράμματα λογισμικού εξοικονόμησης ενέργειας. Το Idle πρόγραμμα σταματά τον επεξεργαστή επιτρέποντας

την SRAM, τους μετρητές, την σειριακή(SPI(Serial Peripheral Interface)) θύρα και το σύστημα των διακοπών να λειτουργούν. Το Power-down πρόγραμμα αποθηκεύει τα περιεχόμενα των καταχωρητών και απενεργοποιεί όλες τις άλλες λειτουργίες μέχρι να προκύψει κάποια διακοπή από το υλικό. Στο Power-save πρόγραμμα, ο ασύγχρονος χρονομετρητής συνεχίζει να τρέχει, επιτρέποντας στο χρήστη να διατηρεί μία χρονική βάση ενώ το υπόλοιπο σύστημα "κοιμάται". Έχει επίσης μία εξωτερική UART και μία SPI θύρα. Ένας βοηθητικός επεξεργαστής, ο AT90LS2343[?, ?] χρησιμοποιείται για να διαχειριστεί τον ασύρματο επαναπρογραμματισμό. Ο AT90LS2343 είναι ένας 8-pin flash μικροελεγκτής με εσωτερικό σύστημα ρολογιού και 5 pins Εισόδου/Εξόδου γενικού σκοπού. Τέλος για να παρέχεται σε κάθε κόμβο ένα μοναδικό ID, υπάρχει ενσωματωμένος ένας DS2401 silicon σειριακός αριθμός[?, ?].

## 3.3 RF Αναμεταδότης

Το μέρος της ραδιοεπικοινωνίας υλοποιείται με έναν RF Monolithics TR1000[?, ?, ?] αναμεταδότη και ένα σύνολο απο διακριτά components, που χρειάζονται για να διαχειριστούν τις ραδιοεπικοινωνίες. Μπορεί να ελεγχθεί εξωτερικά ώστε να εχούμε μία ακτίνα μετάδοσης, που θα κυμαίνεται από μερικά μέτρα μέχρι περίπου τριάντα μέτρα και θα μπορούμε να έχουμε επικοινωνιακές ταχύτητες μέχρι 40Kbps. Η ισχύς της μετάδοσης ελέγχεται εσωτερικά από το TXMOD pin(στη θέση 8, όπως φαίνεται στο σχήμα του [?, ?, ?]). Δυναμικά μπορούμε να ρυθμίσουμε την ισχύ χρησιμοποιώντας ένα DS1804 ψηφιακό ποτενσιόμετρο. Η διεπαφή της ραδιοεπικοινωνίας δίνει άμεσο έλεγχο στο μεταδιδόμενο σήμα επιτρέποντας τη χρήση αυθαίρετων επικοινωνιακών πρωτοκόλλων. Η συχνότητα επικοινωνίας με την οποία εκπέμπει ο αναμεταδότης είναι 916Mhz. Η συχνότητα αυτή δεν μπορεί να επαναπρογραμματιστεί, όπως στις επόμενες πλατφόρμες της Crossbow.

## 3.4 Αποθήκευση

Συνεχής αποθήκευση πληροφοριών παρέχεται μέσω μίας 4Mbit εξωτερικής μνήμης flash. Είναι ένα Atmel AT45DB041B σειριακό flash μοντέλο. Προορίζεται για την αποθήκευση πληροφοριών συλλεγμένες από τους αισθητήρες αλλά και προγραμμάτων, που πρόκειται να εκτελεστούν στον κεντρικό επεξεργαστή. Για να κρατήσεις ένα πρόγραμμα, που προορίζεται για τον μικροελεγκτή, στη μνήμη πρέπει η μνήμη να είναι μεγαλύτερη από τη 128Kb μνήμη του κεντρικού ελεγκτή. Αυτή η απαίτηση μειώνει την πιθανότητα χρήσης EEPROM ως λύση, διότι γενικά είναι μικρότερες από 32Kb.

## 3.5 Ενέργεια

Το υποσύστημα ενέργειας έχει σχεδιαστεί να κανονικοποιεί την παροχή ηλεκτρικής τάσης στο σύστημα. Ένας Maxim1678 DC-DC[?, ?, ?, ?, ?, ?] μετατροπέας παρέχει μία σταθερή 3.0V τροφοδοσία. Το σύστημα έχει σχεδιαστεί να διαχειρίζεται

μία φθηνή μπαταρία, που παράγει μεταξύ των 3,2V και 2,0V(π.χ. ένα ζευγάρι ΑΑ μπαταρίες). Ο μετατροπέας παίρνει την τάση από τα 0,8V και την αυξάνει στα 3,0V. Έτσι παρέχεται μία "καθαρή" και σταθερή τάση τροφοδοσίας για το υπόλοιπο σύστημα. Επιπρόσθετα, επιτρέπει στο σύστημα να αξιοποιήσει μεγαλύτερο μέρος της ενέργειας της μπαταρίας. Σε μία αλκαλική μπαταρία περισσότερο του 50% της ενέργειας της βρίσκεται κάτω από τα 1,2V. Έτσι χωρίς μετατροπέα αυτή η ενέργεια θα ήταν μη χρησιμοποιήσιμη. Για λειτουργία με πολύ μικρή κατανάλωση ενέργειας, το σύστημα ενέργειας μπορεί να απενεργοποιηθεί επιτρέποντας στο υπόλοιπο σύστημα να παίρνει ενέργεια κατευθείαν από μη διαμορφώσιμη τάση. Σταθερή 3V τάση χρειάζεται μόνο για ράδιο-λειτουργίες, και έτσι μικρότερη τάση μπορεί να χρησιμοποιηθεί όταν ο ράδιο-αναμεταδότης δεν χρησιμοποιείται.

## 3.6 Σύστημα Εισόδου/Εξόδου

Το σύστημα Εισόδου/Εξόδου αποτελείται από μία 51-pin υποδοχή επέκτασης σχεδιασμένη να ταιριάζει με μία ποικιλία από προγραμματιστικές πλατφόρμες και πλατφόρμες αισθητήρων. Η υποδοχή χωρίζεται σε 8 αναλογικές γραμμές, 8 γραμμές ελέγχου ενέργειας, 3 PWM γραμμές, 2 αναλογικές γραμμές σύγκρισεις, 4 εξωτερικές γραμμές διακοπών, ένα 12C δίαυλο, ένα SPI δίαυλο, μία σειριακή θύρα, και μία συλλογή από γραμμές αφιερωμένες στον προγραμματισμό των μικροελεγκτών. Η υποδοχή επέκτασης μπορεί επίσης να χρησιμοποιηθεί για το προγραμματισμό της συσκευής και την επικοινωνία με άλλες συσκευές, όπως ένα PC που χρησιμοποιείται σαν gateway.

## 3.7 Επικοινωνία

Το θεμέλιο στο υποσύστημα επικοινωνίας είναι ο αναμεταδότης TR1000. Είναι απλός ASK(amplitude shift keyng) ραδιο-αναμεταδότης. Παρέχει μόνο τα βασικά αρθρώματα του RF καναλιού. Η στήριξη για όλα τα ανώτερα επίπεδα πρέπει να παρέχεται από πρόσθετα components. Δεν υπάρχει προκαθορισμένη πλαισιοποίηση για πακέτα ή bytes και δεν υπάρχει μέγιστο μήκος μετάδοσης. Ο αναμεταδότης έχει τρία προγράμματα λειτουργίας: transmit(μετάδοση), receive(λήψη), και sleep(ανενεργής). Κατά τη διάρκεια της μετάδοσης, η δυαδική τιμή που βρίσκεται σε ένα pin μεταφοράς συνδέεται κατευθείαν με τον RF ενισχυτή. Η μόνη απαίτηση είναι το εύρος του bit της ψηφιακής κυματομορφής να ξεπερνά ένα ελάχιστο εύρος παλμού και η κωδικοποίηση της πληροφορίας στο κανάλι να "σέβεται" συγκεκριμένες απαιτήσεις DC-ισσοροπιών. Η ενίσχυση του μεταδιδόμενου σήματος είναι αναλογική με αυτή του μεταδιδόμενου pin. Γι' αυτό ουσιαστικά όλες οι όψεις του μεταδιδόμενου σήματος από τον TR1000 τίθενται εξωτερικά. Η απλή διεπαφή του TR1000 κάνει δυνατό τον πειραματισμό με οποιοδήποτε αριθμό πρωτοκόλλων μεταφοράς και MAC. Η διαδικασία λήψης είναι εξίσου εύκολη στον TR1000. Υπάρχει ένα pin λήψης, το οποίο είναι ψηφιακή εκδοχή του σήματος που λαμβάνεται. Απλά όρια στο εύρος του σήματος χρησιμοποιούνται για να καθοριστεί αν ο αναμεταδότης λαμβάνει 1 ή 0. Υπάρχει επίσης εξωτερική

πρόσβαση στο απλό σήμα που έρχεται από τα φίλτρα του αναμεταδότη πρίν την ψηφιοποίηση. Αυτό μπορεί να χρησιμοποιηθεί για να εξάγει την εισερχόμενη πληροφορία ή να καθορίσει τήν ισχύ του μεταδιδόμενου σήματος.

# Κεφάλαιο 4

# TinyOS

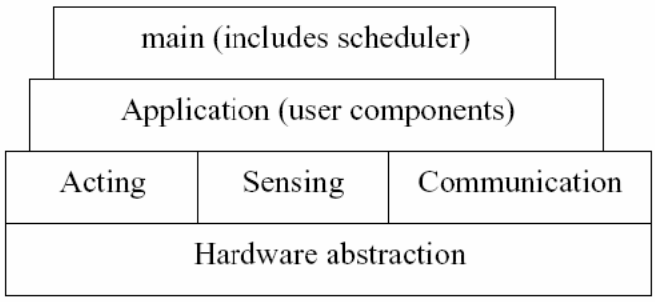
## 4.1 Αρχιτεκτονική Παραδοσιακών Λειτουργικών Συστημάτων

΄Ενας απλοϊκός ορισμός ενός λειτουργικού συστήματος είναι: "Λογισμικό για την αλληλεπίδραση χρήστη και υλικού". Στα παραδοσιακά λειτουργικά συστήματα έχουμε μεγάλες απαιτήσεις όσον αφορά την μνήμη και την αποθήκευση πληροφοριών. Υπάρχουν απαιτητικές και χρονοβόρες λειτουργίες(περίπλοκα συστήματα Εισόδου/Εξόδου) και ως εκ τούτου οι αρχιτεκτονικές αυτές απαιτούν περίπλοκη και πολυδάπανη υποστήριξη απο το υλικό. ΄Ενα τυπικό λειτουργικό σύστημα απεικονίζει το υλικό παρέχοντας ένα σύνολο από υπηρεσίες για εφαρμογές, περιλαμβανομένου του συστήματος διαχείρησης αρχείων, της διαχείρησης μνήμης, του προγραμματισμού εργασιών, των οδηγών για περιφερειακές συσκευές και των λειτουργίων δικτύου.

## 4.2 Εισαγωγή στο TinyOS

Σε αντίθεση με τα παραπάνω παραδοσιακά λειτουργικά συστήματα, στα συστήματα ειδικού σκοπού, όπως τα ασύρματα δίκτυα αισθητήρων, λόγω των ειδικά σχεδιασμένων εφαρμογών και των περιορισμένων πόρων τα λειτουργικά συστήματα τους, γίνονται διάφοροι συμβιβασμοί. Το κυριότερο λειτουργικό σύστημα, που χρησμοποιείται στα ασύρματα δίκτυα αισθητήρων είναι το TinyOS.
Το TinyOS στοχέυει να υποστηρίξει εφαρμογές δικτύων αισθητήρων σε περιορισμένων πόρων πλατφόρμες, όπως τα Berkley motes. Επιλέγει να μην έχει σύστημα διαχείρισης αρχείων, υποστηρίζει μόνο στατική κατανομή μνήμης, υλοποιεί μοντέλο απλής επεξεργασίας των διαδικασιών και όχι παράλληλη ή πολλαπλή επεξεργασία εργασιών και τέλος παρέχει ελάχιστες προγραμματιστικές διεπαφές για τις συσκευές και το δίκτυο.
Το TinyOS είναι ένα λειτουργικό βασισμένο στην έννοια του component, το οποίο είναι κατά κάποιο τρόπο η αφαίρεση ενός λειτουργικού module του συστήματος. Κατ' ουσίαν κάποιος μπορεί να δει το TinyOS σαν ένα ειδικευμένο framework λογισμικού, το οποίο χρησιμοποιείται από μεγάλη κοινότητα λόγω του ανοιχτού κώδικα του. Το πλαίσιο αυτό περιέχει πολλές pre-build εφαρμογές αισθητήρων και αλγόριθμους(π.χ. multi-hop ad-hoc routing) και υποστηρίζει

διάφορες πλατφόρμες αισθητήρων. Πεπειραμένοι προγραμματιστές στη γλώσσα C μπορούν εύκολα να αναπτύξουν TinyOS εφαρμογές αφού γράφονται σε NesC, μία γλώσσα ειδικά για το TinyOS, που είναι βασισμένη στην C.
Η σχεδίαση του TinyOS είναι βασισμένη σε ορισμένα χαρακτηριστικά των δικτύων αισθητήρων : μικρό φυσικό μέγεθος, μικρή κατανάλωση ενέργειας, συγχρονισμένη-εντατική λειτουργία, συνεχή ροή δεδομένων, απουσια ή περιορισμένη χρήση controllers για το υλικό, ποικιλία στο σχεδιασμό και στη χρήση και "σκληρή" λειτουργία για να διευκολύνουν την ανάπτυξη αξιόπιστων κατανεμημένων εφαρμογών. Η πρόθεση των δημιουργών του TinyOS είναι να " διατηρήσουν τους ενεργειακούς, υπολογιστικούς και αποθηκευτικούς περιορισμούς των κόμβων αισθητήρων διαχειριζόμενοι αποτελεσματικά τις hardware δυνατότητες, ενώ παράλληλα να υποστηρίζουν συγχρονισμένη-εντατική λειτουργία με τέτοιο τρόπο που θα πετυχαίνουν αποτελεσματική λειτουργία μέσω modules αλλά και λειτουργία σε αφιλόξενα περιβάλλοντα. Γι' αυτό το TinyOS είναι βελτιστοποιημένο με βάση τη χρήση μνημης και ενεργειακης αποδοτικότητας. Παρέχει καθορισμένες διεπαφές μεταξύ των components, τα οποία εντάσσονται σε γειτονικά επίπεδα-στρώματα. Ένα μοντέλο της διαστρωμάτωσης αυτής φαίνεται στο παρακάτω σχήμα:




Σχήμα 4.1:

Το TinyOS χρησιμοποιεί ένα event-based μοντέλο στη θέση μίας stack-based προσέγγισης με χρήση νημάτων, η οποία θα απαιτούσε περισσότερο χώρο για τις στοίβες και πολυ-εργασιακή υποστήριξη για την εναλλαγή περιεχομένου, ώστε να μπορεί να χειρίζεται υψηλού επιπέδου συγχρονισμό σε ενα μικρό χώρο μνήμης. Οι event-based προσεγγίσεις αποτελούν την καλύτερη λύση για να επιτευχθεί υψηλή απόδοση σε συγρονισμένες εντατικές εφαρμογές. Επιπρόσθετα η event-based προσέγγιση χρησιμοποιεί τους πόρους του επεξεργαστή πιο αποδοτικά και γι' αυτό είναι ιδανική και για το ενεργειακό επίπεδο. Ένα συμβάν εξυπηρετείται από ένα διαχειριστή συμβάντων. Ο διαχειριστής συμβάντων είναι υπεύθυνος για την τοποθέτηση μίας νέας διαδικασίας στον προγραμματιστή εργασιών. Ο προγραμματισμός συμβάντων και εργασιών γίνεται από μία δι-επίπεδη προγραμματιστική

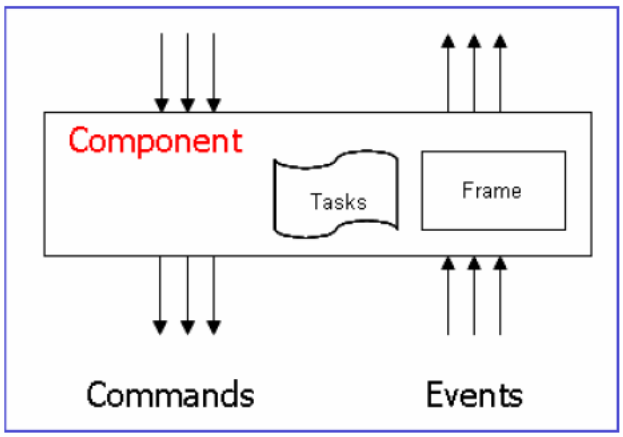
δομή. Αυτού του είδους ο προγραμματισμός παρέχει τη δυνατότητα στα συμβάντα, τα οποία χρησιμοποιούν μικρή ποσότητα της υπολογιστικής ισχύος και δεν απαιτούν πολύ χρόνο για την εκτέλεση τους, να εξυπηρετούνται άμεσα , ενώ οι χρονοβόρες εργασιες να διακόπτονται από τα συμβάντα. Οι διαδικασίες εξυπηρετούνται γρήγορα μεν, αλλά δεν επιτρέπεται blocking ή polling δε.

## 4.3 Σχεδίαση TinyOS

Προκειμένου να επιτύχουμε τα επιθημητά επίπεδα συγχρονισμού, το TinyOS χρησιμοποιεί ένα προγραμματιστικό μοντέλο βασισμένο στα αυτόματα καταστάσεων . Κάνοντας κάθε component ή υπηρεσία αυτόματο καταστάσεων, μπορούμε εύκολα να χειριστούμε πολύ αποδοτικά τη μνήμη και την ισχύ του επεξεργαστή. Αντί να κατανέμουμε για κάθε τρέχουσα εφαρμογή πολλαπλές στοίβες στη μνήμη, μπορούμε να μοιραζόμαστε ένα μοναδικό χώρο εκτέλεσης στη μνήμη μεταξύ πολλαπλών αυτομάτων καταστάσεων. Κάθε component χρησιμοποιεί συμβάντα και εντολές για την μετάβαση μεταξύ καταστάσεων. Αυτές οι μεταβάσεις γίνονται στιγμιαία χωρίς να απαιτούν πολλές υπολογιστικές πράξεις. Κάθε component αποθηκέυεται πρόχειρα στο χώρο εκτέλεσης στη μνήμη για τη διάρκεια αυτής της αλλαγής μεταξύ των καταστάσεων. Στο συγκεκριμένο μοντέλο έχει προστεθεί και η έννοια της διαδικασίας, η οποία επιτρέπει στα components να ζητήσουν το χώρο εκτέλεσης στη μνήμη προκειμένου να εκτελέσουν μεγάλης διάρκειας υπολογισμούς. Αυτές οι διαδικασίες αργότερα προγραμματίζονται για να εκτελεστούν. Καθώς εκτελούνται ατομικά με "σεβασμό" προς τις άλλες διαδικασίες, μπορεί να διακοπούν από συμβάντα μεγαλύτερης προτεραιότητας. Στην παρούσα φάση χρησιμοποιείται μία ουρά FIFO για τον προγραμματισμό, παρολ' αυτά όμως εύκολα μπορεί να προστεθεί και άλλος μηχανισμός.
Ενα δεύτερο πλεονέκτημα της επιλογής αυτού του προγραμματιστικού μοντέλου είναι η διάδοση των αλλαγών στις καταστάσεις του υλικού σε αντίστοιχες του λογισμικού. Για παράδειγμα όπως το αυτόματο καταστάσεων σε επίπεδο υλικού αντιδρά σε αλλαγές του συστήματος Εισόδου/Εξόδου, έτσι και τα components αντιδρούν στα συμβάντα και στις εντολές στις διεπαφές τους.

Το TinyOS αποτελείται από ένα μικρό χρονοπρογραμματιστή και από ένα γράφο components(Σχήμα 4.2).
Τα **Components** ικανοποιουύν την απαίτηση για αρχιτεκτονική με modules. Κάθε component αποτελείται από τέσσερα αλληλοσυσχετιζόμενα μέρη: *ένα χειριστη εντολών, ένα χειριστή συμβάντων, ένα περιεκτικό σταθερού μεγέθους πλαίσιο που κατανέμεται στη μνήμη στατικά, και μία ομάδα απλών εργασιών.* Το πλαίσιο αντιπροσωπέυει την εσωτερική κατάσταση του component. Οι διαδικασίες, οι εντολές και οι χειριστές δουλέυουν στο χώρο της μνήμης που καταλαμβάνει το πλαίσιο και λειτουργούν σύμφωνα με την κατάσταση του πλαισίου. Επιπρόσθετα, το component καθορίζει τις εντόλες που χρησιμοποιεί και τα συμβάντα που ενεργοποιεί. Μέσω αυτού του καθορισμού, συνθέτονται οι γράφοι αρθρωώσεων(modules) των compoonents. Αυτή η διαδικασία σύνθεσης

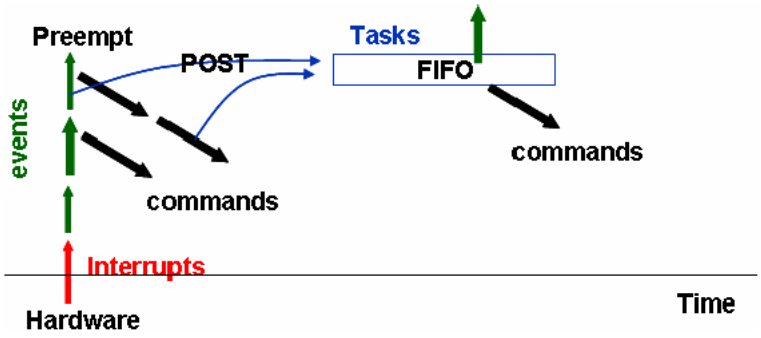
δημιουργεί επίπεδα από components. Μεγαλύτερου επιπέδου components στέλνουν εντολές σε components χαμηλότερων επιπέδων και αυτά με τη σειρά τους ενεργοποιούν συμβάντα σε components υψηλότερου επιπέδου. Για να παρέχουμε ένα αφηρημένο ορισμό της αλληλεπίδρασης δύο components μέσω εντολών και συμβάντων εισάγεται η αμφίδρομη διεπαφή.




Σχήμα 4.2: Δομή Ενός Component

Οι **εντολές** είναι non-blocking αιτήματα που προέρχονται από κατώτερα επίπεδα components. Μία εντολή παρέχει feedback στο component που την καλεί επιστρέφοντας πληροφορίες για την κατάσταση της. Γενικά ο χειριστής εντολών βάζει τις παραμέτρους τις εντολής στο πλαίσιο και στέλνει μία διαδικασία στην ουρά εργασιών για εκτέλεση. Η επιτυχία μίας εντολης μπορεί να φανεί με την ενεργοποίηση ενος συμβάντος. Εντολές ανωτέρου επιπέδου μπορούν να καλέσουν εντολές κατωτέρου επιπέδου.
Οι **χειριστές συμβάντων** ενεργοποιούνται από συμβάντα κατώτερου επιπέδου component, ή όταν συνδέονται απευθείας με το υλικό, μέσω διακοπών(interrupts). Ομοίως με τις εντολές, το πλαίσιο θα μεταβάλλεται και διαδικασίες μπορούν να εισέρχονται στην ουρά. Αμφότερες, οι διαδικασίες και οι εντολές, εκτελούν ένα μικρό σταθερό έργο όμοιο με εκείνο των ρουτινών εξυπηρέτησης των διακοπών. Τα συμβάντα έχουν τη δυνατότητα να καλέσουν εντολές, να ενεργοποιήσουν άλλα συμβάντα, να στείλουν διαδικασίες, να διακόψουν(preempt) διαδικασίες αλλά όχι το αντίστροφο. Αντίθετα οι εντολές δεν μπορούν να ενεργοποιήσουν συμβάντα και τέλος οι διακοπές απο το υλικό ενεργοποιούν συμβάντα του κατώτατου επιπέδου και αποθηκέυουν τις πληροφορίες στο πλάισιο.
Οι **διαδικασίες** εκτελούν το κυριότερο έργο. Είναι ατομικές, τρέχουν μέχρι να τελειώσουν και μπορούν να διακοπούν μόνο από συμβάντα και όχι από άλλες διαδικασίες. Οι διαδικασίες εκτελούν εντατικό υπολογιστικό έργο και χειρίζονται την πολλαπλή ροή δεδομένων. Εισέρχονται στην ουρά ενός FIFO

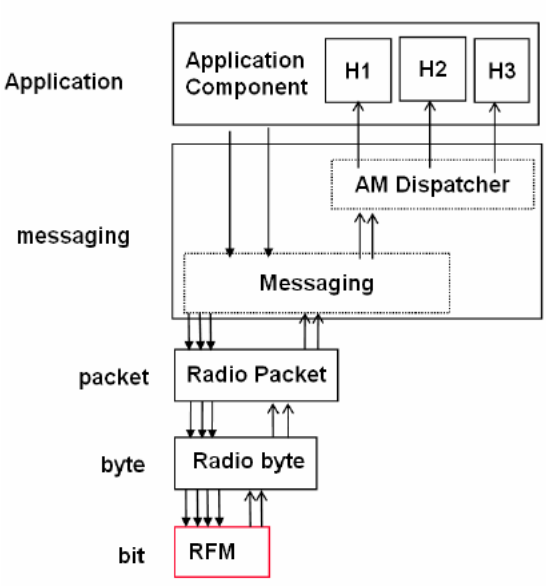
προγραμματιστή εργασιών για να εκτελέσουν μία άμεση επιστροφή ενός συμβάντος ή μίας εντολής που χειρίζεται ρουτίνες. Λόγω του FIFO χρονοπρογραμματισμού, οι διαδικασίες εκτελούνται σειριακά και πρέπει να είναι όσο το δυνατό μικρές. Εναλλακτικά του FIFO χρονοπρογραμματισμού μπορούν να χρησιμοποιηθεί χρονοπρογραμματισμός με βάση την προτεραιότητα ή ενα μέγιστο χρόνο εκτέλεσης που θα παρέχεται σε κάθε διαδικασία. Το TinyOS εξασφαλίζει ότι όταν αδειάσει η ουρά εξυπηρέτησης του χρονοπρογραμματιστή, ο επεξεργαστής θα μεταπηδήσει σε κατάσταση sleep(χαμηλής κατανάλωσης). Τα περιφερειακά κυκλώματα θα συνεχίζουν τη λειτουργία τους, ώστε να μπορούν να αφυπνίσουν ανά πάσα στιγμή τον επεξεργαστή. Ας σημειωθεί ότι όταν αδειάσει η ουρά εξυπηρέτησης, τότε η νέα διαδικασία μπορεί να προστεθεί σε αυτή μόνο μετά την ανίχνευση ενός γεγονότος. Συνεπώς ο χρονοδρομολογητής μπορεί να είναι ανενεργός μέχρι την εμφάνιση κάποιου συμβάντος.




Σχήμα 4.3: Σχηματική αναπαράσταση των παραπάνω

## 4.4 Επικοινωνία

Ο τρόπος επικοινωνίας του TinyOS συμπυκνώνεται στο παρακάτω σχεδιάγραμμα:




Σχήμα 4.4: Αναπαράσταση τρόπου επικοινωνίας στο TinyOS

Παρακάτω θα δούμε κάθε επίπεδο λεπτομερώς:

- **RFM**
  Σε αυτό το επίπεδο μπορούμε να θέσουμε την κατάσταση λειτουργίας(Operation Mode)(εκπομπή ή λήψη) καθώς και το βαθμό της δειγματοληψίας. Επίσης μπορούμε να ειδοποιήσουμε οτι η εκμπομπή ή η λήψη κάποιων δεδομένων έχει τελειώσει καθώς και να σταματήσουμε το RFM.

- **Radio Byte**
  Αυτό το επίπεδο είναι υπεύθυνο για την κατά bit κωδικοποίηση(Manchester),για την ανίχνευση και διόρθωση λαθών, για την ισχύ του σήματος και για την ανίχνευση ελεύθερου καναλιού για μετάδοση(αλλιώς περιμένει για ένα τυχαίο χτύπων ρολογιού).

- **Radio Packet**
  Σε αυτό το επίπεδο υπάρχει 16-bit CRC έλεγχος("πετάει" το πακέτο αν αποτύχει)

- **Messaging**
  Σε αυτο το επίπεδο λαμβάνει χώρα η επεξεργασία των πακέτων(ένωση,διαίρεση

σε μικρότερα πακέτα),η διευθυνσιοδότηση καθώς και η υποστήριψη ειδικών διευθύνσεων(broadcast ή UART)

- **AM dispatcher**
  Είναι μεταβλητή ενός byte που είναι ενσωματωμένη στο μήνυμα και χρησιμοποιείται για να κατευθύνει τα πακέτα στους χειριστές.Το επίπεδο αυτό ακολουθεί την παρακάτω υλοποίηση:

  ```
  if(msg.type == 0) val = Handler0(data);
  if(msg.type == 1) val = Handler1(data);
  ...
  if(msg.type == 255) val = Handler255(data);
  ```

- **Application**
  Σε αυτο το επίπεδο γίνεται διευθυνσιοδότηση με βάση το περιεχόμενο του πακέτου, υπηρεσίες εντοπισμού και επεξεργασία των δεδομένων από τους αισθητήρες.

## 4.5 Active Message

Στο TinyOS δεν μπορούν να χρησιμοποιηθούν παραδοσιακοί τροποί επικοινωνίας(TCP/IP, sockets, πρωτόκολλα διευθυνσιοδότησης κ.α.) διότι με αυτά απαιτείται μεγάλη χρήση της μνήμης(ενδιάμεση αποθήκευση, χρήση νημάτων) αφού χρησιμοποιούν μεγάλα bandwidth και πολιτικές "stop and wait". Παράλληλα στο TCP/IP(με χρήση sockets) τα δεδομένα αποθηκέυονται σε μία στοίβα του δικτύου μέχρι το νήμα της εφαρμογής τα διαβάσει και όταν δεν υπάρχει διαθέσιμη πληροφορία το αντίστοιχο νήμα της εφαρμογής αναβάλει κάθε άλλη εργασία του(blocking).

Όμως στα ασύρματα δίκτυα αισθητήρων υπάρχει ο περιορισμός των υπολογισμών σε πραγματικό χρόνο αλλά και ο "φθηνός" υπολογισμός σε πόρους.

Γι' αυτό δημιουργήθηκε το επίπεδο Active Message, το οποίο είναι υπέυθυνο για:

- Ταυτόχρονη επικοινωνία και υπολογισμούς
- Εναρμόνιση αρχών επικοινωνίας με δυνατότητες υλικού
- Παροχή ενός κατανεμημένου μοντέλου με βάση τα συμβάντα, στο οποίο κάθε δικτυακός κόμβος στέλνει συμβάντα στους άλλους κόμβους
- Εναρμόνιση του μοντέλου με βάση τα συμβάντα του TinyOS

Τα μηνύματα υποστηρίζουν ένα επίπεδο, στο οποίο κατά την άφιξη θα ενεργοποιείται ένας διαχειριστής και το μέγεθος της πληροφορίας λαμβάνεται σαν όρισμα. Οι διαχειριστές εκτελούνται γρήγορα για να αποτρέψουν τη δικτυακή συμφόρηση

και να παρέχουν επαρκή απόδοση. Με το επίπεδο αυτό κάθε μήνυμα περιέχει το όνομα ενός διαχειριστή συμβάντων. Ο αποστολέας δηλώνει τον αποθηκευτικό χώρο που θα χρειαστεί στο πλαίσιο, έναν διαχειριστή, κάνει αίτηση για αποστολή και εκπέμπει ανταγωνιστίκο σήμα. Από την άλλη πλευρά ο διαχειριστής συμβάντων του παραλήπτη ενεργοποιείται αυτόματα στον κόμβο-στόχο. Έτσι δεν έχουμε κανένα νήμα να περιμένει ή να είναι blocked στον παραλήπτη και επίσης έχουμε μία μόνο αποθήκευση.

## 4.6 NesC

Η NesC είναι μία επέκταση της C σχεδιασμένη για να ενσωματώσει τις δομικές αρχές και το μοντέλο λειτουργίας του TinyOS. Χρησιμοποιεί ένα μεταφραστή απο τον οποίο πήρε το όνομα της. Είναι μία χαμηλού επιπέδου, αντικειμενοστροφής γλώσσα που χρησιμοποιείται για τον προγραμματοσμό των κόμβων. Τα κυριότερα χαρακτηριστικά της σύμφωνα με το [?] είναι:

- Διαχωρισμός της κατασκευής και της σύνθεσης: τα προγράμματα αποτελούνται απο components, τα οποία συνθέτονται για να σχηματίσουν ένα ολοκληρωμένο πρόγραμμα. Κάθε component καθορίζει δύο "όψεις"(scopes), η μία περιέχει τις δηλώσεις των συναρτήσεων του component και είναι ενδεικτική της λειτουργικότητας και της χρησιμότητας του αρθρώματος ενώ η άλλη περιέχει τις υλοποιήσεις των συναρτήσεων αυτών. Τα components επικοινωνούν μεταξύ τους και ο τρόπος να γίνει αυτό είναι οι διεπαφές. Η προσπέλαση μπορεί να γίνει είτε με μία διακοπή του υλικού(hardware interrupt) ή από μία απλή διαδικασία(task).

- Περιγραφή τω χαρακτηριστικών ενός component μέσω του συνόλου των διεπαφών. Οι διεπαφές παρέχονται ή χρησιμοποιούνται από τα components. Οι παρεχόμενες διεπαφές προορίζονται να παρουσιάσουν την λειτουργικότητα που παρέχει το component στους χρήστες. Οι διεπαφές που χρησιμοποιούνται παρουσιάζουν τις λειτουργίες που το component χρειάζεται για να εκτελέσει το έργο του.

- Οι διεπαφές είναι αμφίδρομες. Παρέχεται ένα σύνολο από συναρτήσεις που υλοποιούνται από τους παροχείς των διεπαφών(εντολές) και ένα άλλο που οι συναρτήσεις του υλοποιούνται από τους χρήστες των διεπαφών(συμβάντα). Αυτό επιτρέπει σε μία διεπαφή να περιγράφει την αλληλεπίδραση μεταξύ components. Το παραπάνω είναι σημαντικό διότι όλες οι μεγάλες εντολές στο TinyOS είναι απερίσπαστες και η ολοκλήρωση της εργασίας τους ακολουθείται από την αποστολή ενός συμβάντος. Υλοποιώντας τις διεπαφές, ένα component δεν μπορεί να καλέσει μία εντολή αν πρωτα δεν παρέχει μία υλοποίηση του αντίστοιχου συμβάντος. Γενικά οι εντολές έχουν φορά προς τα κάτω, π.χ., από components που ανήκουν στο επίπεδο των εφαρμογών σε components που ανήκουν σε επίπεδα κοντά στο υλικό. Αντίθετα τα συμβάντα έχουν αντίστροφη φορά.

- Τα components "συνδέονται"(linked) στατικά μεταξύ τους μέσω των διεπαφών. Η nesC έχει σχεδιαστεί υπό την προυπόθεση ότι ο κώδικας παράγεται από μεταφραστές, οι οποίοι μεταγλωττίζουν όλο το πρόγραμμα και όχι μόνο ένα μέρος του. Αυτό επιτρέπει καλύτερη παραγωγή και ανάλυση του κώδικα. Παράδειγμα του παραπάνω είναι ο έλεγχος συγχρονισμού(data race) στο πρόγραμμα.

Υπάρχουν μόνο δύο είδη component, τα πρώτα ονομάζονται module και τα δεύτερα configuration. Οι τρεις βασικές έννοιες, οι οποίες κυριαρχούν στη NesC είναι οι module, configuration και interface.

Οι **διεπαφές(interfaces)** χρησιμοποιούνται για να ομαδοποιήσουν λειτουργίες όπως split-phase εργασίες(send, sendDone) ή διεπαφές ελέγχου(init, start, stop). Οι παροχείς διεπαφών πρέπει να υλοποιούν εντολές ενώ οι χρήστες συμβάντα. Τα **modules** υλοποιούν τις προδιαγραφές ενός compoment. ΄Ενα παράδειγμα module φαίνεται παρακάτω:

```
module MyMod {
    provides interface X;
    provides interface Y;
    uses interface Z;
}
implementation
{
    ...// C code
}
```

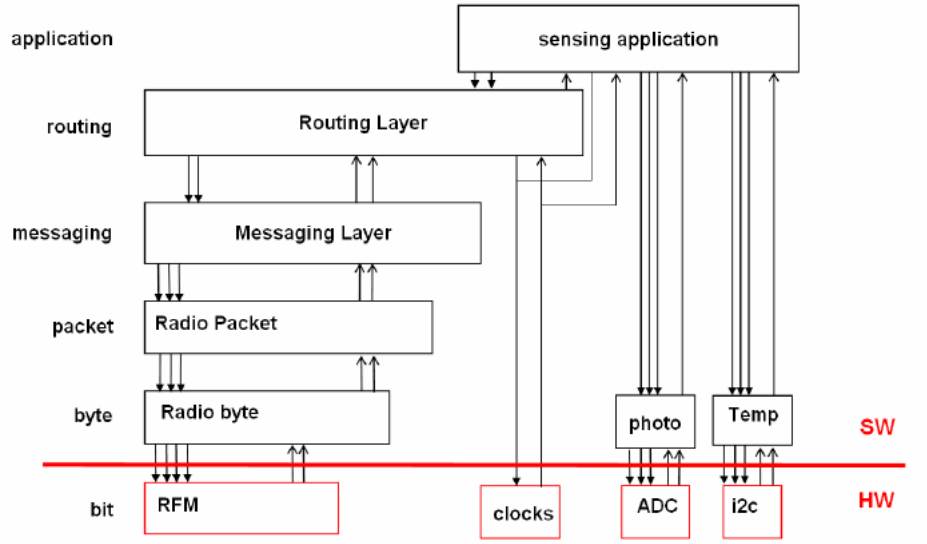
Η λειτουργία των configurations είναι δημιουργία component που αποτελείται από την ένωση(wiring) πολλαπλών components.
΄Ενα παράδειγμα configuration φαίνεται παρακάτω:

```
module MyConfig {
    provides interface X;
    provides interface Y;
    uses interface Z;
}
implementation
{
    ...// wiring code
}
```

Εκτεταμένη ανάλυση παραδείγματος μπορεί να βρεθεί στα στα [**?**] και [**?**] .

## 4.7 TinyOS Εφαρμογές

Στο σχήμα 4.5 βλέπουμε ένα σχεδιάγραμμα μίας ολοκληρωμένης εφαρμογής. Τα κατώτερα επίπεδα των compoments αντιστοιχούν κατευθείαν στο υλικό του συστήματος. Το φυσικό επίπεδο του υλικού αντιστοιχεί στο λογισμικό μοντέλο του component. Η εφαρμογή του χρήστη βρίσκεται στην κορυφή της ιεραρχίας δίνοντας εντολές στα κατώτερα επίπεδα των compoment και αντιδρώντας σε ενεργοποιήσεις συμβάντων από τα components του συστήματος. Κατά τη διάρκεια της εκτέλεσης, όλα τα συμβάντα ενεργοποιούνται από εκείνα του υλικού με φορά από κάτω προς τα πάνω, όπως φαίνεται στο γράφο. Αυτό πηγάζει από το προγραμματιστικό μοντέλο των αυτόματων καταστάσεων, στο οποίο οι αλλαγές των καταστάσεων είναι αποτέλεσμα των αλλαγών στις εισόδους του υλικού.




Σχήμα 4.5: Αναπαράσταση εφαρμογής στο TinyOS

Για μία περαιτέρω ανάλυση ενός παραδείγματος μίας εφαρμογής στο TinyOS ο αναγνώστης μπορεί να προστρέξει στα [?] και [?] .

# Κεφάλαιο 5

# Πειραματική Αξιολόγηση με Πραγματικό Δίκτυο

Στο παρόν κεφάλαιο θα περιγράψουμε τις πειραματικές μετρήσεις που είχαμε την ευκαιρία να πάρουμε και θα προσπαθήσουμε να ερμηνεύσουμε τα αποτελέσματα τους.

## 5.1 Περιγραφή Πειραμάτων

Τα πειράματα πραγματοποιήθηκαν με πλατφόρμες Mica motes σε ανοιχτό χώρο του πανεπιστημίου. Αρχικός σκοπός μας ήταν η παράταξη όσων κόμβων είχαμε στη διάθεση μας σε μία ευθεία. Στην εικόνα 5.1 φαίνεται η παράταξη των Mica motes, έτσι όπως τη φωτογρασίσαμε. Όπως μπορεί κάποιος να δεί τα motes απέχουν μεταξύ τους απόσταση τέτοια, ώστε η εμβέλεια του καθενός να περιορίζεται στους δύο γείτονες του. Ο σκοπός της τοπολογίας αυτής είναι η όσο το δυνατόν μείωση των συγκρούσεων καθώς και η αποφυγή μετάδοσης ίδιων μηνυμάτων. Για να πετύχουμε όμως αυτή τη μικρή εμβέλεια των motes (αφού όπως αναφέραμε σε προηγούμενο κεφάλαιο η εμβέλεια των Mica κυμαίνεται απο 1-30 μέτρα) και συνεπώς τη διατήρηση του δικτύου μας σε μικρά μεγέθη απόστασης, αυξήσαμε την τιμή του ποτενσιόμετρου που διαθέτει ο πομποδέκτης μέσω της διεπαφής που μας προσφέρει το TinyOS. Παράλληλα έχουμε εξυψώσει τα motes 10-15 εκατοστά από το έδαφος για να έχουμε καλύτερη λήψη και μετάδοση των πληροφοριών. Ο χωρος, στον οποίο πραγματοποιήθηκαν τα πειράματα είναι εξωτερικός χωρίς ιδιαίτερο θόρυβο. Παρόλ' αυτά όμως δεν παύουν να υπάρχουν κάποιες ανεπιθύμητες παραμβολές στην επικοινωνία. Θα πρέπει τέλος να αναφερθεί ότι όλα τα motes είχαν γεμάτες μπαταρίες και έτσι δεν υπήρχε περίπτωση αλλοίωσης των αποτελεσμάτων από ενεργειακό λόγο.

Σχήμα 5.1: *Παράταξη των κόμβων του μικρού ασύρματου δικτύου μας(6 hops-7 κόμβοι.*

## 5.2 Διαδικασία Πειράματος

Η διαδικασία που ακολουθήσαμε για την εκτέλεση των πειραμάτων ήταν η επανάληψη των ίδιων πειραμάτων με διαφορετική κάθε φορά αριθμό motes. Αναλυτικότερα, κάθε φορά είχαμε ένα mote πάνω στην πλατφόρμα προγραμματισμού MIB300CA. Η πλαφόρμα συνδεόταν κατευθείαν με τον υπολογιστή μέσω μίας σειριακής θύρας (Εικόνα 5.2). Από εκεί ένα πρόγραμμα μέτραγε τα μηνύματα που λάμβανε το mote πανω στην πλατφόρμα προγραμματισμού. Στην άλλη άκρη της διάταξης το mote δημιουργούσε ένα μήνυμα, που περιείχε έναν αύξοντα ακέραιο και το έκανε broadcast. Τα ενδιάμεσα motes έκαναν απλό broadcast ότι μήνυμα λάμβαναν. Έτσι τα μηνύματα που δημιουργούσε και έστελνε ο αρχικός κόμβος προορίζονταν να φτάσουν στο mote πανω στην πλατφόρμα προγραμματισμού. Στην πρώτη φάση είχαμε δύο motes, αυτό που δημιουργούσε και έστελνε μηνύματα και αυτο πανω στην πλατφόρμα προγραμματισμού. Δηλαδή είχαν απόσταση 1 hop. Κάθε φορά αυξάναμε τον αριθμό των motes και συνεπώς των hops. Οι μετρήσεις σταμάτησαν στα 8 motes, δηλάδη 7 hops. Σε κάθε διάταξη κάναμε οκτώ μετρήσεις μεταβάλλοντας τον ρυθμό, που δημιουργούνται τα μηνύματα. Οι ρυθμοί, που επιλέξαμε είναι 1 μήνυμα ανά 37, 50, 75, 100, 150, 200, 300, 600 ms αντίστοιχα. Για να είμαστε σίγουροι για τα αποτελέσματα των μετρήσεων το πρόγραμμα εμφάνιζε τα συνολικά μηνύματα που δεχόταν ο κόμβος κάθε λεπτό για επτά συνεχόμενα λεπτά. Δηλαδή παίρναμε για κάθε υποπερίπτωση επτά τιμές, από τις οποίες υπολογίζαμε τελικά τον μέσο όρο.

Σχήμα 5.2: *Το mote πάνω στην πλατφόρμα προγραμματισμού MIB300CA, η οποία συνδέεται σειριακά με το laptop.*

## 5.3 Προβλήματα που αντιμετωπίσαμε κατά τη διάρκεια των μετρήσεων

Αρχικά έπρεπε να αποφασίσουμε το χώρο, στον οποίο θα πραγματοποιούσαμε τα πειράματα μας. Έπρεπε να είναι τοποθεσία χωρις πολλές παρομβολές, οι οποίες θα μπορούσαν να αλλοιώσουν τα αποτελέσματα μας και να μας οδηγήσουν σε λάθος συμπεράσματα. Την προϋπόθεση αυτή κάλυπταν οι εσωτερκοί χώροι ενός κτιρίου, όμως θέλαμε να πάρουμε αποτελέσματα, που θα ανταποκρίνονταν όσο το δυνατόν στις πραγματικές εφαρμογές που καλούνται να καλύψουν τα ασύρματα δίκτυα αισθητήρων. Έτσι αφού βρήκαμε την τοποθεσία, στην οποία θα πραγματοποιούσαμε τις μετρήσεις μας, έπρεπε να βρούμε τη σωστή παράταξη των κόμβων. Η "σωστή" παράταξη για το πείραμα ήταν η παράταξη, στην οποία κόμβοι που απείχαν 2 hops δεν έπρεπε να επικοινωνούν μεταξύ τους. Κάτι που πετύχαμε μέτα από αρκετές δοκιμές και αποτυχίες αφού όπως αποδείχτηκε ο συνδυασμός εξωτερικού χώρου και ασύρματης επικοινωνίας αποφέρει πολλά απρόοπτα αποτελέσματα. Παράλληλα τα πειράματα ήταν αρκετά χρονοβόρα και δεν μπορούσαν όλοι οι συνδιασμοί να καληφθούν μέσα σε μία μέρα. Έτσι έπρεπε να επαναλάβουμε τη διαδικασία της τοποθέτησης της "σωστής" παράταξης εκ νέου την επόμενη μέρα.

## 5.4 Προγράμματα που Χρησιμοποιήθηκαν στις Μετρήσεις

Η υλοποίηση των μετρήσεων έγινε με τη βοήθεια δύο ειδών προγραμμάτων. Το ένα είναι υλοποιημένο σε Java και ήταν υπέυθυνο για τη δειγματοληψία των μηνυμάτων από τον υπολογιστή μέσω του κόμβου πανω στην πλατφόρμα προγραμματισμού, ενώ τα υπόλοιπα ήταν σε NesC και υλοποιήθηκαν για να προγραμματίσουμε τα motes. Αρχικά θα αναλύσουμε τα Java πρόγραμμα. Θα περιγράψουμε τα κύρια κομμάτια του κώδικα αρχίζοντας από το παρακάτω:

```
MyMessageListener(){

    try{
        mote = new MoteIF(PrintStreamMessenger.err);
        mote.registerListener(new IntMsg(), this);
       }
    catch(Exception e){
        System.err.println("couldn't
                        contact serial forwarder");
            }

    }
```

Στο κομμάτι υλοποιούμε την συνάρτηση δημιουργού(instructor function), η οποία συνδέεται με την εφαρμογήν SerialForward, που είναι υλοποιημένη στο TinyOS και παρέχει επικοινωνία μέσω της σειριακής θύρας. Πλέον η εφαρμογή μας "ακούει" τη σειριακή θύρα και μπορεί να λάβει ότι πληροφορία έρχεται.

```
public void messageReceived(int dest_addr,
                                Message msg) {
    if (msg instanceof IntMsg)
    {
        msg_num++;
    }
    else {
        throw new RuntimeException("messageReceived:
                        Got bad message type: "+msg);
    }
}
```

Η συνάρτηση messageReceived καλείται μόλις ληφθεί κάποια πληροφορία από την σειριακή θύρα. Όταν λοιπόν ληφθεί ένα μήνυμα και είναι του τύπου που ορίσαμε παραπάνω(απλό μήνυμα με έναν αύξοντα ακέραιο) αυξάνουμε την μεταβλητή msg_num, που αποθηκεύει το σύνολο των μηνυμάτων.

```
public static void main(String[] args) throws
                            IOException,WriteException
{
    int i;
    new MyMessageListener();
    i=0;
    for( ; ; ){
        long start = System.currentTimeMillis();
        while(System.currentTimeMillis()-start<60000)
            { ; }
        long end = System.currentTimeMillis();
        float total = msg_num/(float)(end - start);
       System.out.println("Messages Received: " + msg_num);
        msg_num=0;
        i++;
        if(i>7){
            System.exit(0);
                }
        }
}
```

Εδώ ορίζουμε τη main, η οποία καλεί την MyMessageListener και εισέρχεται σε ένα ατέρμονο loop. Περιμένει κάθε φορά μέσα στο βρόχο while για ένα λεπτό και εμφανίζει τα συνολικά μηνύματα. Όταν φτάσουμε τις επτά επαναλήψεις, που αναφέραμε παραπάνω, το πρόγραμμα τερματίζει αυτόματα.

Πριν αρχίσουμε να αναφερόμαστε στα NesC προγράμματα, υπενθυμίζουμε ότι κάθε εφαρμογή στο TinyOS αποτελείται συνήθως από δύο ειδών components. Ένα module και ένα configuration. Στην συγκεκριμένη περίπτωση και τα δύο προγράμματα αποτελούνται από ένα module και ένα configuration, ενώ το άλλο έχει μόνο configuration.

Το πρώτο πρόγραμμα είναι αυτό που φορτώνουμε στο mote, που είναι στην προγραμματιστική πλατφόρμα.

```
configuration RfmToSerial { } implementation {
  components Main, RfmToInt, IntToRfm, IntToLeds;

  Main.StdControl -> IntToRfm.StdControl;
  Main.StdControl -> RfmToInt.StdControl;
  Main.StdControl -> IntToLeds.StdControl;
  RfmToInt.IntOutput -> IntToRfm.IntOutput;
  RfmToInt.IntOutput -> IntToLeds.IntOutput;
}
```

Η συγκεκριμένη εφαρμογή δεν παρουσιάζει κάποια δυσκολία, αφού το μόνο που κάνει είναι να συνδέει το component RfmToInt με τα IntToRfm και IntTOLeds. Δηλαδή εμφανίζει στα Leds του, τον ακέραιο του μηνύματος που δέχεται και στέλνει το μήνυμα στη σειριακή θύρα. Για να μπορέσουμε να στείλουμε την πληροφορία στη σειριακή θύρα μέσω του component IntToRfm, πρέπει να αλλάξουμε ένα όρισμα της εντολής send από TOS_BCAST_ADDR(που προορίζεται για broadcast) σε TOS_UART_ADDR στο τοπικό αρχείο IntToRfmM.nc. Το επόμενο πρόγραμμα είναι αυτό που έχουν οι ενδιάμεσοι κόμβοι, οι οποίοι όπως προείπαμε απλώς κάνουν broadcast ότι μήνυμα λαμβάνουν. Το συγκεκριμένο πρόγραμμα αποτελείται από δύο components. Το configuration component φαίνεται παρακάτω:

```
configuration RfmToLedsAndRfmPOT { } implementation {
  components Main, RfmToInt, IntToRfm, IntToLeds,
                            RfmToLedsAndRfmPOTM, PotC;

  Main.StdControl -> IntToRfm.StdControl;
  Main.StdControl -> RfmToInt.StdControl;
  Main.StdControl -> IntToLeds.StdControl;
  Main.StdControl -> RfmToLedsAndRfmPOTM.StdControl;
  RfmToInt.IntOutput -> IntToRfm.IntOutput;
  RfmToInt.IntOutput -> IntToLeds.IntOutput;
  RfmToLedsAndRfmPOTM.Pot -> PotC;
}
```

Όπως θα μπορεί να παρατηρήσει κάποιος η διαφορά με το προηγούμενο component είναι στην τελευταία γραμμή, στην οποία αρχικοποιούμε το component του ποτενσιόμετρου(PotC) για μειώσουμε την εμβέλεια των Mica motes. Στην παραπάνω εφαρμογή δεν μας ενδιαφέρει η εμβέλεια του κόμβου καθώς δεν χρησιμοποιεί τον ραδιοπομπό για να στείλει μηνύματα σε άλλους κόμβους. Παρακάτω παραθέτουμε και το module component:

```
module RfmToLedsAndRfmPOTM {
  provides{
    interface StdControl;
  }
  uses{
```

```
    interface Pot;
  }
} implementation {
  command result_t StdControl.init() {
    call Pot.init(99);
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return SUCCESS;
  }
  command result_t StdControl.stop() {
    return SUCCESS;
  }
 }
```

Στο component αυτό υλοποιούμε τις εντολές της διεπαφής που παρέχουμε,της StdControl. Στην init() αρχικοποιούμε το ποτενσιόμετρο με μέγιστη τιμή, ώστε να έχει την μικρότερη δυνατή εμβέλεια.
Το τελευταίο πρόγραμμα είναι αυτό που τρέχει ο κόμβος, που δημιουργεί και τα μηνύματα. Όπως και το προηγούμενο αποτελείται από δύο components. Το configuration φαίνεται παρακάτω:

```
configuration CntToLedsAndRfmPOT { }
 implementation {
   components Main, Counter, IntToLeds,
                        IntToRfm, TimerC,
                            CntToLedsAndRfmPOTM, PotC;

   Main.StdControl -> Counter.StdControl;
   Main.StdControl -> IntToLeds.StdControl;
   Main.StdControl -> IntToRfm.StdControl;
   Main.StdControl -> TimerC.StdControl;
   Main.StdControl -> CntToLedsAndRfmPOTM.StdControl;
   Counter.Timer -> TimerC.Timer[unique("Timer")];
   Counter.IntOutput -> IntToLeds;
   Counter.IntOutput -> IntToRfm;
   CntToLedsAndRfmPOTM.Pot -> PotC;
   }
```

Εδώ συνδέουμε το component Counter με τα Timer, IntToLeds, IntToRfm, ώστε να δημιουργεί μηνύματα μετά από συγκεκριμένη χρονική περίοδο, να εμφανίζει τον ακέραιο του μηνύματος στα Leds του mote, και να κάνει broadcast το μήνυμα αντίστοιχα. Τη χρονική περίοδο την αλλάζουμε από το τοπικό αρχείο Counter.nc, που υπάρχει στον υποφάκελο του TinyOS.
Παρακάτω παραθέτουμε και το module component της εφαρμογής μας:

```
module CntToLedsAndRfmPOTM {
  provides{
    interface StdControl;
  }
```

```
  uses{
    interface Pot;
  }
} implementation {
  command result_t StdControl.init() {
    call Pot.init(99);
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return SUCCESS;
  }
  command result_t StdControl.stop() {
    return SUCCESS;
  }
 }
```

Όπως βλέπουμε το παραπάνω component είναι ακριβώς ίδιο με το module component της RfmToLedsAndRfmPOT εφαρμογής και επιτελεί τις ίδιες λειτουργιες με το παραπάνω.

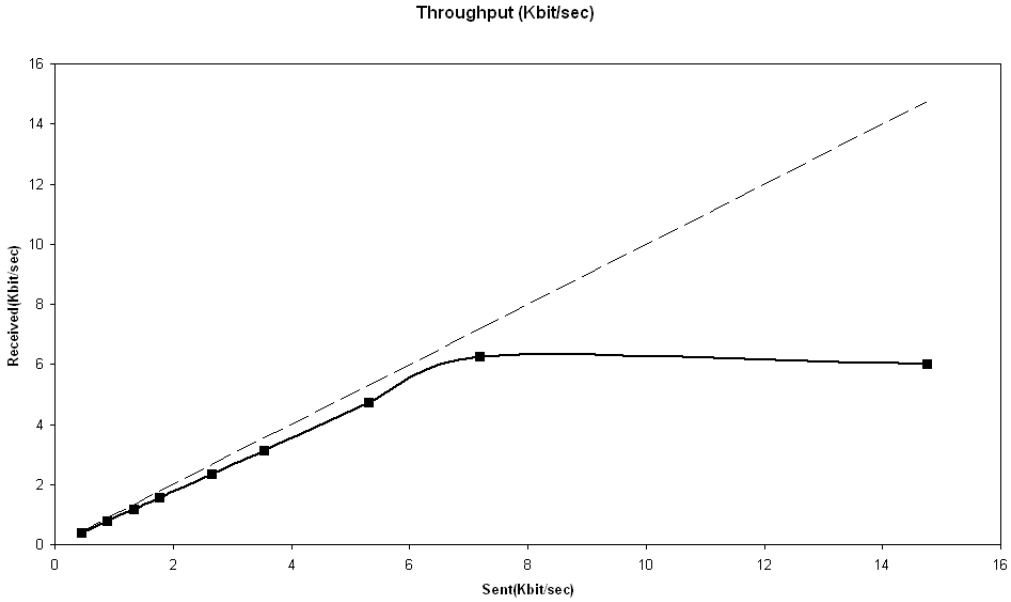
## 5.5 Αποτελέσματα Πειραμάτων

Στο παρόν εδάφιο θα παραθέσουμε τα αποτελέσματα των μετρήσεων. Θα παρουσιάσουμε για κάθε διάταξη ένα πίνακα με τα αποτελέσματα και ένα σχεδιάγραμμα με τα kbit/sec, που έστειλε ο πρώτος κόμβος και τα kbit/sec, που έλαβε ο κόμβος, που συνδεόταν στη σειριακή θύρα. Θα αρχίσουμε με τη διάταξη του 1hop.

| | Msgs | Msgs | | | Kbps | Kbps |
|---|---|---|---|---|---|---|
| Rate (T) | Sent | Received | | | Sent | Received |
| 600 | 100 | 88,14286 | | | 0,442708 | 0,390216 |
| 300 | 200 | 178,2857 | | | 0,885417 | 0,789286 |
| 200 | 300 | 266,1429 | | | 1,328125 | 1,178237 |
| 150 | 400 | 355,2857 | | | 1,770833 | 1,572879 |
| 100 | 600 | 533,2857 | | | 2,65625 | 2,3609 |
| 75 | 800 | 704,4286 | | | 3,541667 | 3,118564 |
| 50 | 1200 | 1066,143 | | | 5,3125 | 4,719903 |
| 37 | 1621,622 | 1412,571 | | | 7,179054 | 6,253571 |
| 18 | 3333,333 | 1358,857 | | | 14,75694 | 6,015774 |

Σχήμα 5.3: *1hop*

Η πρώτη στήλη δείχνει την συχνότητα με την οποία δημιουργούνται μηνύματα(σε ms), η δεύτερη τα μηνύματα που στέλνονται κάθε λεπτό από τον πρώτο κόμβο, η τρίτη τα μηνύματα που λαμβάνονται κάθε λεπτό από τον τελευταίο κόμβο, η έκτη τα Kbps που στέλνονται και πάλι απο τον πρώτο κόμβο και η έβδομη τα Kbps που λαμβάνονται από τον τελευταίο.



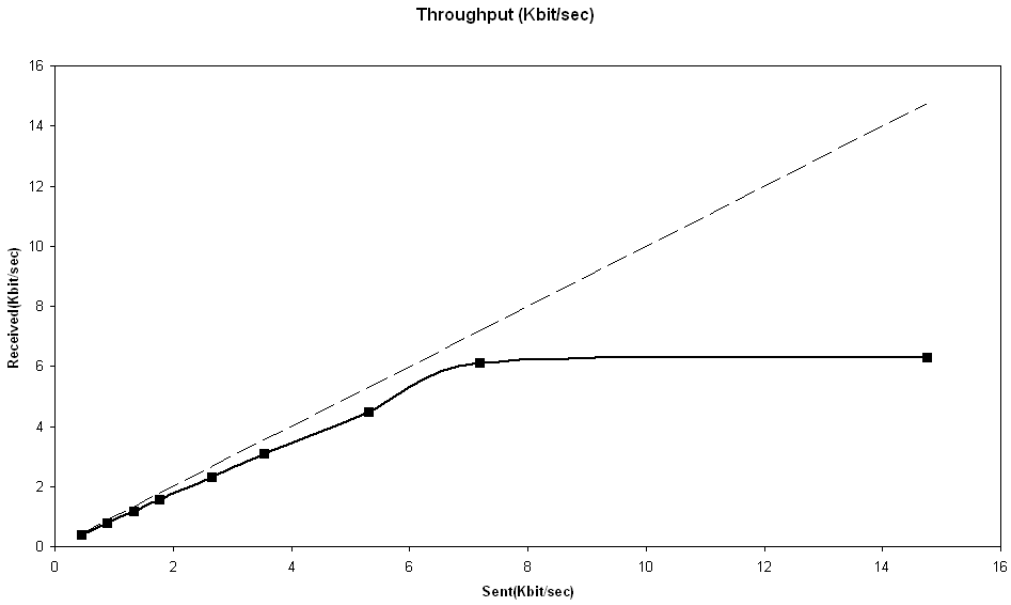

Σχήμα 5.4: *1hop*

Το παραπάνω σχεδιάγραμμα δείχνει τη σχέση μεταξύ των Kbps, που στέλνονται και αυτών που λαμβάνονται σε κάθε διάταξη, ανάλογα με το rate.

Συνεχίζουμε με τη διάταξη των 2hops:

| | Msgs | Msgs | | | Kbps | Kbps |
|---|---|---|---|---|---|---|
| Rate (T) | Sent | Received | | | Sent | Received |
| 600 | 100 | 86,57143 | | | 0,442708 | 0,383259 |
| 300 | 200 | 176,2857 | | | 0,885417 | 0,780432 |
| 200 | 300 | 265,2857 | | | 1,328125 | 1,174442 |
| 150 | 400 | 356,2857 | | | 1,770833 | 1,577307 |
| 100 | 600 | 525,1429 | | | 2,65625 | 2,324851 |
| 75 | 800 | 700,7143 | | | 3,541667 | 3,102121 |
| 50 | 1200 | 1011,286 | | | 5,3125 | 4,477046 |
| 37 | 1621,622 | 1381,286 | | | 7,179054 | 6,115067 |
| 18 | 3333,333 | 1424,143 | | | 14,75694 | 6,304799 |

Σχήμα 5.5: *2hops*

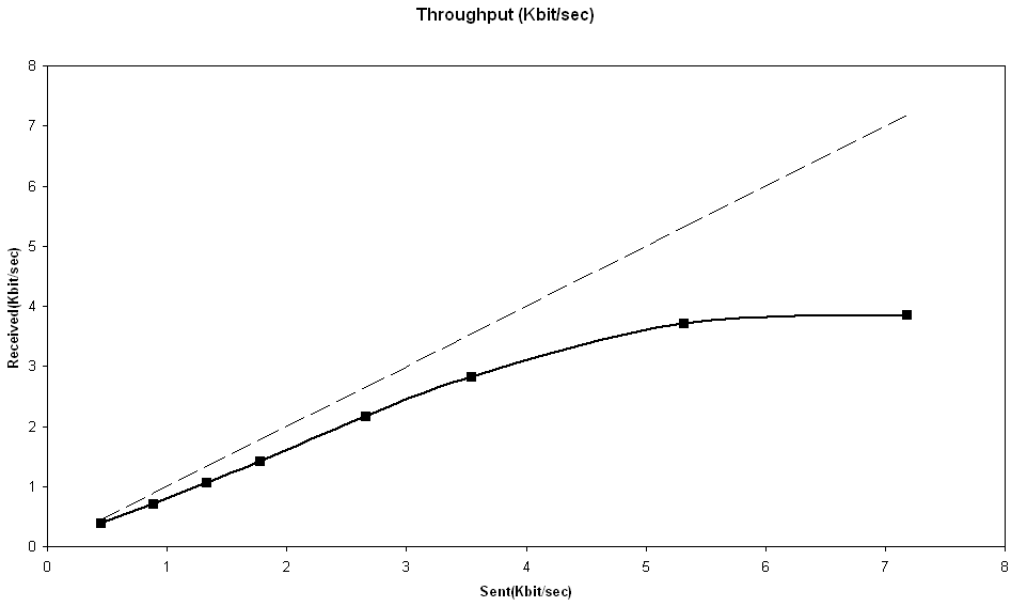



Σχήμα 5.6: *2hops*

΄Επειτα η 3hops:

| | Msgs | Msgs | | | Kbps | Kbps |
|---|---|---|---|---|---|---|
| Rate (T) | Sent | Received | | | Sent | Received |
| 600 | 100 | 88,14286 | | | 0,442708 | 0,390216 |
| 300 | 200 | 162,1429 | | | 0,885417 | 0,71782 |
| 200 | 300 | 240 | | | 1,328125 | 1,0625 |
| 150 | 400 | 320,2857 | | | 1,770833 | 1,417932 |
| 100 | 600 | 491 | | | 2,65625 | 2,173698 |
| 75 | 800 | 637,4286 | | | 3,541667 | 2,821949 |
| 50 | 1200 | 840,4286 | | | 5,3125 | 3,720647 |
| 37 | 1621,622 | 872,2857 | | | 7,179054 | 3,861682 |

Σχήμα 5.7: *3hops*

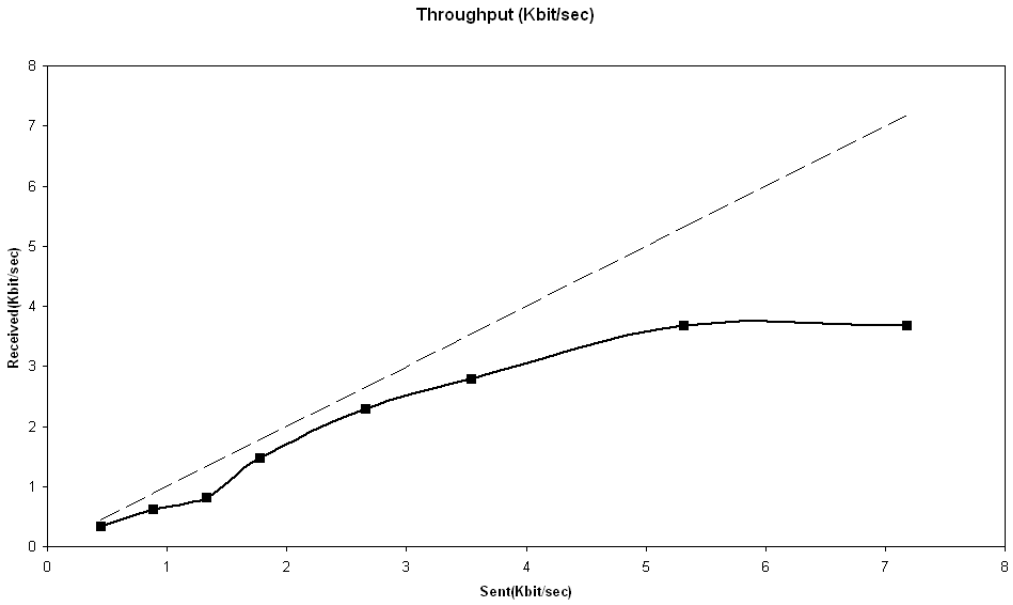



Σχήμα 5.8: *3hops*

Στη συνέχεια η 4hops:

| | Msgs | Msgs | | | Kbps | Kbps |
|---|---|---|---|---|---|---|
| Rate (T) | Sent | Received | | | Sent | Received |
| 600 | 100 | 76,85714 | | | 0,442708 | 0,340253 |
| 300 | 200 | 141,7143 | | | 0,885417 | 0,627381 |
| 200 | 300 | 185,4286 | | | 1,328125 | 0,820908 |
| 150 | 400 | 332,7143 | | | 1,770833 | 1,472954 |
| 100 | 600 | 516,2857 | | | 2,65625 | 2,28564 |
| 75 | 800 | 631,1429 | | | 3,541667 | 2,794122 |
| 50 | 1200 | 830,5714 | | | 5,3125 | 3,677009 |
| 37 | 1621,622 | 833 | | | 7,179054 | 3,68776 |
| | | | | | | |

Σχήμα 5.9: *4hops*

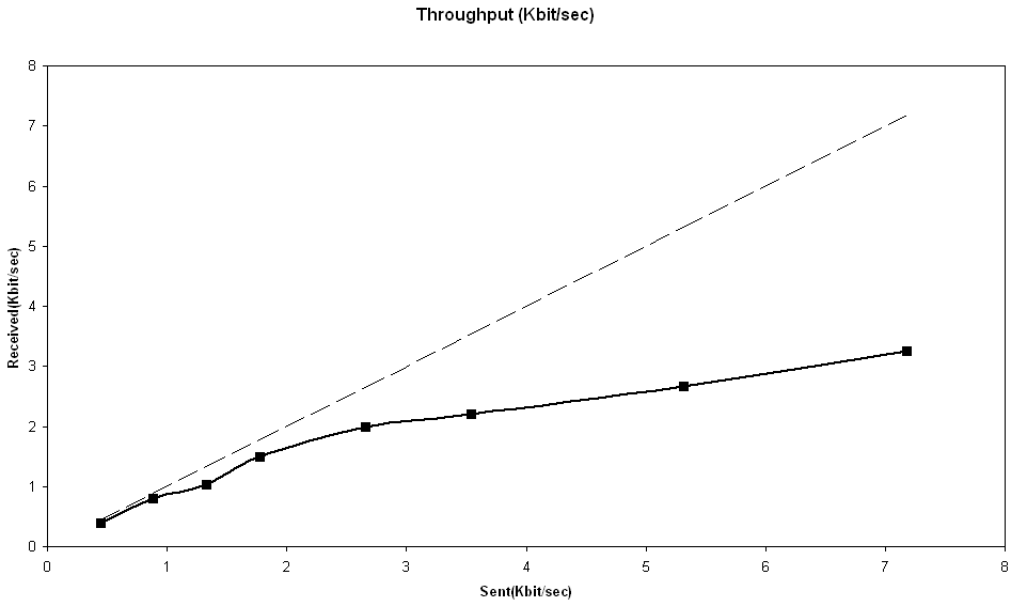



Σχήμα 5.10: *4hops*

Η 5hops:

| | Msgs | Msgs | | | Kbps | Kbps |
|---|---|---|---|---|---|---|
| Rate (T) | Sent | Received | | | Sent | Received |
| 600 | 100 | 87,57143 | | | 0,442708 | 0,387686 |
| 300 | 200 | 180,1429 | | | 0,885417 | 0,797507 |
| 200 | 300 | 234,5714 | | | 1,328125 | 1,038467 |
| 150 | 400 | 339,2857 | | | 1,770833 | 1,502046 |
| 100 | 600 | 450 | | | 2,65625 | 1,992188 |
| 75 | 800 | 497,7143 | | | 3,541667 | 2,203423 |
| 50 | 1200 | 601,375 | | | 5,3125 | 2,662337 |
| 37 | 1621,622 | 734,2857 | | | 7,179054 | 3,250744 |

Σχήμα 5.11: *5hops*

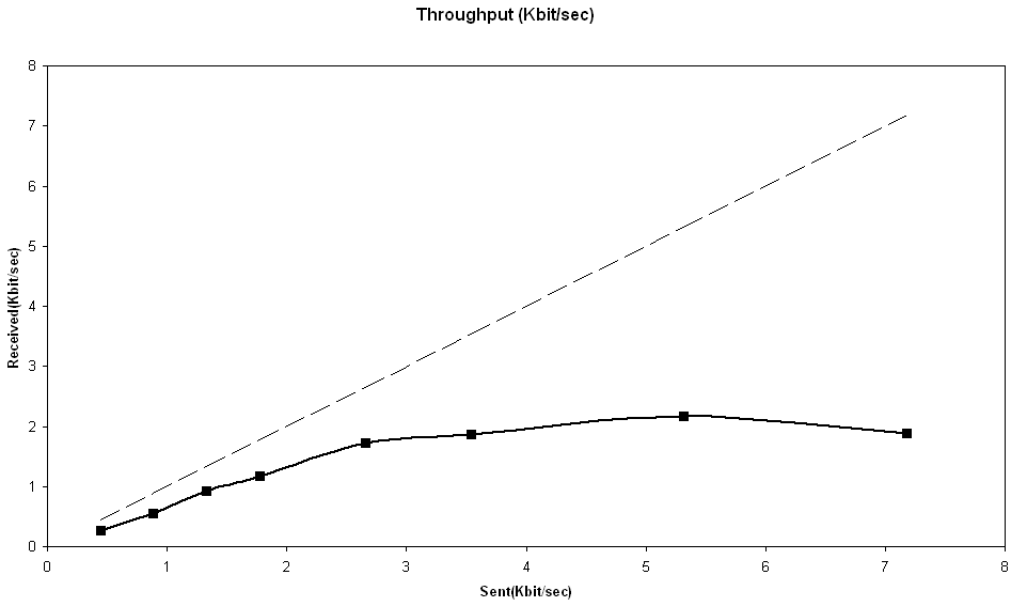



Σχήμα 5.12: *5hops*

Η 6hops:

| | Msgs | Msgs | | | Kbps | Kbps |
|---|---|---|---|---|---|---|
| Rate (T) | Sent | Received | | | Sent | Received |
| 600 | 100 | 60,42857 | | | 0,442708 | 0,267522 |
| 300 | 200 | 123,8571 | | | 0,885417 | 0,548326 |
| 200 | 300 | 210 | | | 1,328125 | 0,929688 |
| 150 | 400 | 265,8571 | | | 1,770833 | 1,176972 |
| 100 | 600 | 388,8571 | | | 2,65625 | 1,721503 |
| 75 | 800 | 420,5714 | | | 3,541667 | 1,861905 |
| 50 | 1200 | 490,4286 | | | 5,3125 | 2,171168 |
| 37 | 1621,622 | 424,4286 | | | 7,179054 | 1,878981 |

Σχήμα 5.13: *6hops*

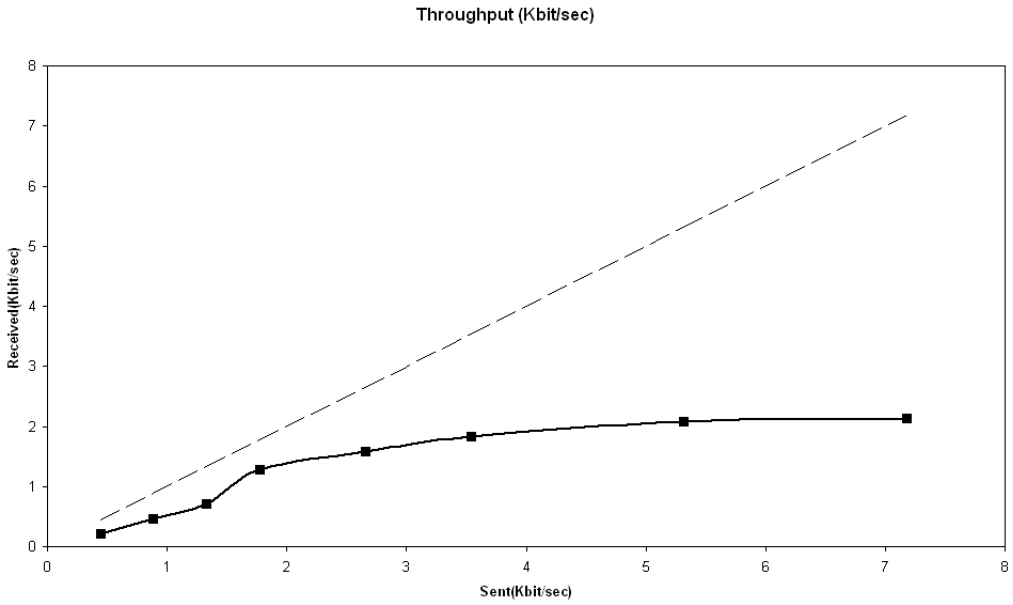



Σχήμα 5.14: *6hops*

Και τέλος η 7hops:

| | Msgs | Msgs | | | Kbps | Kbps |
|---|---|---|---|---|---|---|
| Rate (T) | Sent | Received | | | Sent | Received |
| 600 | 100 | 48,85714 | | | 0,442708 | 0,216295 |
| 300 | 200 | 106,1429 | | | 0,885417 | 0,469903 |
| 200 | 300 | 159,5714 | | | 1,328125 | 0,706436 |
| 150 | 400 | 288,4286 | | | 1,770833 | 1,276897 |
| 100 | 600 | 356,4286 | | | 2,65625 | 1,577939 |
| 75 | 800 | 413 | | | 3,541667 | 1,828385 |
| 50 | 1200 | 470,7143 | | | 5,3125 | 2,083891 |
| 37 | 1621,622 | 482,5714 | | | 7,179054 | 2,136384 |

Σχήμα 5.15: *7hops*




Σχήμα 5.16: *7hops*

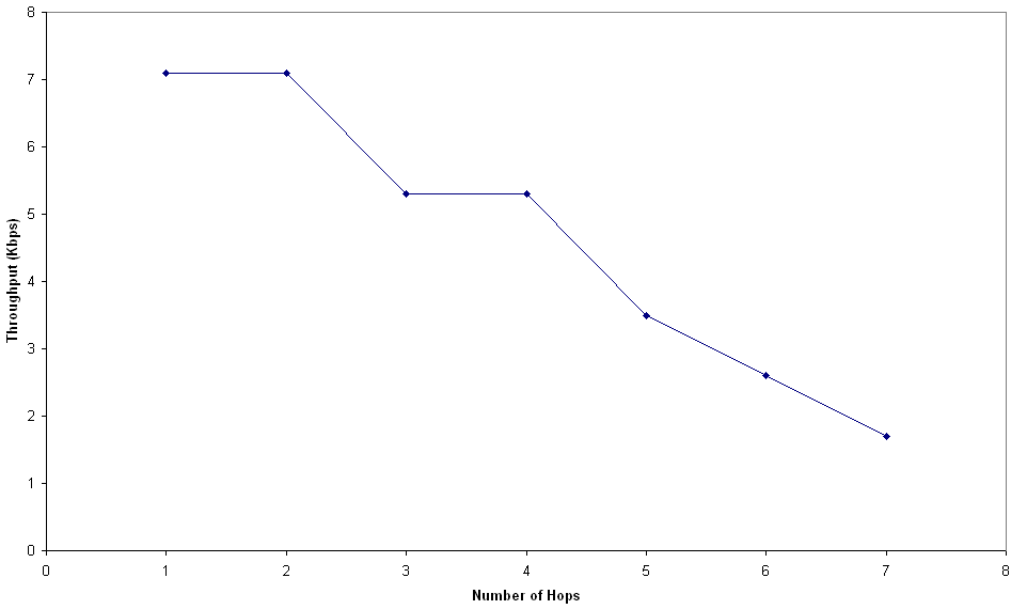
## 5.6 Ανάλυση Αποτελεσμάτων

Αρχικά παρατηρούμε ότι στις δύο πρωτες διατάξεις κατεβάζουμε το rate μέχρι το 18. Αυτό έγινε διότι μέχρι το 37 δεν βλέπαμε κάποια δραματική πτώση στη μεταδιδόμενο όγκο του δικτύου. Στο 18 όμως βλέπουμε ότι έχουμε αρκετή πτώση, η οποία φτάνει την τάξη των 6kbps. Και το γεγονός αυτό συμβαίνει και στη μέτρηση του 1hop και στη μέτρηση των 2hops. Κάτι που σημαίνει ότι ο μέγιστος όγκος που μπορεί να μεταφέρει το δίκτυο γενικά είναι κοντα στα 6kbps. Παρόλαυτά η ικανότητα των Mica motes είναι 40Kbps, σύμφωνα με το manual της Crossbow. Παράλληλα το όριο του B-Mac πρωτοκόλλου που χρησιμοποιούν τα Mica motes είναι γύρω στα 13kbps[**?**]. Μία πιθανή εξήγηση αυτού του φαινομένου είναι η συμβολή του περιβάλλοντος. Δηλαδή δημιουργούνται πολλές παρεμβολές και έτσι χάνονται πολλά μηνύματα, παρά τις προσπαθειες μας για μειωση τους(ανύψωση κόμβων, επιλογή "ήσυχου" εξωτερικού χώρου).
Εκτός από αυτό το φαινόμενο μπορεί κανείς να παρατηρήσει ότι σε κάθε διάταξη υπάρχει ένα σημείο μετά το οποίο η καμπύλη των σχεδιαγραμμάτων που παραθέσαμε παραπάνω τείνουν να γίνουν παράλληλες με τον άξονα των Kbps, που στέλνονται. Στον παρακάτω πίνακα φαίνονται τα σημεία αυτά με τον αριθμό των hops της εκάστοτε διάταξης.

| Hops | Throughput |
|---|---|
| 1 | 7,1 |
| 2 | 7,1 |
| 3 | 5,3 |
| 4 | 5,3 |
| 5 | 3,5 |
| 6 | 2,6 |
| 7 | 1,7 |

Σχήμα 5.17: *Hops - Σημείο Καμπής*

Για να αναδείξουμε καλύτερα αυτή την πτωτική πορεία του μεταδιδόμενου όγκου του δικτύου κάναμε ένα σχεδιάγραμμα με τον αριθμό των hops και το σημείο καμπής του μεταδιδόμενου όγκου. Το σχήμα φαίνεται παρακάτω:




Σχήμα 5.18: *Σχεδιάγραμμα Hops - Σημείο Καμπής*

Όπως μπορούμε να δούμε, όσο αυξάνονται τα hops το σημείο καμπής πέφτει. Αυτό μας δείχνει καθαρά ότι αριθμός των hops επηρεάζει το μεταδιδόμενο όγκο του δικτύου. Κάτι τέτοιο όμως είναι αναμενόμενο αφού αυξάνονται οι μεταφορές και έτσι υπάρχει μεγαλύτερη πιθανότητα να χαθεί ένα μήνυμα.

# Μέρος ΙΙ

# Κεφάλαιο 6

# Εξομοιωτες

## 6.1 Εισαγωγή

Οι εξομοιωτές δικτύου επιχειρούν να μοντελοποιήσουν πραγματικά δίκτυα. Η βασική ιδέα της μοντελοποίησης είναι η δυνατότητα αλλαγής διαφόρων χαρακτηριστικών των μοντελοποιημένων δικτύων και η ανάλυση της επίδρασης στη συμπεριφορά τους. Ιδίως από τη στιγμή που η μοντελοποίηση αυτή είναι φθηνή σε πόρους μπορούν αναπτυχθούν και να αναλυθούν διάφορα σενάρια με μικρό κόστος(σε σχέση με το κόστος των αλλαγών σε πραγματικά δίκτυα). Παρολαυτά οι εξομοιωτές δικτύων δεν είναι τέλειοι. Δεν καταφέρνουν να μοντελοποιήσουν απόλυτα τα δίκτυα. Είναι παρόλαυτά τόσο ακριβείς ώστε να δώσουν στον ενδιαφερόμενο μία αρκετά κατατοπιστική εικόνα για το πώς δουλεύει το δίκτυο ή για το πώς θα επηρεάσουν τη λειτουργία του τυχόν αλλαγές.

Τα ασύρματα δίκτυα αισθητήρων μελλοντικά προορίζονται να επιτελέσουν ρόλο σημαντικών υποσυστημάτων μηχανικών εφαρμογών. Πριν εμπιστευθούμε όμως τόσο σημαντικές λειτουργίες σε τέτοια υποσυστήματα, είναι απαραίτητο να κατανοήσουμε τη δυναμική συμπεριφορά τους, τη αποτελεσματικότητα τους και την αντοχή τους μέσω περιεκτικών και αποτελεσματικών εξομοιώσεων.

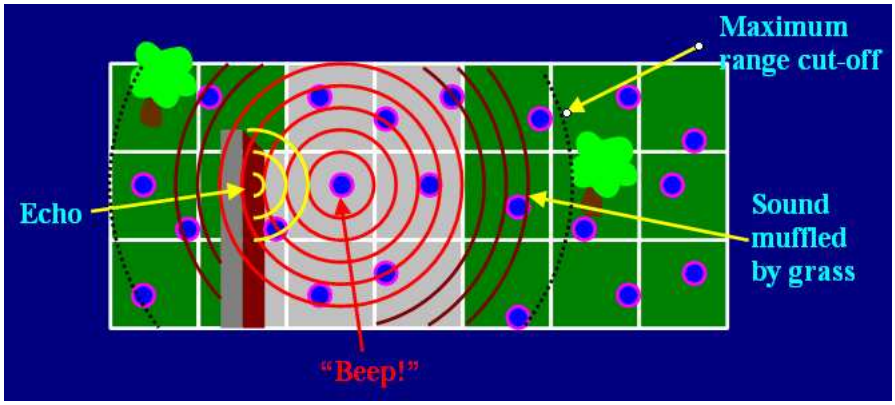
Υπάρχουν πολλοί εξομοιωτές διαθέσιμοι, εμπορικοί ή μη. Εμείς εδώ θα ασχοληθούμε με μερικούς από αυτούς όπως ο SENS, GloMoSim, και ο TOSSIM. Παρακάτω δίνουμε μια περιγραφή αυτών των τριών εξομοιωτών.

## 6.2 SENS

### 6.2.1 Γενικά

Ο SENS είναι ένας εξομοιωτής για ασύρματα δίκτυα αισθητήρων. Η αρχιτεκτονική του βαζίζεται σε modules και επίπεδα, παρέχοντας διαμορφώσιμα components, τα οποία μοντελοποιούν ένα περιβάλλον εφαρμογών, ένα δικτυακής επικοινωνίας και ένα φυσικό(Εικόνα 6.1). Επιλέγοντας κατάλληλες υλοποιήσεις components, οι χρήστες μπορούν να απεικονίσουν μία ποικιλία εφαρμογών-ειδικών σεναρίων, με ακρίβεια και αποτελεσματικότητα πάνω σε μία ανά-κόμβο βάση. Για να επιτύχει ρεαλιστικές εξομοιώσεις, χρησιμοποιεί τιμές απο πραγματικούς

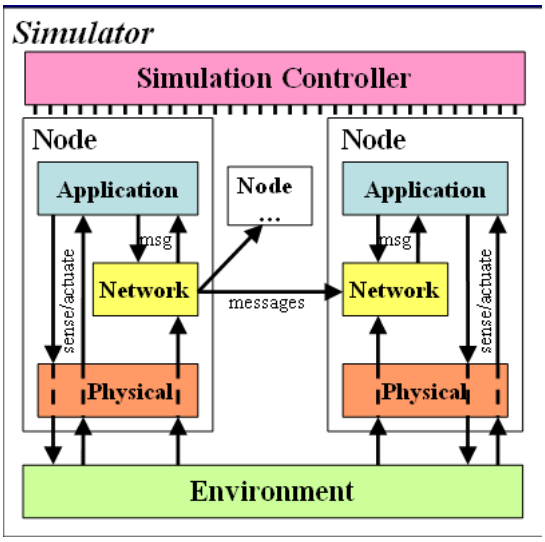
αισθητήρες για να εκφράσουν την συμπεριφορά των υλοποιήσεων των components. Τέτοια συμπεριφορά περιλαμβάνει χαρακτηριστικά ισχύος ηχητικών σημάτων καθώς και ραδιοσημάτων. Παράλληλα ο SENS είναι ανεξάρτητος των πλατφορμών που χρησιμοποιούνται: καθώς νέες πλατφόρμες ασύρματων δικτύων αισθητήρων παρουσιάζονται, μπορούν οι παράμετροι τους να προστεθούν στον εξομοιωτή. Η ικανότητα να αναπτύσσουμε εφαρμογές ανεξάρτητες απο την εκάστοτε πλατφόρμα είναι μία σημαντική πτυχή, αν λάβουμε υπόψιν μας ότι οι πλατφόρμες των ασύρματων δικτύων αισθητήρων συνεχώς εξελίσσονται καθώς υπάρχει συνεχή ανάγκη για καινούριους κόμβους αισθητήρων.
Μία ακόμα αξιόλογη ιδιότητα του SENS είναι ο καινοφανής μηχανισμός για τη μοντελοποίηση του φυσικού περιβάλλοντος. Οι εφαρμογές των ασύρματων δικτύων αισθητήρων χαρακτηρίζονται από την ανάγκη τους για υπολογισμους, και από την επικοινωνία και αλληλεπίδραση με το φυσικό περιβάλλον. Όταν ένας κόμβος ελέγχει τον μηχανισμό κίνησης του μπορεί να επηρεάσει το περιβάλλον και να αλλάξει τα χαρακτηριστικα της διάδοσης στο δίκτυο. Γι' αυτό η αξιοπιστία και η αποδοτικότητα των αποτελεσμάτων μίας εξομοίωσης εξαρτάται σε μεγάλο βαθμό από την ακρίβεια της μοντελοποίησης του γύρω περιβάλλοντος. Για να παρέχει στους χρήστες την ευελιξία της μοντελοποίησης του περιβάλλοντος και της αλληλεπίδρασης του με τις εφαρμογές σε διαφορετικά επίπεδα λεπτομέρειας, ο SENS ορίζει το περιβάλλον σαν ένα πλέγμα εναλλακτικών κομματιών. Επί του παρόντος υλοποιήσεις με τη λογική των κομματιών είναι διαθέσιμες για τσιμέντο,γρασίδι και τοίχους, κάθε μία από τις οποίες έχει διαφορετικά χαρακτηριστικά διάδοσης σήματος. Οι χρήστες μπορούν να ορίσουν και άλλα κομμάτια για να ταιριάζουν με τις ανάγκες τους.




Σχήμα 6.1: Αναπαράσταση της λογικης των κομματιών

### 6.2.2 Δομή του SENS

Ο SENS αποτελείται από διαφορετικούς εξομοιωμένους κόμβους αισθητήρων, που αλληλεπιδρούν με ένα component Environment. Κάθε κόμβος αποτελείται από τρία components, το Application, το Network, και το Physical (Εικόνα 5.2). Κάθε component έχει ένα φανταστικό ρολόι. Έτσι μηνύματα μπορούν να στέλνονται με κάθε καθυστέρηση πέρα απο το φανταστικό χρόνο του αποστολέα. Για παράδειγμα, το Network component κάποιου κόμβου μπορεί να εξομοιώνει την λήψη δύο πακέτων, που συγκρούονται και γι' αυτό δεν παραλαμβάνονται, ενώ τον ίδιο φανταστικό χρόνο το Application component του κόμβου επεξεργάζεται κάποια δεδομένα. Έτσι τα components είναι απομονωμένα και μπορούν να αντικατασταθούν. Ο χρήστης μπορεί να χρησιμοποιεί όποια υλοποίηση component παρέχει ο SENS, να τροποποιήσει ήδη υπάρχοντα components ή να γράψει από την αρχή ένα νεό για ειδικές εφαρμογές, δικτυακά μοντέλα, ικανότητες αισθητήρων ή για περιβάλλοντα.
Ο χρήστης μπορεί να επιλέξει διάφορες υλοποιήσεις των Application, Network, Physical και Environment components. Για παράδειγμα, οι κόμβοι μπορεί να διαμορφωθούν διαφορετικά μεταξύ τους ώστε να σχηματίζουν ένα ετερογενές δίκτυο. Αυτό μπορεί να είναι χρήσιμο όταν διαφορετικοί κόμβοι έχουν διάφορες ιδιότητες(π.χ. στην ιεραρχία), ή για να δοκιμαστούν νέοι κόμβοι σε ένα ήδη υπάρχον δίκτυο.




Σχήμα 6.2: Δομή SENS: Γράφος Components

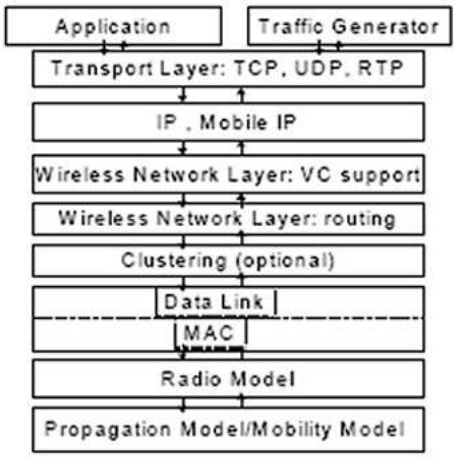
## 6.3 GloMoSim

### 6.3.1 Γενικά

Ο GloMoSim είναι ένας εξομοιωτής για ασύρματα δίκτυα που βασίζεται σε βιβλιοθήκη. Έχει σχεδιαστεί σαν ένα σύνολο από αρθρώματα βιβλιοθηκών, καθένα από τα οποία εξομοιώνει ένα συγκεκριμένο πρωτόκολλο ασύρματης επικοινωνίας στην ιεραρχία των πρωτοκόλλων. Η βιβλιοθήκη έχει αναπτυχθεί με την PARSEC, μία παράλληλη γλώσσα εξομοίωσης βασισμένη στη C. Καινούρια πρωτόκολλα και αρθρώματα μπορούν να προστεθούν στη βιβλιοθήκη χρησιμοποιώντας αυτή τη γλώσσα. Ο GloMoSim έχει σχεδιαστεί να είναι επεκτάσιμος.

### 6.3.2 Αρχιτεκτονική Δικτύου

Η ιεραρχία του δικτύου δομείται από ένα αριθμό επιπέδων όπως φαίνεται στην εικόνα 5.1. Έχουν αναπτυχθεί διάφορα πρωτόκολλα για κάθε επίπεδο και μπορούν να αναπτυχθούν μοντέλα αυτών των πρωτοκόλλων και επιπέδων.




Σχήμα 6.3: Αρχιτεκτονική GloMoSim

Για παράδειγμα, το επίπεδο διάδοσης καναλιού εμπεριέχει α)ένα μοντέλο "ελεύθερου χώρου", το οποίο υπολογίζει την ισχύ του σήματος βασιζόμενο μόνο στην απόσταση μεταξύ κάθε ζεύγους πηγής-δέκτη, β)ένα αναλυτικό μοντέλο, το οποίο υπολογίζει την εξασθένιση του σήματος, γ)ένα μοντέλο καναλιού,το οποίο υπολογιστικά είναι πολύ πιο δαπανηρό αλλά ενσωματώνει τα φαινόμενα πολλαπλού μονοπατιού, shadowing και fading στον υπολογισμό της ισχύς του σήματος. Σαν παράδειγμα εναλλακτικών πρωτοκόλλων, θεωρούμε το Data Link/MAC επίπεδο, για το οποίο έχουν προταθεί διάφορα πρωτόκολλα όπως: CSMA, αποφυγή συγκρούσεων πολλαπλής πρόσβασης, (MACA [**?**]). Καθένα από αυτά έχουν υλοποιηθεί στη βιβλιοθήκη του GloMoSim. Στο επίπεδο του δικτύου έχουν υλοποιηθεί πρωτόκολλα κατακλεισμού(flooding protocols). Σε φάση υλοποίησης βρίσκονται πρωτόκολλα ιεραρχικής διευθυνσιοδότησης για να διαχειριστούν κλιμακωτή δικτυακή διευθυνσιοδότηση. Στο επίπεδο μεταφοράς έχει υλοποιηθεί ένα μοντέλο εξομοίωσης TCP/IP βασισμένο στο Free BSD 2.2.2, το οποίο έχει ενσωματωθεί στην βιβλιοθήκη.

΄Ενα κοινό API για κάθε δύο γειτονικά μοντέλα στη ιεραρχία πρωτοκόλλων προκαθορίζεται για να υποστηρίξει τη σύνθεση τους. Αυτά τα APIs ορίζουν παραμέτρους ανταλλαγής και υπηρεσίες μεταξύ των γειτονικών επιπέδων. Για παράδειγμα, διεπαφές μταξύ του Data LInk/MAC επιπέδου και του επιπέδου δικτύου ορίζονται σαν ανταλλαγή μηνυμάτων που έχουν το ακόλουθο format στη βιβλιοθήκη του εξομοιωτή:

**Διεπαφές Διαχείρισης Πακέτων**:
*Packet_from_NW_to_DLC(P_type, P_dest, P_source, P_payload, P_size, P_VCID)*

*Packet_from_DLC_to_NW(P_type, P_dest, P_source, P_payload, P_size, P_VCID)*, το P_type αναφέρεται στον τύπο του πακέτου(πακέτα πληροφορίας,πακέτα ελέγχου, κ.α.), το P_dest και το P_source αναφέρονται στον αποστολέα και στον λήπτη κόμβο αντίστοιχα, και οι υπόλοιπες παράμετροι χρειάζονται για επεξεργασία των πακέτων και την ύπαρξη ποιότητας υπηρεσιών. Κάθε άρθρωμα πρωτοκόλλου σε ένα συγκεκριμένο επίπεδο απαιτειται να "συμμορφώνεται" με τα APIs που ορίζεται για το επίπεδο αυτό.

### 6.3.3 Σχεδίαση Εξομοιωτή

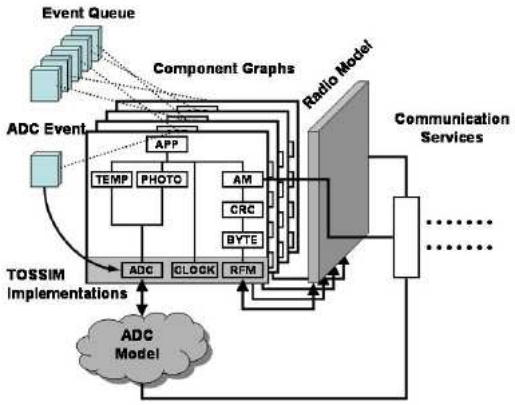
Ο GloMoSim σκοπεύει να αναπτύξει ένα αρθρωματικό(modular) περιβάλλον εξομοίωσης πρωτοκόλλων, που είδαμε παραπάνω, το οποίο θα είναι σε θέση να μοντελοποιήσει δίκτυα με χιλιάδες ετερογενείς κόμβους. Αν όλα τα μοντέλα πρωτοκόλλων υπακούουν στα αυστηρά APIs, που ορίζονται σε κάθε επίπεδο, θα είναι εφικτό να αντικαταστήσουμε μοντέλα πρωτοκόλλων σε ένα συγκεκριμένο επίπεδο(π.χ. να αποτιμήσουμε την επίδραση της χρήσης CSMA πρωτοκόλλου αντι του MACA στο MAC επίπεδο) χωρίς να αλλάξουμε τα μοντέλα στα υπόλοιπα επίπεδα της ιεραρχίας.
΄Οπως αναφέραμε παραπάνω η βιβλιοθήκη του GloMoSim έχει αναπτυχθεί με την PARSEC(απο το **PAR**allel **S**imulation **E**nvironment for **C**omplex Systems),

ένα περιβάλλον εξομοίωσης που προέρχεται από τον εξομοιωτή Maisie[?] . Η PARSEC υιοθετεί μία προσέγγιση βασισμένη στα μηνύματα για την διακριτών-συμβάντων εξομοίωση: οι φυσικές διαδικασίες εξομοιώνονται με αντικείμενα που καλούνται οντότητες, και τα συμβάντα αντιπροσωπεύονται από μεταδόσεις μηνυμάτων, με χρονικούς περιορισμούς και μεταβλητές, μεταξύ των οντοτήτων. ΄Ενα οπτικό-προγραμματιστικό περιβάλλον, που λέγεται PAVE έχει επίσης αναπτυχθεί για να υποστηρίξει την οπτική σχεδίαση των PARSEC προγραμμάτων ή για να διαμορφώσει μοντέλα εξομοίωσης χρησιμοποιώντας προκαθορισμένα components για μία βιβλιοθήκη μίας συγκεκριμένης εφαρμογής, όπως το GloMoSim.
Η ανάγκη της κλιμάκωσης και της χρήσης αρθρωμάτων κάνουν τη σχεδίαση της βιβλιοθήκης ένα ενδιαφέρον αντικείμενο προβληματισμού. Μία ορθόδοξη προσέγγιση είναι η αντιστοίχηση κάθε δικτυακού κόμβου σε ένα αντικείμενο, για παράδειγμα μία PARSEC οντότητα. Παρόλαυτά, προηγούμενη εμπειρία έχει δείξει ότι μεγάλος αριθμός αντικειμένων εξομοίωσης μπορεί να αυξήσουν δραματικά την εξομοίωση, και κάτι τέτοιο δεν είναι σύμφωνο με την έννοια της κλιμάκωσης. Για παράδειγμα για την εξομοίωση δικτύων της τάξης των 100000 κινητών κόμβων πρέπει να χρησιμοπιηθούν τουλάχιστον 100000 οντότητες.
Αντίθετα, ο GloMoSim υποθέτει ότι το δίκτυο διαιρείται σε ένα αριθμό από partitions και μία οντότητα ορίζεται να εξομοιώσει ένα επίπεδο της ιεραρχίας πρωτοκόλλων για όλους τους δικτυακούς κόμβους που ανήκουν στο συγκεκριμένο partition. Οι αλληλεπιδράσεις μεταξύ οντοτήτων πρέπει να υπακούουν τα αντίστοιχα APIs, που αναφέραμε προηγουμένως. Συντακτικώς, οι αλληλεπιδράσεις μπορούν να καθορίζονται χρησιμοποιώντας μηνύματα, κλήση συναρτήσεων, ή παραμέτρους οντοτήτων. Αυτή η μέθοδος υποστηρίζει την ύπαρξη αρθρωμάτων διότι οι PARSEC οντότητες της βιβλιοθήκης, που αντιπροσωπεύουν ένα επίπεδο στην ιεραρχία των πρωτοκόλλων, είναι αυτο-διατηρούμενες. Επίσης συμπυκνώνει την πολυπλοκότητα μίας συγκεκριμένης δικτυακής συμπεριφοράς ανεξάρτητα από άλλες. Παράλληλα αυτή η μέδοθος υποστηρίζει κλιμάκωση διότι η συνάθροιση των κόμβων σε μία οντότητα είναι ικανή να μειώσει τον ολικό αριθμό των οντοτήτων, που έχουν βρεθεί για να βελτιώσουν πολύ περισσότερο την σειριακή εκτέλεση σε σχέση με την παράλληλη.

## 6.4 TOSSIM

Ο TOSSIM είναι ένας διακριτών συμβάντων εξομοιωτής για δίκτυα αισθητήρων που χρησιμοποιούν και υποστηρίζονται από το TinyOS. Αντί να μεταγλωττίσουν μία TinyOS εφαρμογή για ένα mote, οι χρήστες μπορούν να μεταγλωττίσουν τη συγκεκριμένη εφαρμογή στο περιβάλλον του TOSSIM, το οποίο τρέχει σε ένα κοινό υπολογιστή. Αυτό επιτρέπει στους χρήστες να δοκιμάσουν, να αναλύσουν και να εκσφαλματώσουν αλγόριθμους σε ελεγχόμενο περιβάλλον με όσες επαναλήψεις θέλουν.
Η αρχιτεκτονική του TOSSIM συνθέτεται απο πέντε μέρη: υποστήριξη για μετατροπή των components γράφων του TinyOS στην υποδομή του εξομοιωτή,

μία ουρά διακριτών συμβάντων, ένα μικρό αριθμό διορθωμένων components υλικού, μηχανισμούς μοντέλων για ραδιοεπικοινωνία και μετατροπείς αναλογικών σε ψηφιακών πληροφοριών(ADC), και υπηρεσίες επικοινωνίας για εξωτερικές εφαρμογες για να αλληλεπιδρούν με την εξομοίωση. Ο TOSSIM εκμεταλλεύεται τη δομή του TinyOS και τη μεταγλώττιση ολόκληρου του συστήματος για να δημιουργήσει ένα διακριτών συμβάντων εξομοιωτή κατευθείαν από τους γράφους των components του TinyOS. Τρέχει τον ίδιο κώδικα που τρέχουν και οι πλατφόρμες των ασύρματων δικτύων αισθητήρων. Αντικαθιστώντας μερικά χαμηλού επιπέδου components, ο TOSSIM μεταφράζει τις διακοπές του υλικού σε διακριτά συμβάντα. Η ουρά συμβάντων του εξομοιώτη παραδίδει τις διακοπές, που "οδηγούν" την εκτέλεση της TinyOS εφαρμογής. Ο υπόλοιπος κώδικας του TinyOS εκτελείται αμετάβλητος.




Σχήμα 6.4: Αρχιτεκτονική του TOSSIM: Πλαίσια(Frames), Συμβάντα(Events), Μοντέλα(Models), Components, Υπηρεσίες(Services)

Ο TOSSIM χρησιμοποιεί μία πολύ απλή αλλά εκπληκτικά αποτελεσματική προσέγγιση για το ασύρματο δίκτυο του. Το δίκτυο είναι ένα κατευθυνόμενος γράφος, στον οποίο κάθε κορυφή είναι ένας κόμβος και κάθε ακμή έχει ένα bit-λάθος πιθανότητα. Κάθε κόμβος έχει μία ιδιωτική περιοχή που αντιπροσωπεύει τί ακούει στην ραδιοεπικοινωνία με τους άλλους κόμβους. Αυτή η προσέγγιση επιτρέπει δοκιμές κάτω απο συνθήκες τέλειας μετάδοσης(μηδενικό bit-error), μπορεί να απεικονίσει το πρόβλημα του κρυμμένου κόμβου( για τις κορυφές a,b,c υπαρχουν οι ακμες (a,b) και (b,c) αλλά οχι η ακμή (a,c)), και μπορεί να απεικονίσει διάφορα προβλήματα που μπορούν να προκύψουν στην μετάδοση. Ο μηχανισμός του εξομοιωτή παρέχει ένα σύνολο από υπηρεσίες επικοινωνίας για αλληλεπίδραση με εξωτερικές εφαρμογές. Αυτές οι υπηρεσίες επιτρέπουν προγράμματα να συνδέονται στον TOSSIM πάνω απο μία TCP socket και να παρακολουθούν ή να συμμετέχουν σε μία ενεργή εφαρμογή. Προγράμματα

μπορούν επίσης να λαμβάνουν πληροφορίες υψηλού επιπέδου, όπως μεταδόσεις και λήψεις πακέτων ή συμβάντα επιπέδου εφαρμογών.
Ο TOSSIM υποστηρίζει το σύνολο των εργαλείων του TinyOS, κάνοντας τη μετάβαση μεταξύ των εξομοιωμένων δικτύων και των πραγματικών εύκολη. Μεταγλωττίζοντας τον "αυθεντικό" κώδικα επιτρέπει στους προγραμματιστές να χρησιμοποιούν παραδοσιακά εργαλεία, όπως debuggers, στον TOSSIM. Παρέχει επίσης μηχανισμούς για την αλληλεπίδραση άλλων προγραμμάτων με ενεργές εφαρμογές. Παρακολουθώντας και συμμετέχοντας εξωτερικά στον TOSSIM, παραμένει το βασικό τμήμα του μηχανισμού του εξομοιωτή απλό και συνάμα αποδοτικό.
Ο πρωταρχικός σκοπός του TOSSIM είναι η παροχή εξομοίωσης μεγάλης ακρίβειας των TinyOS εφαρμογών. Για αυτό το λόγο, εστιάζει στην εξομοίωση του TinyOS και της εκτέλεσης του και όχι στην εξομοίωση του πραγματικού περιβάλλοντος.Παρόλο που ο TOSSIM μπορεί να χρησιμοποιηθεί για την κατανόηση των αιτιών των συμπεριφορών στον πραγματικό κόσμο, δεν μπορεί να τις αποτυπώσει όλες και γι'αυτό δεν θα πρέπει να χρησιμοποιείται για ακριβείς εκτιμήσεις.
Ο TOSSIM δεν είναι πάντα η σωστή λύση. Όπως κάθε εξομοιωτής, έτσι και αυτός κάνει κάποιες υποθέσεις, εστιάζοντας σε κάποιες συγκεκριμένες συμπεριφορές με αποτέλεσμα σε αυτές να είναι ακριβείς και σε κάποιες άλλες αφαιρετικός και απλουστευτικός. Παρακάτω παραθέτουμε μία περίληψη των χαρακτηριστικών του:

- **Πιστότητα:** Από μόνος του ο TOSSIM αποτυπώνει τη λειτουργία του TinyOS σε πολύ χαμηλό επίπεδο. Εξομοιώνει το δίκτυο απο το επίπεδο των bits, εξομοιώνει κάθε ξεχωριστό αναλογικό-σε-ψηφιακό μετατροπέα, καθώς και κάθε διακοπή στο σύστημα.

- **Χρόνος:** Παρόλο που ο TOSSIM καταγράφει επακριβώς τις διακοπές(επιτρέποντας εξομοιώσεις, όπως η ραδιοεπικοινωνία σε επίπεδο bit), δεν μοντελοποιεί τον χρόνο εκτέλεσης. Για τον TOSSIM ένα κομμάτι κώδικα τρέχει στιγμιαία. Ο χρόνος μετράται σε υποδιαιρέσεις των 4Mhz(ο χρονισμός του επεξεργαστή των Rene και των Mica πλατφορμών). Αυτό σημαίνει ότι οι αποκλεισμοί(spin locks) και οι αποκλεισμοί των διαδικασιών δεν θα τερματίζουν ποτέ: καθώς ο κώδικας τρέχει στιγμιαία, το συμβάν που θα ειδοποιεί τον αποκλεισμό να τερματίσει δεν θα συμβεί μέχρι να τελειώσει κώδικας(δηλαδή ποτέ).

- **Μοντέλα:** Ο TOSSIM δεν μοντελοποεί τον πραγματικό κόσμο. Αντίθετα, παρέχει στιγμιότυπα συγκεκριμένων φαινομένων πραγματικού κόσμου. Με εργαλεία εκτός του εξομοιωτή, οι χρήστες μπορούν να διαχειριστούν αυτά τα στιγμιότυπα και να υλοποιήσουν ότι μοντέλο θέλουν. Αφήνοντας τα πολύπλοκα μοντέλα εκτός του σχεδιασμού, ο TOSSIM καταφέρνει να μένει εύκαμπτος και ευέλικτος στις ανάγκες των περισσοτέρων χρηστών χωρίς να έχει ορίσει τί είναι σωστό και τί όχι. Παράλληλα κρατά τον εξομοιωτή απλό και αποδοτικό.

- **Radio:** Ο TOSSIM δεν μοντελοποιεί την διαδοση της ραδιοεπικοινωνίας. Αντίθετα παρέχει ένα στιγμιότυπο ανεξάρτητων λαθών στα bits μεταξύ δύο κόμβων. Έτσι ένα πρόγραμμα μπορεί να υλοποιεί ένα ράδιο-μοντέλο και να το ενσωματώσει στο στιγμιότυπο των λαθών. Έχοντας κλίμακα άμεσων bit-λαθών σημαίνει ότι οι ασύμμετρες διασυνδέσεις μπορούν να υλοποιηθούν εύκολα. Ανεξάρτητα bit-λάθη σημαίνει ότι τα μεγαλύτερα πακέτα έχουν μεγαλύτερη πιθανότητα φθοράς και ότι η πιθανότητα απώλειας κάθε πακέτου είναι ανεξάρτητη.
- **Ισχύς/Ενέργεια:** Ο TOSSIM δεν μοντελοποιεί κατανάλωση ενέργειας και ένδειξη ισχύος. Παρόλ'αυτά είναι πολυ εύκολα να κάνουμε προσθήκες στα components που καταναλώνουν ενέργεια ώστε να παρέχουν πληροφορίες όταν η κατάσταση ισχύος αλλάξει(π.χ. άνοιγμα-κλείσιμο). Αφού τελειώσει η εξομοίωση, ο χρήστης μπορεί να υλοποιήσει ένα ενεργειακό μοντέλο με αυτές τις αλλαγές, υπολογίζοντας την ολική κατανάλωση ενέργειας.

- "**Χτίσιμο**": Ο TOSSIM "χτίζεται" κατευθείαν από τον κώδικα του TinyOS. Για να εξομοιώσει κάποιος ένα πρωτόκολλο πρέπει να το υλοποιήσει πρώτα στο TinyOS. Από τη μία πλευρά αυτό συχνά είναι δύσκολο σε σύγκριση με μία εξομοίωση, από την άλλη όμως σημαίνει ότι μπορείς να τρέξεις την υλοποίηση κατευθείαν στις πραγματικές πλατφόρμες.

- **Ατέλειες:** Παρόλο που ο TOSSIM αποτυπώνει τη λειτουργία του TinyOS σε πολύ χαμηλό επίπεδο, κάνει κάποιες απλουστευτικές παραδοχές. Αυτο σημαίνει ότι κώδικας που τρέχει στον εξομοιωτή είναι πολύ πιθανό να μην τρέχει στις πραγματικές πλατφόρμες. Για παράδειγμα στον TOSSIM οι διακοπές δεν μπορούν να θέσουν σε αναμονή άλλες λειτουργίες(αποτέλεσμα του "διακριτών-συμβάντων" χαρακτήρα του). Στην πραγματικότητα όμως, μία διακοπή μπορεί να ενεργοποιηθεί ενόσο εκτελείται άλλος κώδικας. Αν μία αναμονή μπορεί να φέρει την πραγματική πλατφόρμα σε μη-αναστρέψιμη κατάσταση, τότε οι εξομοιωμένοι κόμβοι θα συνεχίσουν να τρέχουν ενώ οι πραγματικοί θα αποτύχουν. Επίσης αν τύχει οι διαχειριστές των διακοπών να χρειάζονται ώρα σε μία εφαρμογή, ένας πραγματικός κόμβος μπορεί να αποτύχει, ενώ στον TOSSIM επειδή ο κώδικας εκτελείται στιγμιαία δεν θα υπάρξουν προβλήματα στην εξομοίωση.

- **Δίκτυο:** Επί του παρόντος ο TOSSIM εξομοιώνει το 40Kbit RFM mica δικτυακό μοντέλο, περιλαμβάνοντας παραδοχές για MAC, κωδικοποίηση, χρόνο και για συγχρονισμό.

- **Κύρος:** Η αρχική εμπειρία από τις διασπορές ασύρματων δικτύων αισθητήρων έχει δείξει ότι τα TinyOS δίκτυα έχουν περίπλοκη και ασταθής συμπεριφορά. Παρά το γεγονός ότι ο TOSSIM είναι χρήσιμος για να καταδείξει την αποδοτικότητα ενός αλγορίθμου σε σχέση με κάποιον άλλο, δεν θα πρέπει τα αποτελέσματα του να θεωρούνται έγκυρα. Για παράδειγμα ο TOSSIM

μπορεί να δείξει ότι ένας αλγόριθμος συμπεριφέρεται καλύτερα από κάποιον αλλό κάτω από συνθήκες μεγάλων αποτυχιών στα πακέτα, αλλά το ερώτημα παραμένει αν το συγκεκριμένο σενάριο αποτυχιών έχει κάποια βάση στον πραγματικό κόσμο. Ο TOSSIM δεν θα πρέπει να θεωρείται το τέλος των αποτιμήσεων και των δοκιμών για ένα σύστημα, αντίθετα επιτρέπει στο χρήστη να κατανοήσει καλύτερα τους αλγόριθμους απομακρύνοντας τους εξωτερικούς παράγοντες, που πολλές φορές επηρεάζουν την απόδοση τους.

# Κεφάλαιο 7

# Ανάπτυξη ενός μοντέλου File System

## 7.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο μιλήσαμε για μοντέλα εξομοιωτών, τα οποία ήδη υπάρχουν και μπορεί κανείς να χρησιμοποιήσει. Στο παρόν κεφάλαιο θα παρουσιάσουμε ένα μοντέλο του file system, που χρησιμοποιείται στην οικογένεια των Mica motes(τα οποία χρησιμοποιούν το Atmel AT45DB041B flash ολοκληρωμένο μνήμης). Το file system ονομάζεται Matchbox και παρουσιάστηκε το 2003. Παρακάτω θα αναφερθούμε πιο διεξοδικά στα χαρακτηριστικά του. Το μοντέλο μας είναι υλοποιημένο στην προγραμματιστικη γλώσσα Java και μοντελοποιεί τις βασικές εντολές του συγκεκριμένου file system.

## 7.2 Χαρακτηριστικά Matchbox

Το Matchbox σχεδιάστηκε για να παρέχει ένα απλό file system για εφαρμογές βασισμένες στα Mica motes. Οι σχεδιαστικοί στόχοι, που καλείται να επιτύχει σύμφωνα με το [**?**] είναι οι παρακάτω :

- Αξιοπιστία,
- Χαμηλή Κατανάλωση Ενέργειας,
- Συμμόρφωση με τους τυπικούς περιορισμούς που έχουν οι μνήμες flash.

Οι περιορισμοί που θέτει το συγκεκριμένο flash ολοκληρωμένο μνήμης και θα πρέπει να λάβει υπόψιν του το Matchbox είναι οι ακόλουθοι:

- Η μνήμη flash χωρίζεται σε τομείς(κυριώς μεγέθους 128K, αν και μερικά είναι μικρότερα).
- Κάθε τομέας χωρίζεται σε σελίδες, κάθε μία από τις οποίες είναι μεγέθους 264 bytes.

- Οι σελίδες μπορούν μόνο να γραφούν εξ'ολοκλήρου, και όχι σε μεμονωμένα κομμάτια.
- Οι σελίδες πρέπει να σβήνονται πριν γραφούν.
- Μετά από 10000 εγγραφές σε ένα τομέα, όλες οι σελίδες πρέπει να έχουν γραφτεί τουλάχιστον μία φορά.

Παρόλαυτά ένα σημαντικό μέρος του σχεδιασμού του Matchbox βασίζεται στα στοιχεία,που δεν είναι σημαντικά για τη λειτουργικότητα του. Ειδικότερα δεν χρειαζόμαστε :

- καμίας μορφής ασφάλεια
- ιεραρχία
- προσπέλαση τυχαίων αρχείων
- πολλαπλές αναγνώσεις και εγγραφές του ίδιου αρχείου
- πολλά αρχεία ανοιχτά ταυτόχρονα. Όμως επιθυμούμε να έχουμε τη δυνατότητα να έχουμε (τουλάχιστον) δύο διαφορετικά αρχεία ανοιχτά, ένα για ανάγνωση και ένα για εγγραφή(υποστήριξη λειτουργιών αντιγραφής/αλλαγής)

Το Matchbox είναι ένα επίπεδο file system, το οποίο υποστηρίζει μόνο διαδοχικές αναγνώσεις και εγγραφές, που γράφουν από το σημείο που είχε τελειώσει η προηγούμενη εγγραφή. Τα αρχεία είναι αδόμητα(μία απλή ακολουθία bytes). Ο χώρος μπορεί προαιρετικά να δεσμευτεί από πριν(εξασφαλίζοντας ότι ο χώρος είναι διαθέσιμος, και μειώνοντας έτσι τις επιβαρύνσης εγγραφής).
Το ίδιο αρχείο δεν μπορεί να ανοιχτεί για εγγραφή και ανάγνωση ταυτόχρονα.
Τα αρχεία(που δεν είναι ανοιχτά) μπορούν να μετονομαστούν και να διαγραφούν.
Όπως και στο Unix, ονομάζοντας ένα αρχείο A όπως ένα ήδη υπάρχον αρχείο B, τότε διαγράφεται αυτόματα το B(η μετονομασία θα αποτύχει μόνο αν το B αρχείο είναι ανοιχτό).
Στο κομμάτι της υλοποίησης το Matchbox έχει αναπτυχθεί στη γλώσσα NesC, οπότε έχει όλα τα χαρακτηριστικά γνωρίσματα της. Έτσι το component του Matchbox παρουσιάζει την παρακάτω δομή:

```
provides {
    interface StdControl;
    interface FileDir;
    interface FileRename;
    interface FileDelete;
    interface FileRead[uint8_t fd];
    interface FileWrite[uint8_t fd];
   }
    uses {
        event result_t ready();
    }
```

Το μοντέλο μας υλοποιεί όλες τις παρεχόμενες διεπαφές και τα συμβάντα που χρησιμοποιεί το component. Στην παρακάτω ενότητα θα παρουσιάσουμε κάθε διεπαφή και την αντίστοιχη υλοποίηση της, καθώς και τα συμβάντα που σχετίζονται με αυτή.

## 7.3 Υλοποίηση

Στην παρούσα ενότητα θα παρουσιάσουμε τον κώδικα του υλοποιημένου μοντέλου. Η παρουσίαση ακολουθεί μία συγκεκριμένη ροή, κατά την οποία θα περιγράφουμε τις εντολές και τα συμβάντα των διεπαφών, εν συνεχεία θα παραθέτουμε τον κώδικα της διεπαφής σε NesC, όπως βρίσκεται στο TinyOS και τέλος θα παρουσιάζουμε και τον κώδικα μας. Όπως γίνεται αντιληπτό δεν θα παρουσιάζουμε τα αρχεία αυτούσια διότι κάτι τέτοιο θα απαιτούσε μεγάλη έκταση.
Η πρώτη κλάση που θα παρουσιάσουμε είναι η **Matchbox**. Η **Matchbox** αρχικοποιεί το μοντέλο μας, διαχειρίζεται τα συμβάντα και παρέχει πολλές βοηθητικές συναρτήσεις. Ειδικότερα:

```
public class Matchbox {

    protected String m_path; // Path ,that hosts our filing system
    protected int m_size;  //Size of our filing system

   protected boolean m_state; //Variable that indicates if a system is busy or not
    public int i, j;
    protected final Vector read_files = new Vector();
    protected final Vector write_files = new Vector();

    protected final Vector m_events = new Vector();

    /** List of listeners */
   protected final EventListenerList m_listeners = new EventListenerList();

    /**
     * Default constructor
     * @param path -- path where actual data are stored
     * @param totSize -- total size in KBytes
     */
    public Matchbox(String path, int totSize) {
        m_path = path;
        m_size = totSize * 1024; //Conversion in bytes
        m_state = false;
        i=j=0;
        (new Broadcaster()).start();
    }
```

Όπως βλέπουμε αρχικα ορίζει τις μεταβλητές, που θα χρησιμοποιηθούν και από τις υπόλοιπες διεπαφές. Τα σχόλια δίπλα από κάθε μεταβλητή εξηγούν το ρόλο της υπαρξης τους. Η συνάρτηση δημιουργού(constructor method) δινει τις τιμές που θέλουμε για το μονοπάτι που θα "φιλοξενίσει" το μοντέλο μας και το μέγεθος της υποτιθέμενης μνήμης του. Αρχικοποιεί την μεταβλητη που υποδηλώνει αν το σύστημα είναι απασχολημένο και ξεκινά τη διαδικασία γνωστοποίησης των συμβάντων, που πρόκειται να δημιουργηθούν. Ένα σημαντικό γνώρισμα της

κλάσης αυτής είναι ότι παρέχει τις λειτουργίες για τη διαχείριση των συμβάντων. Παρακάτω θα παρουσιάσουμε τις διάφορες μεθόδους, που πλαισιώνουν το περιβάλλον αυτό. Αρχικά έχουμε την κλάση Broadcaster

```
class Broadcaster extends Thread {

        public void run() {

            while (true) {

                try {
                    broadcastEvents();

                    Thread.sleep(1000);

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

        }

    }


}
```

Εδώ όπως βλέπουμε δημιουργούμε ένα νήμα, το οποίο τρέχει κάθε δευτερόλεπτο την συνάρτηση broadcastEvents, η οποία φαίνεται παρακάτω

```
protected void broadcastEvents() {
        EventObject[] ops = null;

        synchronized (m_events) {
            int listSize = m_events.size();
            ops = new EventObject[listSize];
            for (int i = 0; i < listSize; i++) {
                ops[i] = (EventObject) m_events.get(i);
            }

            m_events.clear();
        }

        if (ops == null) return;

        int i = ops.length;
        for (int cnt = 0; cnt < i; cnt++)
            broadcastEvent(ops[cnt]);
```

```
    }
```

Η παραπάνω συνάρτηση επιλέγει όλα τα συμβάντα που έχουν δημιουργηθεί και εκκρεμούν και τα κάνει broadcast στους υπάρχοντες Listeners μέσω της broadcastEvent.

```
protected void broadcastEvent(final EventObject e) {
       final Object listeners[] = m_listeners.getListenerList();
        final int totListeners = listeners.length;

       // Process the listeners last to first, notifying
        // those that are interested in this event
        for (int i = totListeners - 2; i >= 0; i -= 2) {
          EventListener l = (EventListener) listeners[i + 1];
            broadcastEventListener(l, e);
        }

    }
```

Η broadcastEvent ειδοποιεί όλους τους Listeners για το συμβαν μέσω της broadcastEventListener,

```
protected void broadcastEventListener(final EventListener l, final
EventObject e) {

       if ((l instanceof NextFileEventListener) && (e instanceof NextFileEvent)) {
          ((NextFileEventListener) l).processNextFileEvent((NextFileEvent) e);
        }
       if ((l instanceof RenameEventListener) && (e instanceof RenameEvent)) {
          ((RenameEventListener) l).processRenameEvent((RenameEvent) e);
        }
       if ((l instanceof DeleteEventListener) && (e instanceof DeleteEvent)) {
          ((DeleteEventListener) l).processDeleteEvent((DeleteEvent) e);
        }
       if ((l instanceof appendedEventListener) && (e instanceof appendedEvent)) {
          ((appendedEventListener) l).processappendedEvent((appendedEvent) e);
        }
       if ((l instanceof closedEventListener) && (e instanceof closedEvent)) {
          ((closedEventListener) l).processclosedEvent((closedEvent) e);
        }
       if ((l instanceof OpenedWriteEventListener) && (e instanceof OpenedWriteEvent)) {
          ((OpenedWriteEventListener) l).processOpenedWriteEvent((OpenedWriteEvent) e);
        }
       if ((l instanceof OpenedReadEventListener) && (e instanceof OpenedReadEvent)) {
          ((OpenedReadEventListener) l).processOpenedReadEvent((OpenedReadEvent) e);
        }
       if ((l instanceof readDoneEventListener) && (e instanceof readDoneEvent)) {
          ((readDoneEventListener) l).processreadDoneEvent((readDoneEvent) e);
        }
       if ((l instanceof remainingEventListener) && (e instanceof remainingEvent)) {
          ((remainingEventListener) l).processremainingEvent((remainingEvent) e);
        }
       if ((l instanceof RenameEventListener) && (e instanceof RenameEvent)) {
          ((RenameEventListener) l).processRenameEvent((RenameEvent) e);
        }
       if ((l instanceof reservedEventListener) && (e instanceof reservedEvent)) {
          ((reservedEventListener) l).processreservedEvent((reservedEvent) e);
        }
       if ((l instanceof syncedEventListener) && (e instanceof syncedEvent)) {
```

```
            ((syncedEventListener) l).processsyncedEvent((syncedEvent) e);
        }
    }
```

Όπως βλέπουμε η broadcastEventListener επιλέγει σε ποιόν Listener απευθύνεται το συμβάν και το στέλνει σε αυτόν. Για να προσθέσουμε ένα συμβάν στη λίστα καλούμε την addEvent, η οποία προσθέτει το συμβάν αυτό στο τέλος της λίστας

```
public synchronized void addEvent(final EventObject e) {
        m_events.add(e);
        notifyAll();
    }
```

Για να προσθέσουμε και να αφαιρέσουμε Listeners χρησιμοποιούμε τις δύο παρακάτω συναρτήσεις

```
public synchronized void addNextFileEventListener(final NextFileEventListener l) {
        m_listeners.add(NextFileEventListener.class, l);
    }

public synchronized void removeNextFileEventListener(final EventListener l) {
        m_listeners.remove(NextFileEventListener.class, l);
    }
```

Η διεπαφή που μας δίνει πληροφορίες για το περιεχόμενο της μνήμης είναι η **FileDir**. Τα περιεχόμενα της μνήμης εμφανίζονται καλώντας την εντολή start και μετα καλώντας την readNext για να πάρουμε κάθε διαφορετικό αρχείο. Τα αρχεία επιστρέφονται στο nextFile συμβάν και το σύστημα θεωρείται απασχολημένο από την κλήση της start μέχρι το συμβάν nextFile να επιστρέψει ένα λάθος(κανονικά το FS_NO_MORE_FILES λάθος). Ο αριθμός των ελεύθερων bytes στο σύστημα δίνεται από την εντολή freeBytes. Ο κώδικας της FileDir όπως δίνεται από τους δημιουργούς της είναι:

```
/**
 * List files in filing system
 */
interface FileDir {
  /**
   * List names of all files in filing system. Filing system is busy
   * until FAIL is returned from <code>readNext</code> or no more
   * files remain.
   * @return
   *   SUCCESS: files can be read with <code>readNext</code>
   *   FAIL: filesystem is busy
   */
  command result_t start();

  /**
   * Return next file in filing system
   *
   * @return
   *   SUCCESS: attempt proceeds, <code>nextFile</code> will be signaled
   *   FAIL: no file list operation is in progress.
   */
```

```
  command result_t readNext();

  /**
   * Report next file name.
   * @param filename One of the files in the filing system if
   *   <code>result</code> is FS_OK, NULL otherwise. The storage for
   *   filename only remains valid until the end of the event.
   * @param result
   *   FS_OK filename is the next file.
   *   FS_NO_MORE_FILES No more files.
   *   FS_ERROR_xxx Filing system data is corrupt.
   * @return
   *   SUCCESS: continue reporting file names<br>
   *   FAIL: abort the file lising operation<br>
   * If the <code>result</code> is an error or no more files, the file
   * listing operation terminates.
   */
  event result_t nextFile(const char *filename, fileresult_t result);

  /**
   * @return Number of bytes available in filing system.
   */
  command uint32_t freeBytes();
}
```

Για κάθε εντολή και συμβάν θα παρουσιάσουμε τώρα τη δική μας υλοποίηση. Για την εντολή start θα έχουμε :

```
public Result start() {

        // if file system is busy, return FAIL
        if (m_fs.isFileSystemBusy()) {
             return Result.FAIL;
        }

        // make file system busy
        m_fs.setFileSystemBusy();

        File path = new File(m_fs.getPath());
        m_files = path.list();

        m_currentFile = 0;

        return Result.SUCCESS;
    }
```

Αρχικά ελέγχουμε αν το σύστημα είναι απασχολημένο. Αν ναι επιστρέφουμε FAIL, διαφορετικά ανοίγουμε ένα file descriptor, δημιουργούμε τη λίστα με τα αρχεία και θέτουμε 0 τη μεταβλητή, που μετράει τα ήδη εμφανιζόμενα αρχεία.

```
public Result readNext() {

        if (m_currentFile >= m_files.length) {
```

```
            // file system is no longer busy
            m_fs.setFileSystemIdle();

            // Throw a nextfile event signaling
            //that no more files are available
           nextFile(null, FileResult.FS_NO_MORE_FILES);

            return Result.FAIL;
        }

        // Throw new nextfile event
       nextFile(m_files[m_currentFile++], FileResult.FS_OK);

        return Result.SUCCESS;
    }
```

Η readNext ελέγχει αν ο αριθμός των αρχείων που έχουν εμφανιστεί είναι μεγαλύτερα από αυτά που υπάρχουν και αν είναι όντως μεγαλύτερος τερματίζει τη διαδικασία "δίνοντας" ένα συμβάν με παράμετρο FS_NO_MORE_FILES. Διαφορετικά συνεχίζει τη διαδικασία με ένα ίδιο συμβάν με διαφορετική όμως παράμετρο FS_OK.

```
public Result nextFile(final String filename, final FileResult
result) {

        // Throw new nextfile event
       NextFileEvent newEvent = new NextFileEvent(this, filename, result);

        // Pass new event to Matchbox
        m_fs.addEvent(newEvent);

        return Result.SUCCESS;
    }
```

Η παραπάνω συνάρτηση είναι η υλοποίηση του συμβάντος nextFile, και αυτό που κάνει είναι να προσθέτει ένα τέτοιο συμβάν στη λίστα με τα συμβάντα.

```
public int freeBytes() {
        File path = new File(m_fs.getPath());
        final String[] files = path.list();
        final int totFiles = files.length;
        int totSizeUsed = 0;
        for(int i = 0; i < totFiles; i ++) {
            File file = new File(path, files[i]);
            totSizeUsed += file.length();
        }

        return m_fs.getTotalSize() - totSizeUsed;
    }
```

Η τελευταία συνάρτηση της FileDir είναι η freeBytes, στην οποία υπολογίζουμε πόσο από το μέγεθος που ορίσαμε στην αρχικοποίηση της κλάσης Matchbox είναι ελεύθερο αν αφαιρέσουμε τα μεγέθη των αρχείων, που υπάρχουν στο

μονοπάτι που φιλοξενεί το file system μας.
Η διεπαφή, που ακολουθεί είναι η FileDelete. Πρέπει να επισημάνουμε εδώ ότι ανοιχτά αρχεία για ανάγνωση ή εγγραφή δεν μπορούν να διαγραφούν ή να μετονομαστούν. Ο κώδικας της σε NesC φαίνεται παρακάτω:

```
/**
 * Delete a file
 */
interface FileDelete {
  /**
   * Delete a file
   * @param filename Name of file to delete. Must not be stack allocated.
   * @return
   *   SUCCESS: attempt proceeds, <code>deleted</code> will be signaled<br>
   *    FAIL: filesystem is busy
   */
  command result_t delete(const char *filename);

  /**
   * Signaled at the end of a file delete attempt
   * @param result
   *   FS_OK: file was deleted<br>
   *   FS_ERROR_xxx: delete failure cause
   * @return Ignored
   */
  event result_t deleted(fileresult_t result);
}
```

Όπως βλέπουμε αποτελείται μόνο από μία εντολή delete και ένα συμβάν deleted, το οποίο δημιουργείται στο τέλος της εντολής. Η αντίστοιχη υλοποίηση μας της εντολής delete φαίνεται παρακάτω:

```
public Result Delete(final String name) {


        // if file system is busy, return FAIL
        if (m_fs.isFileSystemBusy()) {
             return Result.FAIL;
        }


        String filename=m_fs.getPath()+name;
        File file = new File(filename);
        if(m_fs.openfile_exists(filename)){
            file.delete();
            Deleted(FileResult.FS_OK);
        }
        else if(!m_fs.openfile_exists(filename)){
            Deleted(FileResult.FS_ERROR_FILE_OPEN);
        }
        else if(!file.exists()){
            Deleted(FileResult.FS_ERROR_NOT_FOUND);
        }
```

```
        return Result.SUCCESS;

    }
```

Αρχικά ελέγχουμε αν το σύστημα είναι απασχολημένο και αν όχι τότε διαγράφουμε το αρχείο και δημιουργούμε συμβάν με παράμετρο FS_OK. Αν ναι τότε επιστρέφουμε FAIL. Παρόλαυτά αν το αρχείο δεν υπάρχει δημιουργούμε συμβάν με παράμετρο FS_ERROR_NOT_FOUND, ενώ αν είναι ήδη ανοιχτό για ανάγνωση ή εγγραφή δημιουργούμε συμβάν με παράμετρο FS_ERROR_FILE_OPEN.Και στις δύο περιπτώσεις όμως επιστρέφουμε SUCCESS.

```
public Result Deleted(final FileResult result){

       DeleteEvent newEvent = new DeleteEvent(this, result);

        // Pass new event to Matchbox
        m_fs.addEvent(newEvent);

        return Result.SUCCESS; //Ignored

    }
```

Τέλος το συμβάν, που δημιουργείται από την εντολή delete επιστρέφει SUCCESS και προσθέτει το συγκεκριμένο συμβάν στη λίστα με τα συμβάντα.

Η διεπαφή FileRename είναι παρόμοια με τη FileDelete σε επίπεδο πολυπλοκότητας και δυσκολίας. Έχει και αυτή μία εντολή rename και ένα συμβάν renamed, όπως φαίνεται παρακάτω. Μετονομάζοντας ένα αρχείο X σε ένα ήδη υπάρχον αρχείο Y έχει ως αποτέλεσμα την διαγραφή του Y(η διαδικασία θα αποτύχει όμως αν το Y είναι ανοιχτό).

```
/**
 * Rename a file
 */
interface FileRename {
  /**
   * Rename a file. If a file called <code>newName</code> exists, it is
   * deleted.
   * @param oldName Name of file to rename. Must not be stack allocated.
   * @param newName New name of file. Must not be stack allocated.
   * @return
   *   SUCCESS: attempt proceeds, <code>renamed</code> will be signaled<br>
   *   FAIL: filesystem is busy or newName is ""
   */
  command result_t rename(const char *oldName, const char *newName);

  /**
   * Signaled at the end of a file rename attempt
   * @param result
   *   FS_OK: file was renamed<br>
   *   FS_ERROR_xxx: rename failure cause
```

```
   * @return Ignored
   */
  event result_t renamed(fileresult_t result);
}
```

Παρακάτω φαίνεται η υλοποιημένη εντολή rename:

```
public Result Rename(final String oldname, final String newname) {

        String oldfilename=m_fs.getPath()+oldname;
        String newfilename=m_fs.getPath()+newname;
        File oldfile = new File(oldfilename);
        File newfile = new File(newfilename);

       if (newname.compareTo(" ") == 0 || m_fs.isFileSystemBusy()) {
            return Result.FAIL;
       }

        if(oldfile.exists()){
          if(m_fs.openfile_exists(oldfilename) && m_fs.openfile_exists(newfilename
               newfile.delete();
               oldfile.renameTo(newfile);
               Renamed(FileResult.FS_OK);
          }
          else {
               Renamed(FileResult.FS_ERROR_FILE_OPEN);
          }
        }
        else {
            Renamed(FileResult.FS_ERROR_NOT_FOUND);
        }


        return Result.SUCCESS;


    }
```

Όπως βλέπουμε αρχικά ελέγχει για τη διαθεσιμότητα του συστήματος και για τον αν το νέο όνομα είναι μία κενή συμβολοσειρά. Αν ικανοποιούνται αυτες οι συνθήκες ελέγχει αν υπάρχει το αρχείο προς μετονομασία υπάρχει. Αν όχι δημιουργεί ένα συμβάν με παράμετρο FS_ERROR_NOT_FOUND. Αν ικανοποιούνται τότε ελέγχει μήπως κάποιο από τα δύο αρχεία είναι ανοιχτό και αν δεν είναι τότε μετονομάζει το αρχείο και σβήνει το άλλο, που είχε το ίδιο όνομα.Αλλιώς αν ένα από τα δύο ήταν ανοιχτό δημιουργεί ένα συμβάν με παράμετρο FS_ERROR_FILE_OPEN.

```
public Result Renamed(final FileResult result){

        RenameEvent newEvent = new RenameEvent(this, result);

        // Pass new event to Matchbox
        m_fs.addEvent(newEvent);

        return Result.SUCCESS;

    }
```

Όπως και στα προηγούμενα συμβάντα έτσι και εδώ η λειτουργία της συνάρτησης αυτής είναι να προσθέτει στη λίστα με τα συμβάντα, το συγκεκριμένο συμβάν.

Οι επόμενες διεπαφές είναι η FileRead και η FileWrite. Κάθε στιγμιότυπο των δύο αυτών διεπαφών αντιστοιχεί σε ένα ξεχωριστό file descriptor και επιτρέπεται να ανοίγονται δοαφορετικά αρχεία. Οι file descriptors για ανάγνωση και εγγραφή είναι ξεχωριστοί. Έτσι αν για παράδειγμα μία εφαρμογή χρειάζεται ταυτόχρονα δύο αρχεία ανοιχτά για ανάγνωση και ένα για εγγραφή τότε ο κώδικας σε NesC θα πρέπει να είναι όπως παρακάτω:

```
configuration MyApp { }
implementation {
    components Matchbox,MyCode, ...;

   MyCode.FileRead1 ->Matchbox.FileRead[unique("FileRead")];
   MyCode.FileRead2 ->Matchbox.FileRead[unique("FileRead")];
   MyCode.FileWrite ->Matchbox.FileWrite[unique("FileWrite")];
    ...
   }
```

Ειδικότερα τώρα για την FileRead ορίζει τις λειτουργίες για άνοιγμα(εντολή open, opened συμβάν), κλείσιμο(close εντολή, θέτει το σύστημα σε κατάσταση απασχόλησης), ανάγνωση(εντολή read, readDone συμβάν) και υπολογισμός εναπομείναντων bytes στο αρχείο. Παρακάτω φαίνεται ο πρωταρχικος κώδικας για όλα αυτά:

```
/**
 * File reading interface, supports sequential reads.
 */

interface FileRead {
  /**
   * open a file for sequential reads.
   * @param filename Name of file to open. Must not be stack allocated.
   * @return
   *   SUCCESS: attempt proceeds, <code>opened</code> will be signaled<br>
   *   FAIL: filesystem is busy or another file is open for reading
   */
  command result_t open(const char *filename);

  /**
   * Signaled at the end of a file open attempt
   * @param result
   *   FS_OK: file was opened<br>
   *   FS_ERROR_xxx: open failure cause
   * @return Ignored
   */
  event result_t opened(fileresult_t result);

  /**
   * Close file currently open for reading
   * @return SUCCESS if a file was open, FAIL otherwise
   */
  command result_t close();

  /**
   * Read bytes sequentially from open file.
```

```
   * @param buffer Target to read into
   * @param n Number of bytes to read
   * @return
   *   SUCCESS: attempt proceeds, <code>readDone</code> will be signaled<br>
   *    FAIL: no file was open for reading, or a read is in progress
   */
  command result_t read(void *buffer, filesize_t n);

  /**
   * Signaled when a <code>read</code> completes
   * @param buffer Buffer that was passed to <code>read</code>
   * @param nRead Number of bytes actually read ,
   *    but result will still be FS_OK)
   * @param result
   *    FS_OK: read was successful (if end-of-file is reached,
   *    <code>nRead</code> will be less than the number of bytes requested)<br>
   *    FS_ERROR_xxx: read failure cause.
   * @return Ignored
   */
  event result_t readDone(void *buffer, filesize_t nRead,
              fileresult_t result);

  /**
   * Return number of bytes remaining in file.
   * @return
   *   SUCCESS: attempt proceeds, <code>remaining</code> will be signaled<br>
   *    FAIL: no file was open for reading, or a read is in progress
   */
  command result_t getRemaining();

  /**
   * Signaled when <code>getRemaining</code> completes
   * @param n Number of bytes remaining in file
   * @param result
   *    FS_OK: operation was successful
   *    FS_ERROR_xxx: read failure cause
   * @return Ignored
   */
  event result_t remaining(filesize_t n, fileresult_t result);
}
```

Η υλοποίηση της open είναι:

```
public Result open(final String name) {



        if (m_fs.isFileSystemBusy()) {
            return Result.FAIL;
           }

        filename=m_fs.getPath()+name;
        if(m_fs.write_exists(filename)){
            file = new File(filename);
            try {
                fd = new FileReader(file);
                m_fs.add_read(filename);
                }
```

```
            catch (FileNotFoundException ioe) {
                opened(FileResult.FS_ERROR_NOT_FOUND);
                return Result.SUCCESS;
                }
            offset = 0;
            opened(FileResult.FS_OK);
        }
        else{
            opened(FileResult.FS_ERROR_FILE_OPEN);
            return Result.SUCCESS;
        }

        return Result.SUCCESS;
    }
```

Όπως βλέπουμε αφού ελέγξει για τη διαθεσιμότητα του συστήματος και για τον αν το ίδιο αρχείο είναι ήδη ανοιγμένο για εγγραφή δημιουργεί ένα file descriptor και προσθέτει το όνομα του αρχείου στη στοίβα με τα αρχεία που είναι ανοιγμένα. Έπειτα θέτει το σημείο εκκίνησης ανάγνωσης ίσο με το μηδέν και δημιουργεί ένα opened συμβάν, το οποίο έιναι παρόμοιο με όλα τα παραπάνω συμβάντα, γιαυτό και παραλείπουμε τον κώδικα του.
Η εντολή close δεν κάνει τίποτα ιδιαιτερο από το να κλείνει το αρχείο, γιαυτό και την παραλείπουμε. Παραθέτουμε απλά τον υλοπποιημένο κώδικα:

```
public Result close() {
        try {
            m_fs.setFileSystemBusy();
            fd.close();
            m_fs.remove_read(filename);
            m_fs.setFileSystemIdle();
            return Result.SUCCESS;
        } catch (IOException ioe) {
            System.out.println(ioe.toString());
            ioe.printStackTrace();
            return Result.FAIL;
        }
    }
```

Παρατηρούμε μόνο ότι χρειάζεται να θέσει το σύστημα σε κατάσταση απασχόλησης. Παράλληλα βλέπουμε ότι διαγράφει και το όνομα του αρχείου από τη λίστα με τα αρχεία που είναι ανοιχτα για ανάγνωση.
Η εντολή read είναι ουσιαστικά υπεύθυνη για τη ανάγνωση.

```
public Result read(char[] buffer, int n, final FileResult result) {


        long nreads_after_EOF;

        if(m_read_state==true || file!=null) {
        try {
            m_read_state=false;
            if(fd.read(buffer, offset, n)!=-1){
```

```
                offset = offset + n;
                readDone(buffer, n, FileResult.FS_OK);
                return Result.SUCCESS;
            }
            else{
                nreads_after_EOF=file.length()-offset;
              readDone(buffer, nreads_after_EOF, FileResult.FS_OK);
                return Result.SUCCESS;
            }
        } catch (IOException ioe) {
            System.out.println(ioe.toString());
            ioe.printStackTrace();
        }
        }
        else
            return Result.FAIL;

    }
```

Τα αξιοσημείωτα σημεία εδώ είναι δύο, η μεταβλητή m_read_state, η οποία χρησιμοποιείται για τη λειτουργία του συντονισμού μεταξύ διαφόρων αναγνώσεων και οι δύο περιπτώσεις για τον αριθμό των bytes που αναγνώθηκαν. Η πρώτη περίπτωση είναι όταν διαβάζει κανονικά όσα bytes έχουμε ορίσει εμείς, και η δεύτερη είναι όταν φτάσει στο τέλος του αρχείου έχοντας διαβάσει λιγότερα bytes από αυτά που έχουμε ορίσει. Και στις δύο όμως περιπτώσεις δημιουργείται συμβάν με παράμετρο FS_OK. Το readDone συμβάν φαίνεται παρακάτω:

```
public Result readDone(char[] buffer, long nRead, final FileResult result) {


        m_read_state=true;
        // Throw new nextfile event
       readDoneEvent newEvent = new readDoneEvent(this, buffer, nRead, result);

        // Pass new event to Matchbox
        m_fs.addEvent(newEvent);

        return Result.SUCCESS;
    }
```

Η μόνη διαφορά εδώ σε σχέση με τα υπόλοιπα είναι ότι θέτει τη μεταβλητή m_read_state σε κατάσταση αληθείας ώστε να μπορεί να λάβει χώρα κάποια άλλη ανάγνωση ή μία εντολή getRemaining.
Η τελευταία εντολή για την διεπαφή αυτή είναι η get Remaining, η οποία επιστρέφει τα εναπομείναντα μη αναγνωσμένα bytes. Ο κώδικας φαίνεται παρακάτω:

```
public Result getRemaining() {

        long nremaining;

        if(m_read_state==true || file!=null) {
            nremaining=file.length()-offset;
            remaining(nremaining, FileResult.FS_OK);
```

```
            return Result.SUCCESS;
        }
        else return Result.FAIL;
    }
```

Η παραπάνω εντολή ελέγχει την μεταβλητή m_read_state για να δει αν υπάρχει ανάγνωση ενεργή και ελέγχει αν όντως έχει ανοιχθεί κάποιο αρχείο. Αν πληρεί τις δύο αυτές προυποθέσεις αφαιρεί από το μέγεθος του αρχείου την τελευταία μετατόπιση που έγινε κατά την τελευταία ανάγνωση και έτσι υπολογίζει το ζητούμενο. Μετά από τον επιτυχή υπολογισμό δημιουργείται συμβάν remaining, το οποίο προσθέτει το συγκεκριμένο συμβάν στην λίστα συμβάντων και είναι παρόμοιο με τα αρχικά συμβάντα.

Τέλος έχουμε τη διεπαφή FileWrite. Η διεπαφή FileWrite ορίζει τις λειτουργίες για άνοιγμα(εντολή open, opened συμβάν), κλείσιμο(close εντολή, closed συμβάν), επισυνάψεις(εντολή append και appended συμβάν), συγχρονισμό(εντολή sync και synced συμβάν) και προκράτηση χώρου για ένα αρχείο(εντολή reserve και reserved συμβάν). Οι επισυνάψεις διασφαλίζεται ότι έχουν αποθηκευτεί μόνο μετά από ένα closed ή ένα synced συμβάν. Χώρος μπορεί να κρατηθεί για ένα αρχείο χρησιμοποιώντας την εντολή reserve, η οποία εγγυάται ότι συνεχόμενες επισυνάψεις μέχρι το ζητούμενο μέγεθος δεν θα αποτύχουν λόγω έλλειψης χώρου, και έτσι αυτό οδηγεί σε ταχύτερες επισυνάψεις. Ο κώδικας ,λοιπόν, της διεπαφής αυτης σε NesC είναι:

```
 /**
 * File reading interface, supports appending writes.
 */

interface FileWrite {
  /**
   * open a file for sequential reads.
   * @param filename Name of file to open. Must not be stack allocated.
   * @param flags: open options, an or (|) of FS_Fxxx constants. <br>
   *   <code>FS_FTRUNCATE</code> Truncate file if it exists<br>
   *   <code>FS_FCREATE</code> Create file if it doesn't exist
   * @param truncate TRUE if file should be truncated if it exists
   * @return
   *   SUCCESS: attempt proceeds, <code>opened</code> will be signaled<br>
   *   FAIL: filesystem is busy, another file is already open for writing,
   *     filename is ""
   */
  command result_t open(const char *filename, uint8_t flags);

  /**
   * Signaled at the end of a file open attempt
   * @param fileSize size of file (if file was opened)
   * @param result
   *   FS_OK: file was opened<br>
   *   FS_ERROR_xxx: open failure cause
   * @return Ignored
   */
  event result_t opened(filesize_t fileSize, fileresult_t result);

  /**
```

```
 * close file currently open for writing
 * @return
 *   SUCCESS: attempts proceeds, <code>closed</code> will be signaled<br>
 *   FAIL: no file was open for writing, or a close/append/reserve/sync
 *     is in progress
 */
command result_t close();

/**
 * Signaled at the end of a file close. File is closed in all cases,
 *   including failure (but in case of failure some data may have been lost).
 * @param result
 *   FS_OK: file was closed without problems. All data has been comitted to
 *     stable storage.<br>
 *   FS_ERROR_xxx: close failure cause
 * @return Ignored
 */
event result_t closed(fileresult_t result);

/**
 * Write bytes sequentially to end of open file.
 * @param buffer Data to write
 * @param n Number of bytes to write
 * @return
 *   SUCCESS: attempt proceeds, <code>appended</code> will be signaled<br>
 *   FAIL: no file was open for writing, or a close/append/reserve/sync
 *     is in progress
 */
command result_t append(void *buffer, filesize_t n);

/**
 * Signaled when a <code>append</code> completes
 * @param buffer Buffer that was passed to <code>append</code>
 * @param nWritten Number of bytes actually written
 *   but result will still be FS_OK)
 * @param result
 *   FS_OK: write was successful
 *   FS_ERROR_xxx: write failure cause. Some bytes may have been written
 *     (as reported by the value of <code>nWritten</code>
 * @return Ignored
 */
event result_t appended(void *buffer, filesize_t nWritten,
            fileresult_t result);

/**
 * Reserve space for the currently open file to be <code>newSize</code>
 * bytes long. <code>append</code>s that do not make the file take
 * more than <code>newSize</code> bytes will not fail with FS_ERROR_NOSPACE.
 * Note: you can find the reserved size of a file by requesting a reserve
 * with a newSize of 0. The <code>reserved</code> event will indicate the
 * space currently reserved.
 * @param newSize Size file is expected to grow to
 * @return
 *   SUCCESS: attempt proceeds, <code>reserved</code> will be signaled<br>
```

```
   *   FAIL: no file was open for writing, or a close/append/reserve/sync
   *     is in progress
   */
  command result_t reserve(filesize_t newSize);

  /**
   * Signaled at the end of a space reservation attempt
   * @param maxSize New reserved size (>= requested size)
   * @param result
   *   FS_OK: space was successfully reserved<br>
   *   FS_ERROR_xxx: failure cause
   * @return Ignored
   */
  event result_t reserved(filesize_t reservedSize, fileresult_t result);

  /**
   * Ensure data appended is comitted to stable storage.
   * @return
   *   SUCCESS: attempt proceeds, <code>synced</code> will be signaled<br>
   *   FAIL: no file was open for writing, or a close/append/reserve/sync
   *     is in progress
   */
  command result_t sync();

  /**
   * Signaled at the end of a sync attempt
   * @param result
   *   FS_OK: sync was successful<br>
   *   FS_ERROR_xxx: failure cause
   * @return Ignored
   */
  event result_t synced(fileresult_t result);
}
```

Αρχίζοντας με την εντολή open παραθέτουμε αρχικά τον κώδικα της:

```
 public Result open(final String name, final int flag) {


        if (m_fs.isFileSystemBusy()) {

            return Result.FAIL;
           }

        filename=m_fs.getPath()+name;
        file = new File(filename);
        if(m_fs.read_exists(filename)){

            if (flag == 0) {
                try {
                    if(file.exists()){
                        fd = new FileWriter(file, true);
                        m_fs.add_write(filename);
                      opened(file.length(), FileResult.FS_OK);
                        return Result.SUCCESS;
```

```
                    }
                    else{
                     opened(file.length(), FileResult.FS_ERROR_NOT_FOUND);
                         return Result.SUCCESS;
                    }
                }

               catch (IOException ioe) {
                    System.out.println(ioe.toString());
                    ioe.printStackTrace();
                }
            }
            else {
                try {
                    fd = new FileWriter(file, false);
                    m_fs.add_write(filename);
                    opened(file.length(), FileResult.FS_OK);
                    return Result.SUCCESS;
                }
                catch (IOException ioe) {
                    System.out.println(ioe.toString());
                    ioe.printStackTrace();
                }
            }
        }
        else{
          opened(file.length(),FileResult.FS_ERROR_FILE_OPEN);
            return Result.SUCCESS;
        }

        return null;
    }
```

Όπως βλέπουμε η συνάρτηση μέτα από τους απαραίτητους ελέγχους για τη διαθεσιμότητα του συστήματος και την περίπτωση να είναι το αρχείο ήδη ανοιγμένο για ανάγνωση διαχωρίζεται σε δύο περιπτώσεις. Η μία είναι όταν αρχείο ήδη υπάρχει, οπότε η flag είναι 0, δηλαδή F_TRUNCATE. Ελέγχει αν όντως υπάρχει το αρχείο και αν ναι δημιουργεί τον file descriptor, προσθέτει το όνομα του αρχείου στη λίστα με τα ονόματα των αρχείων που είναι ανοιχτά για εγγραφή, δημιουργεί το συμβάν opened με παράμετρο FS_OK και επιστρέφει SUCCESS. Αν όχι δημιουργεί απλώς ένα συμβάν opened με παράμετρο FS_ERROR_NOT_FOUND και επιστρέφει SUCCESS. Η δεύτερη περίπτωση είναι όταν δημιουργούμε νεό αρχείο. Σε αυτή την περίπτωση δημιουργούμε τον file descriptor, προσθέτουμε το όνομα του αρχείου στη λίστα με τα ονόματα των αρχείων που είναι ανοιχτά για εγγραφή, δημιουργούμε το συμβάν opened με παράμετρο FS_OK και τέλος επιστρέφουμε SUCCESS. Το συμβάν opened δεν αποτελεί κάτι ιδιαίτερο, αφού είναι ίδιο με τα παραπάνω.
Η εντολή append φαίνεται παρακάτω:

```
public Result append(char[] buffer, int n) {
```

```
        if(m_write_state==false || file==null) {

            return Result.FAIL;
        }

        try {
            m_write_state=false;
            fd.write(buffer, offset, n);
            offset = offset + n;
            appended(buffer, n, FileResult.FS_OK);
            return Result.SUCCESS;
            }
        catch (IOException ioe) {
            System.out.println(ioe.toString());
            ioe.printStackTrace();
        }

    }
```

Αρχικά εδώ ελέγχουμε για τον αν κάποια άλλη εντολή από τις append,sync,reserve,close εκτελείται τη συγκεκριμένη στιγμή και αν υπάρχει πράγματι ανοιχτό αρχείο για εγγραφή. Αφού διασφαλίσουμε αυτές τις δύο προϋποθέσεις θέτουμε τη μεταβλητή m_write_state σε ψευδή κατάσταση, ώστε να μην μπορεί να εκτελειστεί άλλη εντολή από τις παραπάνω και γράφουμε στο αρχείο. Ανανεώνουμε το δείκτη εγγραφής(offset) και δημιουργούμε ένα νέο συμβάν appended με παράμετρο FS_OK. Το συμβάν διαφέρει από τα παραπάνω στο ότι θέτει την m_write_state σε αληθή τιμή και πάλι ώστε να μπορούν να εκτελεστούν άλλες εντολές.
Η εντολή reserve όπως φαίνεται παρακάτω

```
public Result reserve(final int newSize){

        if(m_write_state==false || file==null) {

            return Result.FAIL;
        }
        m_write_state=false;
        ms_remaining_space=m_dir.freeBytes()-newSize;
        if(ms_remaining_space<0)
        {
          reserved(newSize, FileResult.FS_ERROR_NOSPACE);
            return Result.SUCCESS;
        }
        else{
            reserved(newSize, FileResult.FS_OK);
            return Result.SUCCESS;
        }
    }
```

επιτυχαίνει το στόχο της αφαιρώντας από τα εναπομείναντα bytes του μοντελοποιημένου συστήματος μας τον αριθμό των bytes που θέλουμε να προκρατήσουμε. Αν ο

χώρος που απομένει στο σύστημα δεν επαρκεί τότε δημιουργούμε συμβάν reserved με παράμετρο FS_ERROR_NOSPACE. Αλλιώς δημιουργούμε συμβάν reserved με παράμετρο FS_OK.
Οι επόμενες δύο εντολές είναι οι sync και close. Δεν παρουσιάζουν κάποια ιδιαίτερη δυσκολία γιαυτό απλά τις παραθέτουμε. Sync:

```
public Result sync() throws IOException {

      if(m_write_state==false || file==null) {

          return Result.FAIL;
      }

      m_write_state=false;
      fd.flush();
      synced(FileResult.FS_OK);
      return Result.SUCCESS;

  }
```

Close:

```
public Result close() throws IOException {

      if(m_write_state==false || file==null) {

          return Result.FAIL;
      }

      m_write_state=false;
      fd.close();
      closed(FileResult.FS_OK);
      return Result.SUCCESS;

  }
```

Θα πρέπει όμως να αναφέρουμε ότι κα οι τρεις τελευταίες συναρτήσεις έχουν μηχανισμό συνγχρονισμού ίδιο με την append.

# Κεφάλαιο 8

# Ανακεφαλαίωση και Μελλοντικές Επεκτάσεις

Σε αυτή τη διπλωματική ασχοληθήκαμε με κάποιες πτυχές των ασύρματων δικτύων αισθητήρων. Στο πρώτο μέρος παρουσιάσαμε αρχικά ένα ιστορικό, τα χαρακτηριστικά κάθως και τα πλεονεκτήματα-μειονεκτήματα τέτοιων δικτύων. Ακολούθως παρουσιάμε τις εμπορικές και κάποιες επιλεγμένες πλατφόρμες, που χρησιμοποιούνται σε τέτοια δίκτυα. Συνεχίσαμε με την λεπτομερειακή ανάλυση της πλατφόρμας που χρησιμοποιήσαμε για τα πειράματα μας καθώς και του λειτουργικού TinyOS που έτρεχαν οι συγκεκριμένες πλατφόρμες. Στη συνέχειαπαρουσιάσαμε τις πειραματικές μετρήσεις παραγματικού ασύρματου δικτύου αισθητήρων. Στο δεύτερο μέρος είδαμε επιλεγμένους εξομοιωτές που μπορούν να προσφέρουν μία εικόνα της συμπεριφοράς τέτοιων δικτύων. Τελος μέσα στα πλαίσια της εξομοίωσης παρουσιάσαμε ένα μοντέλο εξομοίωσης του file system Matchbox, που χρησιμοποιεί το TinyOS.

Μελλοντικά, θα είχε ενδιαφέρον να πραγματοποιηθούν τα ίδια πειράματα σε κάποιον εξομοιωτή ασύρματων δικτύων αισθητήρων. Έτσι θα μπορέσουμε να δούμε το μέγεθος της απόκλισης των πραγματικών αποτελεσμάτων σε σχέση με αυτά της εξομοίωσης, δηλαδή θα μπορέσουμε να αξιολογήσουμε και την εγκυρότητα των αποτελεσμάτων του εξομοιωτή. Παράλληλα θα εξάγουμε συμπεράσματα για το βαθμό επιρροής του περιβάλλοντος λαμβάνοντας υπόψιν ότι τέτοια δίκτυα καλούνται να λειτουργήσουν πολλές φορές σε περιβάλλοντα αφιλόξενα.

Ενδιαφέρον θα είχε και το σενάριο της επανάληψης των πειραμάτων, αυτή τη φορά όμως με διαφορετική πλατφόρμα, π.χ. Mica2 motes. Τα αποτελέσματα θα ήταν πολύ διαφωτιστικά ως προς τις διαφορές των δύο πλατφορμών, των χαρακτηριστικών τους αλλά και της συμπεριφοράς τους κάτω από τις ίδιες συνθήκες. Με αυτό τον τρόπο θα μπορούσαμε να έχουμε δοκιμασμένα κριτήρια επιλογής για τις εκάστοτε πλατφόρμες. Ιδιαίτερο ενδιαφέρον θα είχε η επανάληψη με Micaz motes, όπου θα βλέπαμε τη διαφορά των πρωτοκόλλων IEEE 802.14.5

και των ZigBee προδιαγραφών των Micaz, με το S-Mac πρωτόκολλο που χρησιμοποιούν τα Mica motes. ΄Ετσι θα γινόταν ίσως εμφανής η διαφορά των νέων και πολλά υποσχόμενων πρωτοκόλλων με τα παλιότερα.