Ατομική διπλωματική εργασία

ΑΛΓΟΡΙΘΜΟΙ ΕΠΕΞΕΡΓΑΣΙΑΣ ΕΡΩΤΗΜΑΤΩΝ ΣΕ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

Λουκάς Τζέλης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Επιβλέποντες καθηγητές

Μποζάνης Παναγιώτης
Κατσαρός Δημήτριος

ΣΕΠΤΕΜΒΡΙΟΣ 2011

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα καθηγητή μου κ Μποζάνη Παναγιώτη για την πολύτιμη βοήθεια του. Επίσης ευχαριστώ την οικογένειά μου και όσους με στήριξαν κατά την διάρκεια των σπουδών μου.

Περίληψη

Στα πλαίσια της διπλωματικής εργασίας μου ασχολήθηκα με αλγορίθμους επεξεργασίας επερωτημάτων σε δίκτυα αισθητήρων. Τα δίκτυα αισθητήρων αποτελούν ένα πεδίο έρευνας και ανάπτυξης που αυξάνεται όλο και περισσότερο τα τελευταία χρόνια. Οι αισθητήρες πλέον βρίσκουν εφαρμογή σε ποικίλες κατηγορίες όπως η βιομηχανία, το περιβάλλον και οι στρατιωτικές επιχειρήσεις. Ταυτόχρονα εμφανίζονται πολλές προκλήσεις σε επίπεδο έρευνας και εφαρμογής. Γίνεται μια προσπάθεια ανάπτυξης hardware και software τα οποία θα προσφέρουν αποδοτικά δίκτυα. Αυτό περιλαμβάνει πρώτον την προσαρμογή τεχνολογιών που χρησιμοποιούνται ήδη σε άλλα δίκτυα και δεύτερον την δημιουργία τεχνολογιών που εφαρμόζονται αποκλειστικά σε δίκτυα αισθητήρων.

Στην εργασία αυτή παρουσιάζεται η λειτουργία των αισθητήρων και των δικτύων που αυτοί περιλαμβάνονται. Αναλύονται τα χαρακτηριστικά, η δομή και η εφαρμογή τους.

Επίσης γίνεται μεγάλη αναφορά στα πρωτόκολλα δρομολόγησης καθώς και σε αλγορίθμους επεξεργασίας ερωτημάτων(query) και αναφέρονται πολλά συστήματα που τους ενσωματώνουν.

Τέλος παρουσιάζεται ένας αλγόριθμος επεξεργασίας, που δημιουργήθηκε από την μελέτη μου πάνω στα δίκτυα αισθητήρων, και τα πειράματα που έγιναν πάνω σε αυτόν.

Συντομογραφίες & λέξεις κλειδιά

WSN	-	Wireless Sensor Networks(ασύρματα δίκτυα αισθητήρων)
Query	-	επερώτημα
Sensor	-	αισθητήρας, συχνά αναφέρεται και ως κόμβος
BS	-	Base Station(σταθμός βάσης για WSN και γενικά ρίζα, gateway)
Leader	-	κόμβοι που έχουν συγκεκριμένες αρμοδιότητες σε ένα δίκτυο(χρησιμοποιείται και ο όρος proxy)
ADT	-	αφηρημένος τύπος δεδομένων

Περιεχόμενα

Κεφάλαιο 4
Query processing

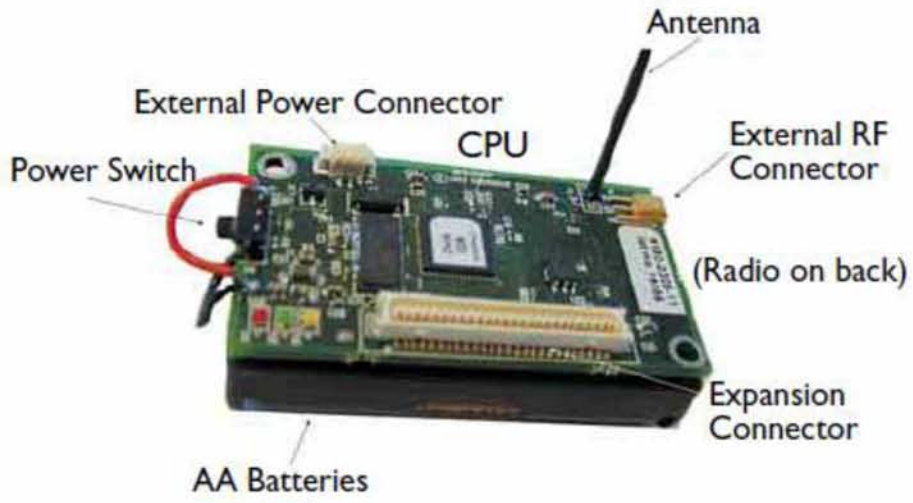
Κεφάλαιο 5
Simulation

Κεφάλαιο 1

1.1 Αισθητήρας

Αισθητήρας είναι μια συσκευή που μετράει μια φυσική ποσότητα από το περιβάλλον και την μετατρέπει σε σήμα το οποίο μπορεί να επεξεργαστεί από υπολογιστικά συστήματα. Πλέον οι αισθητήρες έχουν ευρεία χρήση και μεγάλο φάσμα εφαρμογών. Ανάλογα με το είδος του αισθητήρα τα δεδομένα που μπορεί να συλλέξει είναι μεταβλητές του περιβάλλοντος(υγρασία, θερμοκρασία) παράμετροι συστημάτων (ροή, κίνηση) και πολλά ακόμα δεδομένα. Το μέγεθος του είναι συνήθως μικρό από μερικά εκατοστά έως και μερικά χιλιοστά.

Ένας αισθητήρας αποτελείται από τέσσερα βασικά στοιχεία: από έναν ή περισσότερους αισθητήρες-ανιχνευτές(τις μονάδες δηλαδή που συλλέγουν τα δεδομένα), από τη μονάδα ελέγχου-επεξεργασίας που είναι υπεύθυνη να συντονίζει τις ενέργειες του αισθητήρα και να ελέγχει τη λειτουργία του, την μονάδα επικοινωνίας που είναι υπεύθυνη για την επικοινωνία(συνήθως ασύρματη) με άλλους αισθητήρες, την μνήμη που αποθηκεύονται τα δεδομένα που συλλέγουν οι αισθητήρες και φυσικά μια μονάδα που παρέχει ενέργεια για να λειτουργεί ο αισθητήρας.



Εικόνα 1

Ανιχνευτές

Οι ανιχνευτές αφού συλλέξουν μια φυσική ποσότητα την μετατρέπουν από συνεχές αναλογικό σήμα σε ψηφιακό με έναν analog-to-digital converter και στη συνέχεια στέλνεται στον ελεγκτή για περεταίρω επεξεργασία. Ένας αισθητήρας θα πρέπει να είναι μικρός σε μέγεθος, να καταναλώνει πολύ χαμηλή ενέργεια, να λειτουργεί για μεγάλο φάσμα τιμών , να είναι αυτόνομο, να λειτουργεί χωρίς επιτήρηση και να προσαρμόζεται στο περιβάλλον. Συνήθως χρειάζεται ρεύμα 1.2-3.7 volts.

Οι αισθητήρες μπορούν να κατηγοριοποιηθούν σε τρείς κατηγορίες: παθητικούς omni-directional, παθητικούς, narrow-beam και ενεργούς. Οι παθητικοί συλλέγουν δεδομένα χωρίς να αλληλεπιδρούν με το περιβάλλον. Συνήθως έχουν αυτόνομη παροχή ρεύματος είναι δηλαδή ενεργειακά αυτόνομοι. Οι ενεργοί από την άλλη αλληλεπιδρούν με το περιβάλλον(πχ τα ραντάρ και τα σόναρ) και χρειάζονται περισσότερη ενέργεια την οποία αντλούν από κάποια πηγή. Οι narrow-beam αισθητήρες

έχουν γνώση της κατεύθυνση που παρακολουθούν(όπως οι κάμερες) ενώ οι omni-directional δεν έχουν καμία γνώση του περιβάλλοντος στο οποίο βρίσκονται.

Ελεγκτής

Ο ελεγκτής επεξεργάζεται τα δεδομένα που συλλέγει ο αισθητήρας και ελέγχει τη συντονισμένη λειτουργία των τμημάτων του αισθητήρα. Είναι επίσης υπεύθυνος και για την ρύθμιση της ενέργειας που χρησιμοποιείται σε κάθε λειτουργία. Συνήθως είναι ένας μικροελεκτής ή μικροεπεξεργαστής(σε λίγες περιπτώσεις) όπως αυτοί που χρησιμοποιούνται στα ενσωματωμένα συστήματα. Ή χρήση αυτών τον εξαρτημάτων είναι κατάλληλη λόγω του χαμηλού τους κόστους, την χαμηλή κατανάλωση ενέργειας, την ευελιξία στη συνεργασία με άλλες συσκευές και του εύκολου προγραμματισμού τους.

Μνήμη

Η μνήμη συνήθως είναι ενσωματωμένη στον μικρο-ελεγκή ή ξεχωριστή μνήμη flash και πολύ σπάνια ξεχωριστή μνήμη Ram. Οι μνήμες flash είναι οι πιο διαδεδομένες λόγω του χαμηλού τους κόστους και της χωρητικότητας τους. Η μνήμη χωρίζεται σε δύο εφαρμογές: πρώτον για να αποθηκεύει δεδομένα από της μετρήσεις και δεύτερον σαν μνήμη που χρειάζεται για να προγραμματιστεί η συσκευή. Επίσης στη μνήμη υπάρχουν και δεδομένα για την επικοινωνία με το υπόλοιπο δίκτυο(identification data).

Πομποδέκτης

Οι αισθητήρες συνήθως κάνουν χρήση του ISM band κυρίως γιατί είναι δωρεάν και επειδή χρησιμοποιείται παγκοσμίως. Οι πιθανές επιλογές των ασύρματων μέσων μετάδοσης είναι ραδιοσυχνότητες (RF), οπτικής επικοινωνίας (Laser) και Υπέρυθρες. Τα λέιζερ απαιτούν λιγότερη ενέργεια, αλλά χρειάζεται οπτική επαφή για την επικοινωνία και είναι ευαίσθητα στις ατμοσφαιρικές συνθήκες. Οι υπέρυθρες, όπως τα λέιζερ, δεν χρειάζονται κεραία, αλλά η χωρητικότητας μετάδοσης τους είναι περιορισμένη. Το πιο κατάλληλο είδος επικοινωνίας είναι οι ραδιοσυχνότητες. Συνήθως χρησιμοποιούνται οι δωρεάν συχνότητες: 173, 433, 868, και 915 Mhz και 2.4 Ghz. Οι πομποδέκτες έχουν τέσσερις καταστάσεις transmit, receive, idle, and sleep. Οι τελευταίοι πομποδέκτες πραγματοποιούν αυτές τις μεταβάσεις σχεδόν αυτόματα. Επίσης επειδή η κατανάλωση ενέργειας σε κατάσταση idle είναι σχεδόν ίδια με αυτήν σε receive είναι πιο αποδοτικό ο πομποδέκτης να κλείνει όταν δεν χρησιμοποιείται.

1.2 Χαρακτηριστικά αισθητήρων

Εύρος. Τα όρια στα οποία η συσκευή λειτουργεί αξιόπιστα.

Ακρίβεια. Η εγγύτητα της τιμής εξόδου προς τη τιμή εισόδου.

Σφάλμα. Η διαφορά ανάμεσα στη μετρούμενη τιμή και τη πραγματική τιμή.

Ανοχή. Το μέγιστο σφάλμα που μπορεί να δημιουργήσει ο αισθητήρας.

Διακριτική ικανότητα. Η μικρότερη αλλαγή τιμής εισόδου που μπορεί να ανιχνεύσει.

Ευαισθησία. Η σχέση της αλλαγής εξόδου προς τη αλλαγή εισόδου, είναι ίση με τη διαφορά των

τιμών της εξόδου προς τη διαφορά των αντίστοιχων τιμών εισόδου.

Βαθμονόμηση. Η βαθμολόγηση της κλίμακας σε μονάδες.

Νεκρή ζώνη. Το μέγιστο ποσό αλλαγής της εισόδου που δεν επιφέρει αλλαγή στην έξοδο.

Γραμμικότητα. Ο βαθμός στον οποίο η γραφική παράσταση της εξόδου προσεγγίζει ευθεία ως προς την είσοδο του αισθητήρα.

Απόκριση. Ο χρόνος που απαιτείται για να λάβει τη τελική τιμή η έξοδος.

Καθυστέρηση. Η καθυστέρηση της αλλαγής της εξόδου ως προς την είσοδο.

Ευστάθεια. Η μεταβολή της εξόδου σε μεγάλη χρονική περίοδο, χωρίς μεταβολή της εισόδου και των συνθηκών.

Υστέρηση. Η διαφορά στην έξοδο όταν η κατεύθυνση της μεταβολής της εισόδου αντιστραφεί.

Επαναληψιμότητα. Η παραγωγή του ιδίου αποτελέσματος, σε διαφορετικές χρονικές στιγμές, με την ίδια είσοδο.

Ολίσθηση. Η μεταβολή των χαρακτηριστικών του αισθητήρα με το χρόνο και το περιβάλλον.

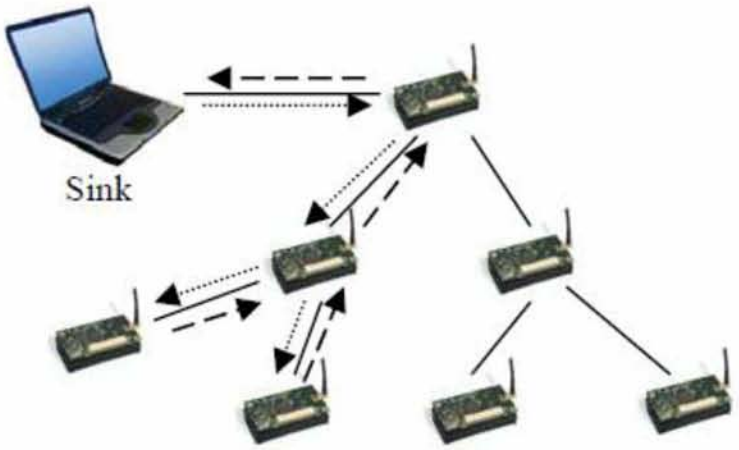
Στατικό σφάλμα. Σταθερό σφάλμα σε όλο το εύρος λειτουργίας, το οποίο μπορεί να αντισταθμιστεί.

Χρόνος λειτουργίας. Ο εκτιμώμενος χρόνος λειτουργίας στα πλαίσια των προδιαγραφών του.

1.3 Δίκτυα αισθητήρων.

Ένα δίκτυο αισθητήρων αποτελείται από αυτόνομους αισθητήρες που έχουν κατανεμηθεί στο χώρο για την παρακολούθηση φυσικών ή περιβαλλοντικών συνθηκών, όπως η θερμοκρασία, ο ήχος, δόνηση, η πίεση, κίνηση και που αποστέλλουν τα δεδομένα σε ένα κεντρικό σημείο. Τα πιο σύγχρονα δίκτυα λειτουργούν αμφίδρομα, που επιτρέπει στους αισθητήρες να αλληλεπιδρούν με το περιβάλλον. Η ανάπτυξη των ασύρματων δικτύων αισθητήρων υποκινήθηκε από στρατιωτικές εφαρμογές, όπως η επιτήρηση στο πεδίο της μάχης. Σήμερα, όμως τα δίκτυα αισθητήρων χρησιμοποιούνται σε πολλές βιομηχανικές και εμπορικές εφαρμογές, όπως η βιομηχανική διαδικασία παρακολούθησης και ελέγχου, ιατρικά μηχανήματα παρακολούθησης, και ούτω καθεξής.

Τα δίκτυα αισθητήρων αποτελούνται από κόμβους που μπορεί να είναι από εκατοντάδες μέχρι και χιλιάδες. Κάθε κόμβος μπορεί να επικοινωνεί με έναν ή και περισσότερους κόμβους και κάποιοι με ένα σημείο εισόδου στο σύστημα(gateway)



Εικόνα 2

1.4 Εφαρμογές δικτύων αισθητήρων

Παρακολούθηση χώρου. Είναι η πιο κοινή χρήση των δικτύων αυτών, με ευρεία εφαρμογή σε στρατιωτικές δραστηριότητες, για έλεγχο παραβίασης χώρων, για περιβαλλοντικές μετρήσεις, για κυκλοφορία οχημάτων κτλ.

Μόλυνση του αέρα και του νερού. Υπάρχουν συστήματα τα οποία ελέγχουν για συγκεκριμένα αέρια και ουσίες που προκαλούν μόλυνση, κυρίως σε πόλεις και σε περιοχές που υπάρχει βιομηχανία.

Ανιχνευτές πυρκαγιών σε δάση. Οι αισθητήρες αυτοί ελέγχουν την θερμοκρασία, την υγρασία και την ύπαρξη κάποιων αερίων και σε περίπτωση πυρκαγιάς ενημερώνουν την πυροσβεστική. Επίσης συλλέγουν σημαντικά δεδομένα για την εστία της πυρκαγιάς καθώς και για την κατεύθυνση και την ταχύτητα με την οποία εξαπλώνεται η φωτιά.

Έλεγχος θερμοκηπίων. Αισθητήρες που ελέγχουν θερμοκρασία, υγρασία και φώς ειδοποιούν σε περίπτωση που οι τιμές αγγίξουν κάποιο κατώφλι ή ακόμα και ξεκινούν αυτόματες διαδικασίες όπως αυτόματο πότισμα, ρύθμιση φωτός.

Πρόβλεψη κατολισθήσεων. Αισθητήρες που ελέγχουν ελάχιστες κινήσεις του εδάφους μπορούν να προειδοποιήσουν εγκαίρως για περιοχές που υπάρχει κίνδυνος κατολίσθησης.

ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΜΗΧΑΝΙΑ ΚΑΙ ΤΗ ΠΑΡΑΓΩΓΗ

Έλεγχος κατάστασης μηχανημάτων. Αισθητήρες που ελέγχουν την κατάσταση ενός μηχανήματος και σε περίπτωση εντοπίζεται κάποια συνθήκη που μπορεί να προκαλέσει προβλήματα ή και να καταστρέψει τη μηχανή ή εξάρτημα αυτής, αντιμετωπίζεται.

Έλεγχος ύδρευσης και υδάτων. Πλέον με αισθητήρες μπορεί να ελεγχθεί μεγάλο μέρος της παραγωγικής διαδικασίας έτσι ώστε ο αγρότης να έχει πλήρη γνώση ανά πάσα στιγμή των περιβαλλοντικών συνθηκών(θερμοκρασία, υγρασία) και να μπορεί να αυτοματοποιεί σημαντικές

διαδικασίες όπως το πότισμα με σημαντική μείωση των απωλειών.

Δομικός και αστικός έλεγχος. Με μια πληθώρα συστημάτων είναι δυνατός η παρακολούθηση στο εσωτερικό των κτιρίων αλλά και σε κόμβους συγκοινωνίας(δρόμους, γέφυρες, τούνελ). Έτσι μπορούν να γίνουν μετρήσεις πολύ εύκολα ενώ παλαιότερα απαιτούνταν η παρουσία φυσικού προσώπου για όλες σχεδόν τις μετρήσεις. Επίσης οι μετρήσεις γίνονται συνεχώς και όχι μια φορά την εβδομάδα ή μια φορά το μήνα.

1.5 Χαρακτηριστικά

Ορισμένα από τα ζητήματα που εμφανίζονται στα δίκτυα αισθητήρων είναι:

- η κατανάλωση ρεύματος που βασίζεται συνήθως σε μπαταρίες ή σε μερικές περιπτώσεις σε ηλιακούς συλλέκτες
- η μεταφερσιμότητα των αισθητήρων
- η δυνατότητα αλλαγής της τοπολογίας του δικτύου
- τοποθέτηση και συντήρηση σε δύσκολο έως και εχθρικό περιβάλλον
- πιθανές βλάβες σε αισθητήρες
- προβλήματα στην επικοινωνία
- η ύπαρξη και συνεργασία διαφορετικών κόμβων-αισθητήρων σε ένα δίκτυο
- περιορισμοί πόρων(μνήμη, ενέργεια, επικοινωνία, υπολογιστική δύναμη)
- η δυνατότητα ύπαρξης πολύ μεγάλου αριθμού κόμβων
- απλοποίηση της χρήσης
- αυτοματοποίηση και αυτόνομη λειτουργία
- προσαρμογή σε αλλαγές που μπορεί να γίνουν στο περιβάλλον
- ασφάλεια

Τα παραπάνω είναι χαρακτηριστικά τα οποία είναι κοινά σε όλα τα δίκτυα αισθητήρων και ο σχεδιαστής θα πρέπει να έχει λύσεις για το κάθε ζήτημα από τα παραπάνω καθώς μια λάθος εκτίμηση μπορεί να μειώσει σημαντικά την απόδοση του δικτύου είτε επειδή τα δεδομένα που θα συλλέγονται μπορεί να είναι ανακριβή ή και λανθασμένα, είτε επειδή θα καταναλώνονται πόροι που θα μπορούσαν να χρησιμοποιηθούν για άλλες λειτουργίες.

1.6 Σχεδιασμός δικτύου

Η σχεδίαση ενός τέτοιου δικτύου θα πρέπει να είναι προσεκτική και προσαρμοσμένη στο έργο που καλείται να εκτελέσει αλλά και στις δυνατότητες των αισθητήρων και του δικτύου.

Έτσι για να πετύχουμε τη μέγιστη απόδοση σε ένα δίκτυο θα πρέπει να προσέξουμε τα εξής:

Ελάχιστη κατανάλωση ενέργειας. Μπορεί οι αισθητήρες να λειτουργούν με ελάχιστο ρεύμα(από μια μπαταρία) θα πρέπει να λάβουμε υπόψη όμως ότι πολλές φορές αυτό μπορεί να σημαίνει ότι αν τελειώσει η μπαταρία είναι πολύ δύσκολο ή και αδύνατον να αλλαχθεί. Οι κόμβοι μπορεί να είναι τοποθετημένοι σε σημεία που η πρόσβαση είναι αδύνατη ή ασύμφορη. Επίσης ακόμα και αν η αντικατάσταση της πηγής ρεύματος είναι εύκολη χάνεται χρόνος μέχρι αυτή να πραγματοποιηθεί, χρόνος που θα μπορούσε να χρησιμοποιηθεί για εργασία. Σύμφωνα με τον νόμο του Moore, το κόστος ενέργειας ανά κύκλο CPU θα μειώνεται αφού τα τρανζίστορ μειώνουν το μέγεθος τους και χρησιμοποιούν όλο και λιγότερη τάση. Από την άλλη, οι νόμοι της φυσικής και οι τάσεις στον σχεδιασμό μπαταριών δείχνει ότι η επικοινωνία θα συνεχίσει να είναι απαιτητική.

Ελάχιστη χρήση υπολογιστικής ισχύς. Είναι αυτονόητο ότι οι αισθητήρες θα πρέπει να πραγματοποιούν τον ελάχιστο αριθμό πράξεων για τους υπολογισμούς τους. Πρέπει να λάβουμε υπόψη την περιορισμένη υπολογιστική ικανότητα καθώς επίσης και τους περιορισμούς που έχουμε στην παροχή ρεύματος.

Ελάχιστος αριθμός κόμβων. Είναι φυσικό να επιθυμούμε όσο το δυνατόν λιγότερους κόμβους στο δίκτυο μας. Περισσότεροι κόμβοι σημαίνει μεγαλύτερο κόστος(τοποθέτησης και συντήρησης) και πιο προσεκτικό σχεδιασμό ολόκληρου του δικτύου.

Ευέλικτη δομή. Το δίκτυο θα πρέπει να είναι σχεδιασμένο έτσι ώστε να μπορεί να αντεπεξέλθει σε οποιαδήποτε αλλαγή στη δομή του. Αν υπάρξει βλάβη σε έναν κόμβο θα πρέπει το σύστημα να συνεχίσει να λειτουργεί κανονικά ή τουλάχιστον να λειτουργεί όσο καλύτερα γίνεται στις νέες συνθήκες. Επίσης θα πρέπει να προσαρμόζει τη λειτουργία του στη περίπτωση που επιθυμούμε να προσθέσουμε καινούργιους κόμβους στο δίκτυο. Τόσο το δίκτυο όσο και ο ίδιος ο αισθητήρας θα πρέπει να είναι σχεδιασμένοι έτσι ώστε η ενσωμάτωση να γίνεται αυτόματα και χωρίς προβλήματα. Ακόμα σε οποιαδήποτε αλλαγή της τοπολογίας ο κάθε αισθητήρας θα πρέπει να είναι σε θέση να προσδιορίσει τον εαυτό του σε σχέση με τους υπόλοιπους.

Κλιμάκωση. Το δίκτυο θα πρέπει να είναι σχεδιασμένο έτσι ώστε να μπορεί να υποστηρίξει από έναν μικρό αριθμό κόμβων έως έναν πολύ μεγάλο.

Ακρίβεια και σφάλματα. Η επικοινωνία θα πρέπει να γίνεται με τέτοιο τρόπο ώστε να εξασφαλίζεται η αξιοπιστία των δεδομένων η οποία μπορεί να μειωθεί από την ύπαρξη θορύβου και από πιθανά σφάλματα και αποτυχίας συγχρονισμού. Το σύστημα θα πρέπει να είναι ανεκτικό σε σφάλματα και να είναι σε θέση να αντιλαμβάνεται τυχόν βλάβες σε κόμβους ή προβλήματα επικοινωνίας.

Επεξεργασία δεδομένων. Η συντήρηση και διάδοση μεγάλης ποσότητας δεδομένων, είναι συνήθως πολύ ακριβή, γιαυτό το λόγο το σύστημα θα πρέπει να μπορεί να διατηρεί μόνο το απαραίτητα δεδομένα και να μεταδίδεται όσο το δυνατόν πιο περιεκτική πληροφορία γίνεται. Έτσι το δίκτυο θα πρέπει να έχει μηχανισμούς μείωσης των δεδομένων όπως summarize και aggregation ωστε η επικοινωνία και επεξεργασία να γίνεται σε πραγματικό χρόνο και αποδοτικά.

Interface. Το σύστημα θα πρέπει να παρέχει τα κατάλληλα εργαλεία στον χρήστη, για να μπορεί να αναζητεί και να επεξεργάζεται δεδομένα. Επίσης ο χρήστης θα πρέπει να είναι σε θέση να κατανοεί την δομή και λειτουργία του δικτύου και να μπορεί να κάνει αλλαγές αφαιρώντας κόμβους ή προσθέτοντας νέους, πολλές φορές με διαφορετικές δυνατότητες και λειτουργίες.

Επικοινωνία. Τα πρωτόκολλα επικοινωνίας θα πρέπει να κάνουν αποδοτική χρήση του καναλιού επικοινωνίας και να εξασφαλίζουν την γρήγορη μετάδοση με τη χαμηλότερη απώλεια πακέτων.

Ασφάλεια. Η επικοινωνία θα πρέπει να είναι τέτοια ώστε να εξασφαλίζεται η ασφαλής μεταφορά δεδομένων κάτι που είναι δύσκολο αν σκεφτεί κανείς ότι πρόσθετα headers ταυτοποίησης θα επιβάρυναν σημαντικά την μετάδοση.

1.7 Software & hardware

Η ανάπτυξη αισθητήρων έχει ως γνώμονα την μεγιστοποίηση της απόδοσης καθώς μειώνεται το κόστος και η κατανάλωση ενέργειας. Αυτό γίνεται φυσικά στοχεύοντας το hardware αλλά και το software.

Η τεχνολογία MEMS(micro eletro-mechanical systems) έχει μειώσει σημαντικά το κόστος αλλά και το μέγεθος των αισθητήρων. Πλέον οι νέοι αισθητήρες έχουν κυκλώματα χαμηλής κατανάλωσης και μηχανισμούς που τους επιτρέπουν να γνωρίζουν την ενεργειακή κατανάλωση. Η κατανάλωση ενέργειας μπορεί να μειωθεί περεταίρω με τεχνικές δυναμικής διαχείρισης ενέργειας(DPM). Η τεχνικές αυτές επιτρέπουν στον αισθητήρα να απενεργοποιεί ορισμένα τμήματα όταν αυτά δεν χρησιμοποιούνται και επιπλέον χρησιμοποιείται και δυναμική κλιμάκωση τάσης στα κυκλώματα των ενεργών τμημάτων.

Το software αν σχεδιαστεί προσεκτικά μπορεί να μειώσει σημαντικά την κατανάλωση με αποδοτικότερους αλγορίθμους. Επίσης θέτει περιορισμούς χρόνου στις εργασίες που προγραμματίζονται ώστε να μειώσει τον χρόνο υπολογισμού για την κάθε εργασία.

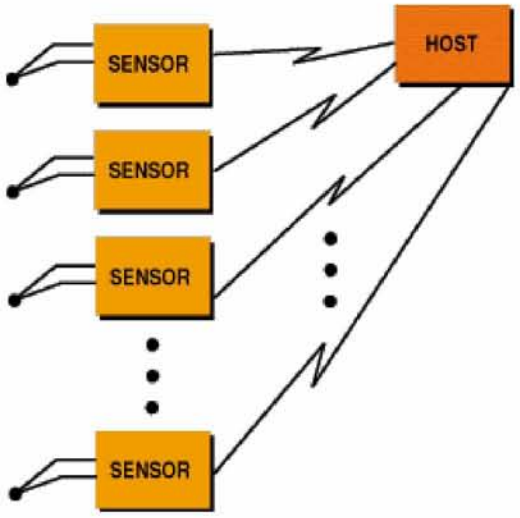
Κεφάλαιο 2

2.1 Τοπολογίες δικτύου αισθητήρων

Ενσύρματα δίκτυα

Οι τοπολογίες σε δίκτυα αισθητήρων δεν διέφεραν αρχικά σε τίποτα από τα υπόλοιπα δίκτυα αφού οι πρώτοι αισθητήρες ήταν ενσύρματοι. Οι κανόνες άλλαξαν όμως όταν εμφανίστηκαν οι ασύρματοι αισθητήρες οι οποίοι και κυριαρχούν πλέον. Έτσι νέα δεδομένα και νέες επιλογές εμφανίστηκαν όσον αφορά τις τοπολογίες. Τα παλαιότερα πρότυπα έπρεπε να τροποποιηθούν ώστε να προσαρμοστούν στα νέα δεδομένα. Ακολουθούν τρείς βασικές τοπολογίες για τα αρχικά ενσύρματα δίκτυα αισθητήρων.

2.1.1 Point-to-Point



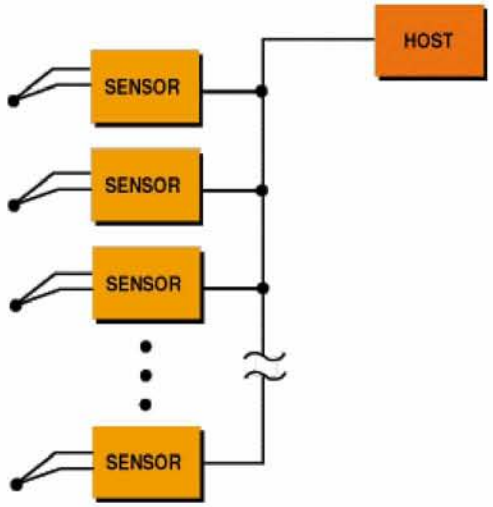
Θεωρητικά, τα συστήματα αυτά είναι τα πιο αξιόπιστα, επειδή υπάρχει μόνο ένα ενιαίο σημείο της αποτυχίας στην τοπολογία, ο host. Προβλήματα μπορούν να προκύψουν αν οι συσκευές

δημιουργήσουν φόρτο επικοινωνίας παραπάνω από τις δυνατότητες του host. Είναι όμως ένα πρόβλημα που είναι γνωστό και αντιμετωπίζεται.

Ορισμένα δίκτυα παρέχουν διαμόρφωση συχνότητας (FM) σε ενσύρματη επικοινωνία και α μεταφέρονται πολλαπλά σήματα αισθητήρων σε ξεχωριστά κανάλια FM. Μερικά πρότυπα (π.χ., το HART) υποστηρίζει πολυπλεξία των ψηφιακών σημάτων με την υπάρχουσα αναλογική καλωδίωση σε παλαιότερες εγκαταστάσεις.

Ασύρματα δίκτυα έχουν υλοποιηθεί με τη χρήση αυτής της τεχνικής. Πολλοί σχεδιαστές εφάρμοσαν απομακρυσμένα συστήματα συλλογής δεδομένων, με αυτή τη τοπολογία χρησιμοποιώντας ένα συγκεντρωτή δεδομένων που τροφοδοτούσε για τα δεδομένα σε έναν ασύρματο πομπό για να διαβιβασθεί στον host, όπου τα σήματα μετά από demultiplexing επέστρεφαν το αρχικό σήμα του αισθητήρα.

2.1.2 Multidrop Networks

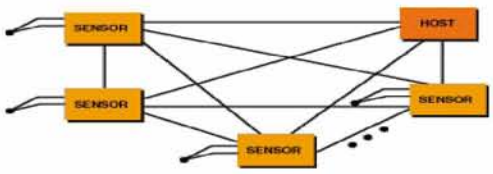


Σε ένα multidrop δίκτυο, κάθε κόμβος αισθητήρας βάζει τις πληροφορίες του σε ένα κοινό μέσο. Αυτό απαιτεί ιδιαίτερη προσοχή στα πρωτόκολλα σε εξοπλισμό και λογισμικό. Το μοναδικό καλώδιο σύνδεσης αποτελεί πιθανό σημείο αποτυχίας.

Μόλις η βιομηχανία ξεκίνησε τη μετάβαση στα multidrop δίκτυα, τα προβλήματα που σχετίζονται με την ψηφιοποίηση άρχισαν να εμφανίζονται. Με το προηγούμενο point-to-point σύστημα, η ψηφιοποίηση γινόταν στον host, όπου ένα ενιαίο ρολόι θα μπορούσε να χρησιμοποιηθεί για χρονοσφραγίδα, όταν αναλογικά σήματα από πολλαπλούς αισθητήρες έχουν αποκτηθεί. Με τη διανομή πληροφοριών που απαιτούνται για την εφαρμογή ενός δικτύου multidrop, ο συγχρονισμός έγινε ένα κρίσιμο ζήτημα σε ορισμένες εφαρμογές.

Ασύρματα συστήματα χρησιμοποιούν τους ίδιους τύπους πρωτοκόλλων για να εφαρμόσουν multidrop τοπολογίες, προσομοιώνοντας ενσύρματες συνδέσεις με RF συνδέσεις.

2.1.3 Web Networks



Σε μια τοπολογία web, θεωρητικά όλοι οι κόμβοι συνδέονται μεταξύ τους. Η σύνδεση ενός μεγάλου αριθμού αισθητήρων γίνεται πολύπλοκη, διότι όλοι οι κόμβοι πρέπει να έχουν σύνδεση με όλους τους άλλους κόμβους. Ορισμένες συνδέσεις μπορούν να εξαλειφθούν με τη χρήση repeaters και δρομολογητές κάνοντας εικονικές συνδέσεις.

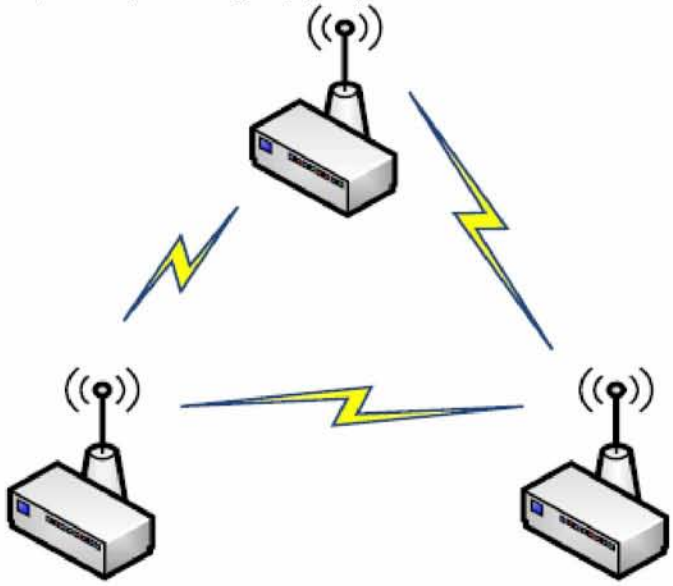
Τα πλεονεκτήματα των web networks για δίκτυα αισθητήρων είναι μεγάλα όσο αυξάνονται οι δυνατότητες τους. Συνεργαζόμενοι αισθητήρες μπορούν να σχηματίσουν ένα δίκτυο που παρέχει επαρκή χωρητικότητα και υπολογιστική ισχύ για να αντικαταστήσει τον host.

2.2 Ασύρματα δίκτυα

Οι παραπάνω τοπολογίες δεν ήταν δυνατό να καλύψουν τα νέα δεδομένα που δημιούργησαν οι ασύρματοι αισθητήρες. Χρειάστηκαν νέες, προσαρμοσμένες τοπολογίες για να αξιοποιηθούν οι δυνατότητες που προέκυψαν από τις νέες συσκευές. Ακολουθούν μερικές τοπολογίες για ασύρματα δίκτυα αισθητήρων.

2.2.1 Peer-to-peer

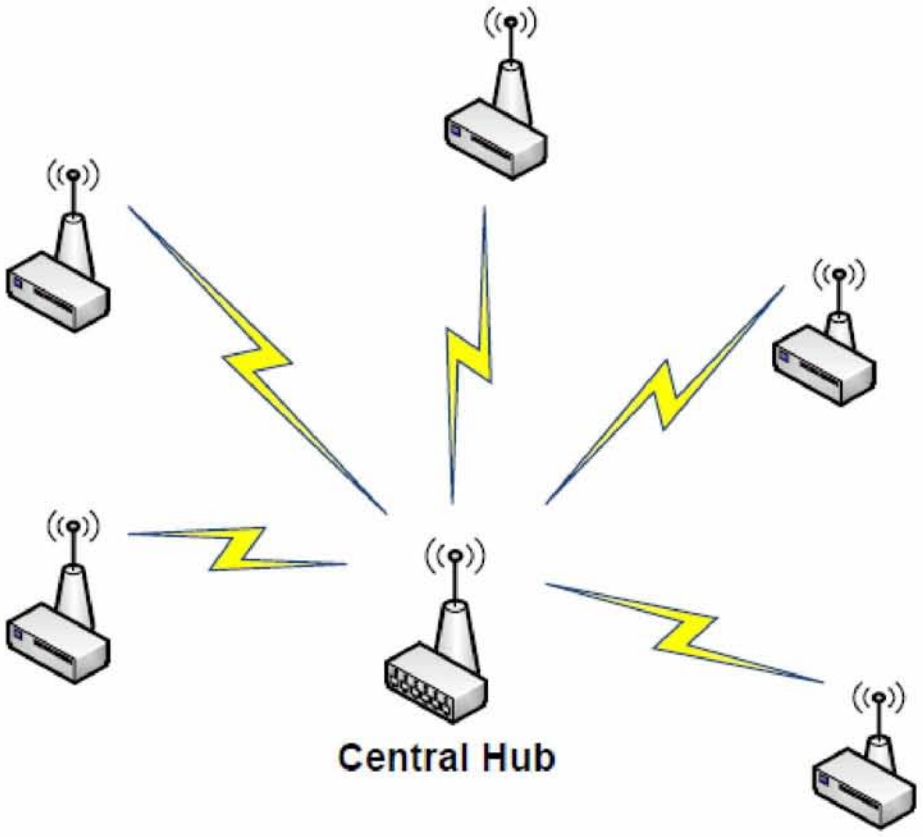
Τα Peer-to-peer δίκτυα επιτρέπουν σε κάθε κόμβο να επικοινωνεί απευθείας με ένα άλλο κόμβο χωρίς να χρειάζεται να περάσουν από έναν host. Κάθε συσκευή Peer είναι σε θέση να λειτουργήσει τόσο ως "client" και "server" για τους άλλους κόμβους στο δίκτυο.



2.2.2 Τοπολογία αστέρα (Single Point-to-Multipoint)

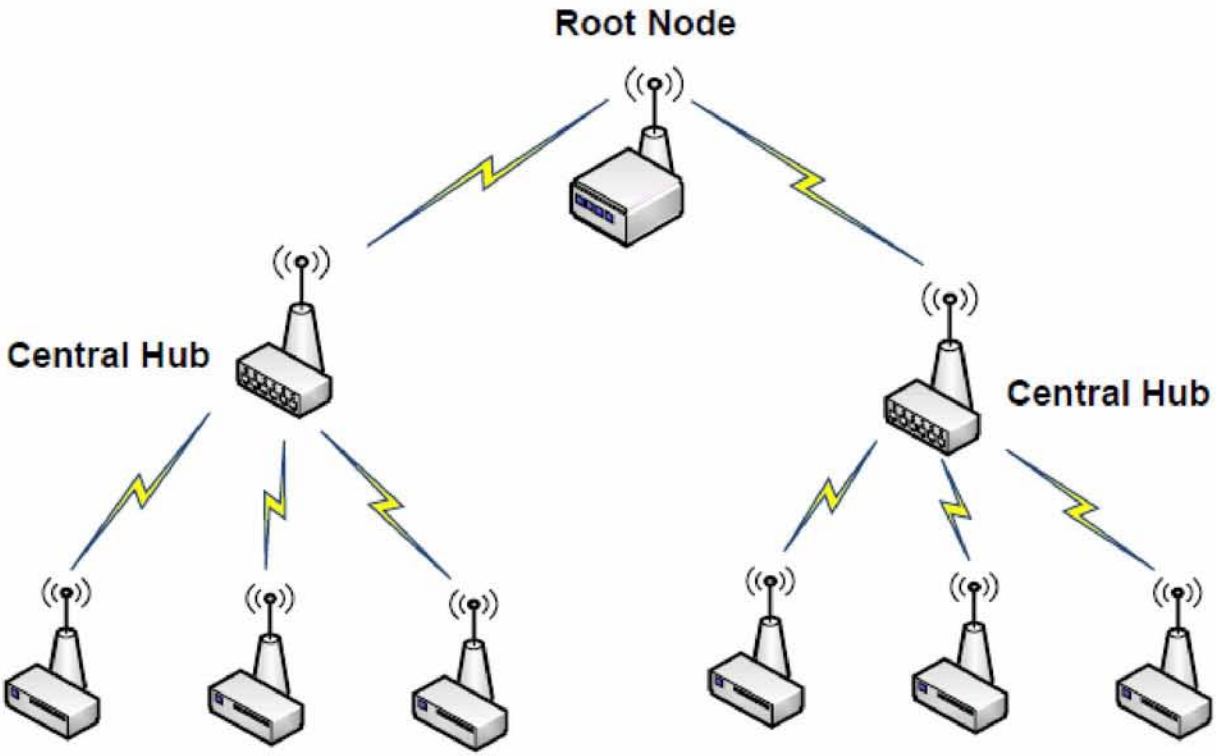
Τα δίκτυα αστέρα είναι συνδεδεμένα με ένα κεντρικό σύστημα επικοινωνίας. Οι κόμβοι

δεν μπορούν να επικοινωνούν απευθείας μεταξύ τους, όλες οι επικοινωνίες πρέπει να κατευθύνονται μέσω του κεντρικού κόμβου. Κάθε κόμβος είναι τότε ένας "client", ενώ το κομβικό σημείο είναι ο "Server"



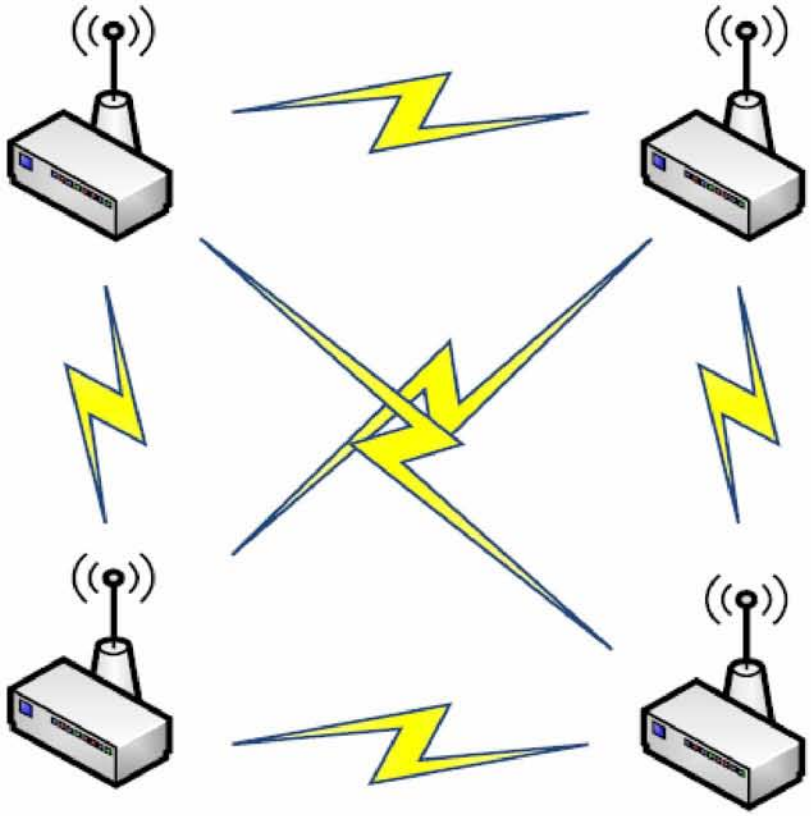
2.2.3 Tree networks

Τα δίκτυα με τη τοπολογία αυτή έχουν ένα κομβικό σημείο που ονομάζεται ρίζας ως τον κύριο διακομιστή επικοινωνίας. Ένα επίπεδο πιο κάτω από τον κόμβο ρίζας στην ιεραρχία, είναι ένας ενδιάμεσος κόμβος. Με βάση αυτόν το κόμβο δημιουργείτε ένας αστέρας με αισθητήρες.



2.2.4 Mesh networks

Τα δίκτυα Mesh επιτρέπουν τα δεδομένα να μεταβαίνουν(hop) από κόμβο σε κόμβο. Όλοι οι κόμβοι δηλαδή είναι σε θέση να επικοινωνούν μεταξύ τους αφού τα δεδομένα δρομολογούνται από κόμβο σε κόμβο μέχρι να φτάσουν στον επιθυμητό προορισμό. Εάν ένας κόμβος αποτύχει, ένας απομακρυσμένος κόμβος μπορεί ακόμα να επικοινωνήσει με οποιονδήποτε άλλο κόμβο στην εμβέλεια του, ο οποίος με τη σειρά του, μπορεί να προωθήσει το μήνυμα στον κατάλληλο προορισμό. Το μειονέκτημα αυτού του τύπου του δικτύου είναι ότι η κατανάλωση ενέργειας για τους κόμβους που εκτελούν multihop είναι γενικά υψηλότερη από ό, τι για τους κόμβους που δεν έχουν αυτή τη δυνατότητα, περιορίζοντας συχνά τη διάρκεια ζωής της μπαταρίας. Επιπλέον, όσο αυξάνονται τα βήματα(hop) αυξάνεται και ο χρόνος παράδοσης του μηνύματος.



2.2.5 Υβριδικό Star – Mesh Network

Ένα υβρίδιο δικτύου Star – Mesh παρέχει ένα ισχυρό και ευέλικτο δίκτυο επικοινωνίας, διατηρώντας παράλληλα την ικανότητα να κρατήσει την κατανάλωση ενέργειας των κόμβων-αισθητήρων στο ελάχιστο. Σε αυτή την τοπολογία του δικτύου, οι αισθητήρες με την λιγότερη ενέργεια δεν έχουν τη δυνατότητα να διαβιβάσουν μηνύματα. Αυτό επιτρέπει την ελάχιστη κατανάλωση ενέργειας. Ωστόσο, άλλοι κόμβοι του δικτύου έχουν τη δυνατότητα να κάνουν multihop, που τους επιτρέπει να διαβιβάσουν τα μηνύματα από τους κόμβους με χαμηλή ενέργεια στους άλλους κόμβους στο δίκτυο. Σε γενικές γραμμές, οι κόμβοι με την ικανότητα multihop έχουν περισσότερη ενέργεια και αν είναι δυνατόν, είναι συνδεδεμένοι σε παροχή ρεύματος.

Κεφάλαιο 3

Πρωτόκολλα επικοινωνίας και δρομολόγησης

Καθώς τα τελευταία χρόνια έγινε εφικτή η κατασκευή μικρών και φθηνών αισθητήρων, τα δίκτυα αισθητήρων άρχισαν να αποτελούνται από όλο και μεγαλύτερο αριθμό κόμβων. Έτσι δημιουργήθηκε ανάγκη για πιο ακριβή πρωτόκολλα επικοινωνίας. Ένα δίκτυο αισθητήρων μπορεί να περιέχει εκατοντάδες ή χιλιάδες κόμβους-αισθητήρες. Αυτοί οι αισθητήρες πρέπει να έχουν την ικανότητα να επικοινωνούν είτε απευθείας είτε μέσω άλλων κόμβων με μια εξωτερική βάση-σταθμό(base station).

Οι κόμβοι είναι συνήθως διάσπαρτοι σε συγκεκριμένο χώρο, που είναι μια περιοχή όπου γίνεται η ανίχνευση. Οι αισθητήρες συνεργάζονται μεταξύ τους για να παράγουν υψηλής ποιότητας πληροφορία σχετικά με το φυσικό περιβάλλον.

Κάθε αισθητήρας στηρίζει τις αποφάσεις του ανάλογα με την αποστολή του, τις πληροφορίες που έχει τη συγκεκριμένη στιγμή και τις δυνατότητες που έχει σε υπολογιστική ικανότητα, επικοινωνία και ενέργεια. Κάθε ένας από αυτούς του διάσπαρτους κόμβους-αισθητήρες έχει τη δυνατότητα να συλλέγει και να δρομολογεί τα δεδομένα , είτε σε άλλους αισθητήρες ή σε έναν εξωτερικό σταθμό βάσης. Ένας σταθμός-βαση μπορεί να είναι ένας σταθερός κόμβος ή ένας κινητός κόμβος που είναι σε θέση να συνδέει το δίκτυο αισθητήρων σε ένα υπάρχον δίκτυο ή με το Internet όπου ο χρήστης μπορεί να έχει πρόσβαση στα δεδομένα.

3.1 Στόχοι

Οι στόχοι που έχουν τα διάφορα πρωτόκολλα επικοινωνίας είναι:

- *Η κατανομή των κόμβων.* Ο τρόπος με τον οποίο οι κόμβοι κατανέμονται στο χώρο επηρεάζει την διαδικασία δρομολόγησης. Η κατανομή μπορεί να είναι τυχαία η ντετερμινιστική. Σε ντετερμινιστική κατανομή, οι αισθητήρες τοποθετούνται χειροκίνητα και τα δεδομένα κατευθύνονται μέσω προκαθορισμένων διαδρομών. Ωστόσο, σε τυχαία ανάπτυξη κόμβων οι κόμβοι-αισθητήρες είναι διασκορπισμένοι τυχαία δημιουργώντας μια υποδομή παρόμοια με αυτή των ad-hoc. Αν η δομή που προκύψει δεν είναι ομοιόμορφη τότε είναι απαραίτητη η ομαδοποίηση για να μπορεί να γίνεται μεταφορά δεδομένων με πολλαπλά hop και με την λιγότερη κατανάλωση ενέργειας.

- *Κατανάλωση ενέργειας χωρίς απώλεια ακρίβειας.* οι αισθητήρες πρέπει να χρησιμοποιούν την ενέργεια τους για να κάνουν υπολογισμούς και να μεταδίδουν δεδομένα. Ένας κόμβος παίζει διπλό ρόλο αφού είναι παραγωγός δεδομένων και δρομολογητής. Επειδή η λειτουργία του αισθητήρα συνδέεται άμεσα με το χρόνο ζωής της μπαταρίας, είναι απαραίτητο να γίνεται προσεκτική χρήση των ενεργειακών πόρων. Αν ένας κόμβος σταματήσει τη λειτουργία του λόγω έλλειψης ενέργειας, πραγματοποιούνται μεγάλες αλλαγές στη τοπολογία του δικτύου.

- *Μοντέλα αναφοράς δεδομένων.* Τα δεδομένα μπορούν να κατηγοριοποιηθούν ως time-driven (continuous), event-driven, query-driven, και υβριδικά. Τα μοντέλα time-driven είναι κατάλληλα για περιοδικούς ελέγχους, σε συγκεκριμένες χρονικές(περιοδικά) στιγμές ανοίγουν τους αισθητήρες τους, συλλέγουν δεδομένα, τα μεταδίδουν και τέλος απενεργοποιούνται ξανά. Σε event-driven και

query-driven μοντέλα οι αισθητήρες αντιδρούν άμεσα όταν ανιχνεύσουν μια τιμή ή όταν δεχτούν κάποιο query αντίστοιχα.

- *Ετερογένεια κόμβων και επικοινωνίας.* Η ύπαρξη διαφορετικών κόμβων προκαλεί πολλά τεχνικά ζητήματα. Ο κάθε κόμβος μπορεί να έχει διαφορετικό ρόλο και διαφορετικούς πόρους. Για παράδειγμα σε έναν κόμβο μπορεί να έχουμε ενσωματώσει αισθητήρες που συλλέγουν διαφορετικά δεδομένα όπως θερμοκρασία, υγρασία, πίεση. Ο κάθε αισθητήρας διαχειρίζεται διαφορετικά δεδομένα και μπορεί να έχει τον δικό του ρυθμό παραγωγής δεδομένων. Έτσι σε ένα τέτοιο σύστημα θα πρέπει να έχουμε ξεχωριστά μοντέλα αναφοράς που θα λαμβάνουν υπόψη τους περιορισμούς του κάθε αισθητήρα.

- *Ανοχή σε σφάλματα.* Μερικοί κόμβοι μπορεί να υπολειτουργούν ή να καταστραφούν λόγω έλλειψης ενέργειας, φυσικής φθοράς ή βλάβης, αλλά και από παρεμβολές από το περιβάλλον. Όταν γίνει αυτό τα πρωτόκολλα θα πρέπει να εξασφαλίσουν εναλλακτικές διαδρομές. Αυτό μπορεί να σημαίνει ότι θα χρειαστεί και αναπροσαρμογή της ισχύς μετάδοσης αλλά και του ρυθμού μετάδοσης ώστε το δίκτυο να είναι αποδοτικό.

- *Απότομη κλιμάκωση.* Σε ένα event-driven μοντέλο μεγάλος αριθμός κόμβων μπορεί να είναι απενεργοποιημένος και ξαφνικά να ανταποκριθεί μαζικά σε κάποιο ερέθισμα. Το πρωτόκολλο θα πρέπει να είναι σχεδιασμένο ώστε να μπορεί να δεχτεί τέτοιες αλλαγές στο δίκτυο.

- *Δυναμική δικτύου.* Οι περισσότερες αρχιτεκτονικές θεωρούν τους κόμβους σταθερούς. Πολλές φορές όμως οι αισθητήρες ή ο σταθμός βάσης μπορεί κινούνται. Έτσι η δρομολόγηση προς ή από κινούμενο κόμβο προσθέτει νέα ζητήματα στη διαχείριση ενέργειας και bandwith. Επίσης το αντικείμενο παρακολούθησης μπορεί να είναι στατικό ή δυναμικό. Στατικό είναι για παράδειγμα ο έλεγχος για φωτιά σε ένα δάσος και δυναμικό η παρακολούθηση ενός κινούμενου αντικειμένου.

- *Μέσα επικοινωνίας.* Σε multi-hop δίκτυα αισθητήρων η επικοινωνία γίνεται συνήθως ασύρματα. Αυτό σημαίνει ότι οι αισθητήρες έχουν να αντιμετωπίσουν επιπλέον τα προβλήματα ενός ασύρματου καναλιού.(παρεμβολές, εξασθένηση σήματος). Η επικοινωνία των αισθητήρων απαιτεί συνήθως 1-100 kb/s. Έτσι είναι απαραίτητος ο σχεδιασμός ενός ελεγκτή μέσου επικοινωνίας (medium access control). Ένας MAC μπορεί να χρησιμοποιεί TDMA ή το λιγότερο απαιτητικό(σε ενέργεια) CSMA ή ακόμα και bluetooth.

- *Συνδεσιμότητα.* Τα δίκτυα αισθητήρων έχουν συνήθως μεγάλη πυκνότητα που σημαίνει ότι υπάρχει υψηλή συνδεσιμότητα ανάμεσα στους κόμβους. Όμως αυτό μπορεί να αλλάξει εάν υπάρξει αν κάποιο κόμβοι σταματήσουν να λειτουργούν. Επίσης η συνδεσιμότητα εξαρτάται από το αν η κατανομή των κόμβων είναι τυχαία ή ντετερμινιστική.

- *Συνυπολογισμός δεδομένων(aggregation).* Εάν από πολλούς αισθητήρες παράγεται παρόμοια πληροφορία, τα δεδομένα μπορούν να συγχωνευτούν ώστε να μειωθεί η πλεονάζουσα πληροφορία. Ο συνυπολογισμός δεδομένων είναι συνδυασμός πληροφορίας που προέρχεται από διαφορετικές πηγές τα οποία υποβάλλονται σε μια συνάρτηση όπως απαλοιφή διπλότυπων, εύρεση μεγίστου, εύρεση μέσου όρου.

- *QoS.* Σε ορισμένα συστήματα τα δεδομένα πρέπει να παραδοθούν σε συγκεκριμένο χρόνο αλλιώς είναι άχρηστα. Σε μερικές εφαρμογές όμως η βέλτιστη χρήση ενεργειακών πόρων είναι πιο σημαντική από την ποιότητα των δεδομένων. Έτσι όσο πέφτουν τα επίπεδα ενέργειας το πρωτόκολλο

μειώνει την ακρίβεια της πληροφορίας.

3.2 Χαρακτηριστικά

Η δρομολόγηση στα δίκτυα αισθητήρων(και ειδικά στα ασύρματα) διαφέρει σημαντικά από αυτήν σε άλλα δίκτυα ακόμα και σε δίκτυα που φαίνεται να εμφανίζουν μεγάλες ομοιότητες(όπως ad-hoc και δίκτυα κινητής τηλεφωνίας). Παρακάτω αναλύουμε τα βασικά χαρακτηριστικά που κάνουν τα δίκτυα αισθητήρων να ξεχωρίζουν από τα υπόλοιπα.

Πρώτον, λόγω του μεγάλου αριθμού των κόμβων είναι πολύ δύσκολή και καθόλου αποδοτική, η ύπαρξη ενός καθολικού σχήματος διευθύνσεων(addressing) καθώς η δημιουργία και διατήρησή των ID επιβαρύνει τρομερά το σύστημα. Επίσης στα δίκτυα αισθητήρων το σημαντικό είναι η απόκτηση της πληροφορίας και σπάνια ενδιαφέρει το ποιός παρέχει αυτή τη πληροφορία.

Δεύτερον, σε αντίθεση με τα περισσότερα δίκτυα, τα δίκτυα αισθητήρων(τουλάχιστον στην πλειοψηφία τους) απαιτούν δεδομένα από πολλές πηγές να κατευθύνονται σε συγκεκριμένο σταθμό βάσης. Αυτό όμως σημαίνει ότι η ροή δεδομένων μπορεί να γίνει με διαφόρους τρόπους.(multicast, peer to peer κτλ.)

Τρίτον, οι αισθητήρες έχουν περιορισμούς στους πόρους γιαυτό χρειάζονται προσεκτικό σχεδιασμό.

Τέταρτον, οι κόμβοι στα ασύρματα δίκτυα αισθητήρων(WSN) παραμένουν σε σταθερό σημείο(τις περισσότερες φορές) σε αντίθεση με τα υπόλοιπα ασύρματα δίκτυα, οι κόμβοι των οποίων μπορεί να αλλάζουν θέση συχνά με απρόβλεπτες και συνεχής μεταβολές στη τοπολογία.

Πέμπτον, τα δίκτυα αισθητήρων είναι σχεδιασμένα για συγκεκριμένη λειτουργία, η απαιτήσεις και ο σχεδιασμός για κάθε εφαρμογή μπορεί να διαφέρει σημαντικά.

Έκτον, η γνώση της τοποθεσίας είναι πολύ σημαντικό για τους αισθητήρες αφού οι περισσότερες μετρήσεις βασίζονται στη τοποθεσία τους. Επειδή μέθοδοι όπως το GPS δεν μπορούν να εφαρμοστούν χρησιμοποιούνται αλγόριθμοι triangulation και multilateration ώστε να υπολογίζεται η θέση προσεγγιστικά αναλύοντας την ισχύ γνωστών κόμβων.

Τέλος, επειδή η συλλογή δεδομένων συνήθως γίνεται πάνω σε κοινές συνθήκες, πολλοί κόμβοι επιστρέφουν τα ίδια αποτελέσματα. Αυτό το πλεόνασμα σε πληροφορία πρέπει να μπορεί να αντιμετωπιστεί από τα πρωτόκολλα έτσι ώστε να βελτιωθεί η κατανάλωση ενέργειας και η χρήση του bandwith

Λόγω των διαφορών αυτών, πολλοί νέοι αλγόριθμοι έχουν προταθεί για τη δρομολόγηση στα WSN. Αυτοί οι μηχανισμοί δρομολόγησης έχουν λάβει υπόψη τα εγγενή χαρακτηριστικά των WSN σε συνδυασμό με την εκάστοτε εφαρμογή και αρχιτεκτονική. Για την ελαχιστοποίηση της κατανάλωσης ενέργειας, μέθοδοι δρομολόγησης που προτάθηκαν περιέχουν γνωστές τακτικές δρομολόγησης καθώς και ειδικές τακτικές προσαρμοσμένες για τα WSN, π.χ., data aggregation, in-network επεξεργασία, ομαδοποίηση(clustering), ανάθεση διαφορετικών ρόλων σε κόμβους, και data-centric μεθόδους.

Σχεδόν όλα τα πρωτόκολλα δρομολόγησης μπορούν να ταξινομηθούν ανάλογα με τη δομή του δικτύου(network structure), όπως επίπεδα(flat), ιεραρχικά(hierarchical) ή location-based. Επιπλέον, τα πρωτόκολλα αυτά μπορούν να ταξινομηθούν σε multipath-based, query-based, negotiation-based, QoS-based, και coherent-based ανάλογα με τη λειτουργία του πρωτοκόλλου(protocol operation).

Στα flat networks όλοι οι κόμβοι παίζουν τον ίδιο ρόλο, ενώ τα ιεραρχικά πρωτόκολλα στοχεύουν στην ομαδοποίηση των κόμβων(clustering), ώστε οι cluster heads να μπορούν να κάνουν κάποια συγκέντρωση(aggregation) και μείωση(reduction) των δεδομένων με στόχο την εξοικονόμηση ενέργειας. Τα location-based πρωτόκολλα χρησιμοποιούν τις πληροφορίες που έχουν για τις τοποθεσίες για να αναμεταδώσουν τα δεδομένα στις επιθυμητές περιοχές και όχι στο σύνολο του δικτύου. Τα πρωτόκολλα που βασίζονται στο protocol operation διαφέρουν ανάλογα με τη προσέγγιση που

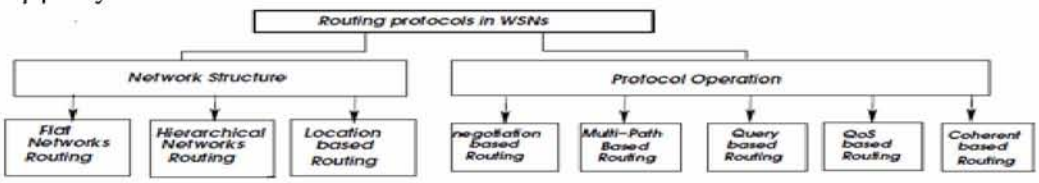
χρησιμοποιείται στο πρωτόκολλο.

Ένα πρωτόκολλο δρομολόγησης θεωρείται προσαρμοστικό αν ορισμένες παράμετροι του συστήματος μπορούν να ελεγχθούν, προκειμένου να προσαρμοστεί στις τρέχουσες συνθήκες του δικτύου και τα διαθέσιμα επίπεδα ενέργειας.

Επιπλέον τα πρωτόκολλα δρομολόγησης κατηγοριοποιούνται σε τρείς κατηγορίες proactive, reactive και υβριδικά ανάλογα με το πως μια πηγή βρίσκει τη διαδρομή για κάποιον προορισμό.

Στα proactive μοντέλα όλες οι διαδρομές υπολογίζονται πριν να χρειαστούν, ενώ στα reactive υπολογίζονται την στιγμή που θα χρειαστούν. Όταν οι κόμβοι είναι στατικοί, είναι προτιμότερο να υπάρχει ένα έτοιμο routing table και να μη χρησιμοποιούνται reactive πρωτόκολλα.

Μια άλλη κατηγορία πρωτοκόλλων δρομολόγησης είναι τα cooperative routing πρωτόκολλα. Σε αυτόν τον τύπο οι κόμβοι στέλνουν δεδομένα σε ένα κεντρικό κόμβο, όπου μπορούν να συγκεντρώνονται και να υπόκεινται σε περαιτέρω επεξεργασία, μειώνοντας έτσι το κόστος διαδρομής από άποψη ενέργειας



3.3 Πρωτόκολλα που βασίζονται στη δομή του δικτύου(network structure protocols)

3.3.1 Flat Routing

Η πρώτη κατηγορία των πρωτοκόλλων δρομολόγησης είναι τα multihop flat routing πρωτόκολλα Στα flat networks, κάθε κόμβος παίζει τον ίδιο ρόλο και όλοι οι κόμβοι-αισθητήρες συνεργάζονται από κοινού για να εκτελέσουν την εργασία. Λόγω του μεγάλου αριθμού αυτών των κόμβων, δεν είναι εφικτό να οριστεί ένα καθολικό αναγνωριστικό σε κάθε κόμβο. Αυτή η θεώρηση έχει οδηγήσει σε data-centric δρομολόγηση, όπου ο σταθμός βάσης στέλνει ερωτήματα σε ορισμένες περιοχές και περιμένει τα στοιχεία από τους αισθητήρες που βρίσκονται στις επιλεγμένες περιοχές. Εφόσον τα δεδομένα ζητούνται μέσω queries, χρειάζεται attribute-based ονομασία των δεδομένων. Μελέτες πάνω στη data-centric δρομολόγηση πχ SPIN και directed diffusion εχουν δείξει ότι είναι δυνατή η εξοικονόμηση ενέργειας μέσω της διαπραγμάτευσης των δεδομένων(data negotiation) και την εξάλειψη των περιττών δεδομένων.

- *Sensor Protocols for Information via Negotiation (SPIN)[1].* Τα πρωτόκολλα αυτά κάνουν χρήση της ιδιότητας των κόμβων σε κοντινή απόσταση με παρόμοια στοιχεία. Ως εκ τούτου είναι ανάγκη να διανείμουν μόνο τα δεδομένα που οι άλλοι κόμβοι δεν κατέχουν. Τα SPIN πρωτοκόλλα χρησιμοποιούν data negotiation και resource-adaptive αλγορίθμους. Οι κόμβοι εκχωρούν ένα υψηλού επιπέδου όνομα για να περιγράψουν πλήρως τα δεδομένα τους(meta-data) και εκτελεί διαπραγματεύσεις με τα υπόλοιπα meta-data πριν από τη μετάδοση των δεδομένων. Αυτό εξασφαλίζει ότι δεν υπάρχουν περιττά δεδομένα που αποστέλλονται σε όλο το δίκτυο. Οι κόμβοι λειτουργούν πιο αποτελεσματικά και εξοικονομούν ενέργεια, στέλνοντας δεδομένα που περιγράφουν τα δεδομένα του αισθητήρα αντί να στείλουν όλα τα δεδομένα. Ένα από τα μειονεκτήματα είναι ότι η αλλαγές στην τοπολογία επηρεάζουν πολύ την δομή καθώς ένας κόμβος γνωρίζει μόνο του γείτονες που βρίσκονται σε κοντινή απόσταση(single-hop).

- *Directed Diffusion [2].* Είναι ένα data-centric και application-aware μοντέλο με την έννοια ότι όλα τα δεδομένα που παράγονται ονομάζονται από ένας ζεύγος attribute-value. Η κεντρική ιδέα είναι να συνδυάσει τα δεδομένα που προέρχονται από διαφορετικές πηγές (in-network aggregation) με την εξάλειψη των πλεονασμάτων, ελαχιστοποιώντας τον αριθμό των μεταδόσεων. Έτσι εξοικονομείται ενέργεια. Οι αισθητήρες δημιουργούν κατευθύνσεις(gradient) πληροφοριών για την γειτονιά τους. Ο σταθμός βάσης ζητά δεδομένα προωθώντας το "ενδιαφέρον"(interests) του. Ένα interest περιγράφει μια εργασία που πρέπει να πραγματοποιηθεί στο δίκτυο. Το interest προωθείται hop-by-hop από κάθε κόμβο στους γείτονες του. Όσο το interest προωθείται οι κόμβοι κατασκευάζουν gradients που κατευθύνουν στον τελικό κόμβο. Όταν βρεθούν interests που ταιριάζουν σε gradient σχηματίζονται μονοπάτια από τα οποία επιλέγεται το καλύτερο. Ο σταθμός βάσης στέλνει περιοδικά interests. Όλοι οι κόμβοι είναι application-aware, το οποίο επιτρέπει τη διάχυση για εξοικονόμηση ενέργειας με την επιλογή εμπειρικά καλύτερων μονοπατιών και με την προσωρινή αποθήκευση και επεξεργασία δεδομένων μέσα στο δίκτυο.

- *Rumor routing [3].* Είναι μια παραλλαγή του Directed Diffusion όταν έχουμε καθορισμένα γεωγραφικά κριτήρια. Γενικά το Directed Diffusion προωθεί το query σε ολόκληρο το δίκτυο όταν δεν υπάρχουν γεωγραφικοί περιορισμοί. Όμως αρκετές φορές χρειάζεται μόνο συγκεκριμένη πληροφορία από ένα μικρό κομμάτι του δικτύου οπότε είναι περιττή η διάχυση του query σε ολόκληρο το δίκτυο. Όταν τα events είναι λίγα και τα queries πολλά είναι προτιμότερο να ενημερώνεται το δίκτυο για τα events. Η γενική ιδέα είναι να έχει κάθε κόμβος έναν πίνακα με events. Στο δίκτυο υπάρχουν πακέτα τα οποία ονομάζονται agents. Όταν σε έναν κόμβο γίνεται ένα event τότε ο κόμβος δημιουργεί έναν agent ο οποίος ταξιδεύει σε μακρινούς κόμβους και τους ενημερώνει για το event. Όταν ένα query φτάνει σε έναν κόμβο αυτός ελέγχει τον πίνακα των events και προωθεί το query στους κατάλληλους κόμβους. Η μέθοδος αυτή είναι ενεργειακά αποδοτική και έχει μεγάλη ανοχή σε σφάλματα. Το μειονέκτημα της είναι ότι είναι κατάλληλη μόνο για μικρό αριθμό event. Αν υπάρχουν πολλά events τότε το κόστος των agents και των event table είναι μεγάλο αν ορισμένα events δεν έχουν ενδιαφέρον για τον σταθμό βάσης.

- *Minimum Cost Forwarding Algorithm (MCFA) [4].* Εκμεταλλεύεται το γεγονός ότι η κατεύθυνση της δρομολόγησης είναι πάντα γνωστή: είναι προς το σταθερό εξωτερικό σταθμό βάσης. Ως εκ τούτου, ένας κόμβος δεν χρειάζεται να έχει ένα μοναδικό ID ούτε διατηρεί routing table αρκεί να έχει μια εκτίμηση του ελάχιστου κόστους μετάδοσης προς το σταθμό βάσης. Κάθε μήνυμα διαβιβάζεται από τον κόμβο στους γείτονες του. Όταν ένας κόμβος λάβει ένα μήνυμα ελέγχει αν βρίσκεται μέσα στο μονοπάτι χαμηλότερου κόστους ανάμεσα στον κόμβο προορισμό και τον σταθμό βάσης. Αν είναι τότε μεταδίδει το μήνυμα στους γείτονες του μέχρι να φτάσει τον σταθμό βάσης. Όπως αναφέρθηκε κάθε κόμβος θα πρέπει να ξέρει το ελάχιστο μονοπάτι από αυτόν προς το σταθμό-βασης(BS). Αυτό γίνεται ως εξής: Αρχικά ο BS στέλνει ένα μήνυμα με κόστος μηδέν ενώ οι κόμβοι αρχικά έχουν ελάχιστο κόστος άπειρο. Κάθε κόμβος, μετά την παραλαβή του μηνύματος που προήλθε από τον BS, ελέγχει αν η εκτίμηση στο μήνυμα συν τη σύνδεση την οποία έχει ληφθεί είναι μικρότερη από την τρέχουσα εκτίμηση. Αν ναι, η τρέχουσα εκτίμηση και η εκτίμηση στο μήνυμα ενημερώνονται. Αν το μήνυμα που λαμβάνεται ενημερωθεί, τότε γίνεται εκ νέου αποστολή. Σε αντίθετη περίπτωση, το μήνυμα καταστρέφεται.

- *Gradient-Based Routing [5].* Η βασική ιδέα στην GBR είναι να απομνημονεύσει τον αριθμό των hop, όταν το interest διαχέεται σε όλο το δίκτυο. Με αυτό, κάθε κόμβος μπορεί να υπολογίσει μια παράμετρο που ονομάζεται ύψος(height) του κόμβου, που είναι ο ελάχιστος αριθμός των hop για την εύρεση του BS. Η διαφορά μεταξύ το ύψος ενός κόμβου και του γείτονά του θεωρείται η

κατεύθυνση(gradient) σε αυτό το σύνδεσμο. Ένα πακέτο προωθείται σε μια σύνδεση με το μεγαλύτερο gradient. Ο GBR χρησιμοποιεί κάποιες βοηθητικές τεχνικές, όπως data aggregation και traffic spreading για να διαιρέσει ομοιόμορφα την κίνηση μέσω του δικτύου. Όταν πολλαπλά μονοπάτια περνούν μέσα από έναν κόμβο, που λειτουργεί ως κόμβος αναμεταδότης, ο κόμβος αυτός μπορεί να συνδυάζει δεδομένα σύμφωνα με μια ορισμένη συνάρτηση. Στην GBR, τρεις διαφορετικές τεχνικές διάδοσης δεδομένων έχουν προταθεί. *Stochastic Scheme*, όπου επιλέγεται τυχαία κάποιο gradient και στη συνέχεια 2 ή και περισσότερα hop με το ίδιο gradient. *Energy-based scheme,* όπου ένας κόμβος αυξάνει το ύψος του(height) όταν η ενέργεια του πέσει κάτω από ένα κατώφλι, ώστε να αποθαρρύνονται άλλοι αισθητήρες και να μην στέλνουν δεδομένα σε αυτόν τον κόμβο. *Stream-based scheme,* όπου νέες διαδρομές δεν δρομολογούνται από κόμβους που ανήκουν ήδη σε άλλες διαδρομές. Ο κύριος στόχος αυτών των τεχνικών είναι να αποκτήσουν μια ισορροπημένη κυκλοφορία στο δίκτυο.

- *IDSQ και CADR [6].* Στους αλγόριθμοι Information-driven sensor querying (IDSQ) και Constrained anisotropic diffusion routing (CADR), η βασική ιδέα είναι να ερωτούνται οι αισθητήρες και να δρομολογούνται τα δεδομένα της στο δίκτυο, ώστε το κέρδος πληροφορίας να μεγιστοποιείται ενώ οι καθυστερήσεις και το εύρος ζώνης να ελαχιστοποιούνται. Ο CADR διαχέει queries χρησιμοποιώντας ένα σύνολο κριτηρίων για την επιλογή των αισθητήρων που μπορούν να πάρουν τα δεδομένα. Αυτό επιτυγχάνεται με την ενεργοποίηση μόνο των αισθητήρων που βρίσκονται κοντά σε ένα συγκεκριμένο γεγονός και προσαρμόζοντας δυναμικά τα δρομολόγια δεδομένων. Η κύρια διαφορά από κατευθυνόμενη διάχυση είναι η εξέταση του κέρδους πληροφορίας(gain) σε συνδυασμό με το κόστος επικοινωνίας. Στο IDSQ, ο κόμβος που κάνει το query μπορεί να προσδιορίσει ποιός κόμβος μπορεί να παρέχει τις πιο χρήσιμες πληροφορίες με γνώμονα την εξισορρόπησης του κόστους ενέργειας. Ωστόσο επειδή ο IDSQ δεν καθορίζει τον τρόπο που το αίτημα και οι πληροφορίες δρομολογούνται μεταξύ των αισθητήρων και της BS, μπορεί να θεωρηθεί ως συμπληρωματική διαδικασία βελτιστοποίησης.

- *ACQUIRE [7].* Το ACQUIRE(ACtive QUery forwarding In sensoR nEtworks) βλέπει το δίκτυο σαν μια κατανεμημένη βάση δεδομένων όπου τα queries διαιρούνται σε sub-queries. Ο κόμβος BS στέλνει ένα query, το οποίο στη συνέχεια διαβιβάζεται από κάθε κόμβο που λαμβάνει το query. Όσο γίνεται αυτό, κάθε κόμβος προσπαθεί να απαντήσει εν μέρει στο query με την προσωρινά αποθηκευμένη πληροφορία του και στη συνέχεια να το διαβιβάσει στον επόμενο κόμβο. Όταν το query επιλυθεί εντελώς, αποστέλλεται πίσω μέσω είτε της αντίστροφης αρχικής διαδρομής ή της συντομότερης διαδρομής για το BS. Αν τα δεδομένα στην cache δεν είναι ενημερωμένα τότε ο κόμβος παίρνει πληροφορίες από του γείτονες του σε απόσταση d hop. Αν το d είναι η περίμετρος του δικτύου τότε λειτουργεί σαν flooding.

- *Energy Aware Routing[8].* Το μοντέλο αυτό επιδιώκει να αυξήσει τον χρόνο ζωής του δικτύου. Διαφέρει από το direct infusion καθώς διατηρεί ένα σετ από διαδρομές και όχι μόνο τη βέλτιστη. Οι διαδρομές αυτές επιλέγονται με βάση μια συγκεκριμένη πιθανότητα. Η τιμή της πιθανότητας εξαρτάται από το πόσο χαμηλή ενέργεια θα καταναλωθεί σε αυτό το μονοπάτι. Το πρωτόκολλο ξεκινά με ένα flooding, το οποίο χρησιμοποιείται για να ανακαλύψει όλα τα δρομολόγια μεταξύ ζευγών πηγής / προορισμού και το κόστος του, δημιουργώντας έτσι τα routing table. Οι υψηλού κόστους διαδρομές απορρίπτονται και ένα forwarding table δημιουργείται με την επιλογή των γειτονικών κόμβων με κριτήριο το κόστος.

- *Routing Protocols with Random Walks[9].* Στόχος είναι να επιτευχθεί κατανεμημένος φόρτος με την χρήση δρομολόγησης πολλαπλών διαδρομών. Στο πρωτόκολλο αυτό θεωρείται ότι οι αισθητήρες μπορούν να ανοίγουν και να κλείνουν σε τυχαίους χρόνους. Για να βρεθεί μια διαδρομή από την πηγή

στον προορισμό, οι πληροφορίες θέσης προκύπτουν από τον υπολογισμό της απόστασης μεταξύ των κόμβων χρησιμοποιώντας την κατανεμημένη ασύγχρονη έκδοση του αλγορίθμου Bellman-Ford. Ένας ενδιάμεσος κόμβος θα επιλέξει ως το επόμενο hop, τον γειτονικό κόμβο που είναι πιο κοντά στον προορισμό σύμφωνα με μια πιθανότητα.

3.3.2 Ιεραρχικό Routing

Το ιεραρχικό routing, που αρχικά χρησιμοποιήθηκε για ενσύρματα δίκτυα, έχει πολλά πλεονεκτήματα που σχετίζονται με την κλιμάκωση και την επικοινωνία. Ως τέτοια προσαρμόστηκαν για να είναι ενεργειακά αποδοτικά σε WSN. Σε μια ιεραρχική αρχιτεκτονική, κόμβοι με υψηλή ενέργεια μπορούν να χρησιμοποιηθούν για την επεξεργασία και την αποστολή των πληροφοριών, ενώ οι κόμβοι χαμηλής ενέργειας μπορούν να χρησιμοποιηθούν για την εκτέλεση της συλλογής δεδομένων. Ακολουθούν μερικά βασικά μοντέλα.

- *Πρωτόκολλο LEACH[10].* Το Leach(Low Energy Adaptive Clustering Hierarchy) είναι ένα cluster-based πρωτόκολλο, το οποίο περιλαμβάνει κατανεμημένες συστάδες. Το Leach επιλέγει τυχαία μερικούς κόμβους-αισθητήρες ως clusterheads (Chs) και εναλλάσσει τον ρόλο αυτόν ώστε να κατανέμεται ο φόρτος ομοιόμορφα. Ο clusterhead (CH) συμπιέζει τα δεδομένα που έρχονται από τους κόμβους που ανήκουν στο συγκεκριμένο cluster, και να στέλνει ένα συνολικό πακέτο στο σταθμό βάσης, προκειμένου να μειωθεί η ποσότητα των πληροφοριών που διαβιβάζεται στον σταθμό βάσης. Χρησιμοποιεί ένα TDMA/CDMA MAC για να μειώσει τις συγκρούσεις μέσα και έξω από τα clusters. Η λειτουργία του LEACH έχει δύο φάσεις. Στην φάση setup οργανώνονται τα clusters και επιλέγονται οι clusterheads. Στην φάση steady γίνεται η μεταφορά δεδομένων με τον σταθμό βάσης. Όταν επιλεχθεί ο clusterhead στέλνει ένα μήνυμα σε όλους τους κόμβους και τους ενημερώνει. Οι υπόλοιποι κόμβοι αφού λάβουνε γνώση ενημερώνουν τον κατάλληλο clusterhead οτι επιθυμούν να ανήκουν στο συγκεκριμένο cluster. Έπειτα ο clusterhead δημιουργεί ένα σχέδιο TDMA και αναθέτει σε κάθε κόμβο ένα time-slot. Στη συνέχεια το σχέδιο αυτό στέλνεται σε όλους τους κόμβους του cluster. Κατά τη διάρκεια της φάσης steady, οι αισθητήρων μπορούν να συλλέγουν δεδομένα και να τα προωθούν στον clusterhead. Ο clusterhead επεξεργάζεται τα δεδομένα και τα στέλνει στον σταθμό βάσης. Κάθε cluster χρησιμοποιεί διαφορετικούς CDMA κώδικες για να μειώσει τις παρεμβολές από κόμβους που ανήκουν σε άλλα cluster. Με τα από κάποιο διάστημα το σύστημα ξαναπερνάει στη φάση setup. Αν και το LEACH αυξάνει σημαντικά τον χρόνο ζωής του δικτύου υπάρχουν αρκετά μειονεκτήματα. Θεωρεί ότι όλοι οι κόμβοι έχουν τη δυνατότητα να επικοινωνούν με τον σταθμό βάσης και την υπολογιστική ισχύ να υποστηρίξουν διαφορετικά πρωτόκολλα MAC. Θεωρεί επίσης ότι οι κόμβοι έχουν πάντα δεδομένα να στείλουν και ότι αυτοί που είναι κοντά έχουν παρόμοια δεδομένα. Ακόμα η ιδέα της δυναμικής συστηματοποίησης επιφέρει επιπλέον κόστος στο σύστημα. Τέλος θεωρεί ότι όλοι οι αισθητήρες ξεκινούν με την ίδια ενέργεια και ότι οι clusterheads καταναλώνουν την ίδια ενέργεια με τους απλούς αισθητήρες.

- *PEGASIS [11].* Το PEGASIS(Power-Efficient Gathering in Sensor Information Systems) είναι ένα ενισχυμένο πρωτόκολλο LEACH. Η βασική ιδέα είναι οι κόμβοι να επικοινωνούν μόνο με τους γείτονες τους και να επικοινωνούν εναλλάξ με τον BS. Όταν ένας κύκλος περάσει και όλοι οι κόμβοι έχουν επικοινωνήσει με τον BS τότε η διαδικασία ξεκινάει από την αρχή. Αυτό μειώνει το κόστος σε ενέργεια καθώς η επικοινωνία κατανέμεται ομοιόμορφα. Έτσι το PEGASIS έχει δύο στόχους: να μειώσει την ενέργεια που καταναλώνεται σε κάθε κόμβο και κατά συνέπεια και συνολικά στο δίκτυο και να μειώσει το απαραίτητο bandwith μέσω της συνεργασίας. Αντίθετα με το LEACH το PEGASIS

δεν δημιουργεί clusters αλλά αλυσίδες που επικοινωνούν με τον BS μέσω πολλαπλών κόμβων. Για την δημιουργία της αλυσίδας ο κάθε κόμβος μειώνει την ισχύ εκπομπής του έτσι ώστε να επικοινωνεί μόνο με έναν κόμβο. Από τους κόμβους που σχηματίζουν αλυσίδα ένας κόμβος εναλλάξ θα κάνει υπολογισμούς και θα επικοινωνεί με τον BS. Προσομοιώσεις έχουν δείξει ότι επιτυγχάνει διπλάσιο χρόνο ζωής από το LEACH. Όμως αν και δεν δημιουργεί clusters πάλι υπάρχει φόρτος λόγω της δυναμικής τοπολογίας, αφού κάθε κόμβος πρέπει να ξέρει την ενεργειακή κατάσταση των γειτόνων του. Επίσης θεωρεί ότι κάθε κόμβος είναι σε θέση να επικοινωνεί με τον BS και πρακτικά μπορεί να χρειαστούν πολλαπλά βήματα για να φτάσει ένας κόμβος τον BS. Το PEGASIS θεωρεί ότι κάθε κόμβος γνωρίζει την θέση όλων των κόμβων του δικτύου και ότι η όλοι οι κόμβοι έχουν την ίδια διάρκεια ζωής. Ακόμα υπάρχει σημαντική καθυστέρηση για τον κόμβο που βρίσκεται στο τέλος της αλυσίδας.

- *TEEN[12] και APTEEN[13].* Τα δύο αυτά πρωτόκολλα, TEEN (Threshold-sensitive Energy Efficient sensor Network protocol) και APTEEN (Adaptive Periodic Threshold-sensitive Energy Efficient sensor Network protocol) θεωρούνται κατάλληλα για time-critical εφαρμογές. Στο TEEN οι αισθητήρες συλλέγουν δεδομένα συνεχώς αλλά η μετάδοση δεδομένων γίνεται πιο σπάνια. Οι clusterhead στέλνουν στα μέλη της συστάδας ένα ισχυρό κατώφλι(hard threshold) που είναι το κατώφλι μεταβλητής που συλλέγεται και ένα χαλαρό κατώφλι(soft threshold) που είναι μια μικρή αλλαγή στη τιμή και ενεργοποιεί την διαδικασία αποστολής δεδομένων. Το ισχυρό κατώφλι μειώνει τις εκπομπές, αφού επιτρέπει στους κόμβους να στέλνουν δεδομένα μόνο όταν αυτά βρίσκονται σε μια περιοχή ενδιαφέροντος. Το χαλαρό κατώφλι μειώνει ακόμα περισσότερο τις εκπομπές όταν υπάρχει μικρή ή μηδενική μεταβολή στα δεδομένα. Εφόσον μειώνεται η επικοινωνία μειώνεται και η ενέργεια που καταναλώνεται. Το σημαντικότερο μειονέκτημα είναι πως αν για οποιονδήποτε λόγω ο αισθητήρας δεν λάβει τις τιμές κατωφλίου δεν στέλνει δεδομένα ή στέλνει δεδομένα με παλαιότερες τιμές κατωφλίου. Ο APTEEN είναι ένα υβριδικό πρωτόκολλο που αλλάζει την περιοδικότητα ή τις τιμές κατωφλίου ανάλογα με τις απαιτήσεις του χρήστη και το είδος της εφαρμογής. Εισάγει μεταβλητές που ενδιαφέρουν τον χρήστη(attributes) ένα TDMA πρόγραμμα και έναν μετρητή(Count Time) που είναι ο μέγιστος χρόνος ανάμεσα σε δύο επιτυχής εκπομπές από τον ίδιο κόμβο. Αν ένας κόμβος ξεπεράσει το CT υποχρεώνεται να συλλέξει δεδομένα και να τα στείλει. Ο APTEEN δίνει μεγάλη ευελιξία στον χρήστη αφού μπορεί να μεταβάλλει το Count Time και τις τιμές κατωφλίου όπως επιθυμεί. Το μεγαλύτερο μειονέκτημα είναι η πολυπλοκότητα που προσθέτει το CT και τις συναρτήσεις για τις τιμές κατωφλίου.

- *MECN.* Ο Small Minimum Energy Communication Network (MECN) ορίζει μια περιοχή αναμετάδοσης για κάθε κόμβο. Η περιοχή αυτή αποτελείται από κόμβους μέσω των οποίων συμφέρει η αναμετάδοση περισσότερο απ'οτι η απευθείας μετάδοση. Η περιοχή αναμετάδοσης ενός κόμβου δημιουργείται από την ένωση με το σύνολο των περιοχών αναμετάδοσης των κόμβων, που μπορεί να φτάσει. Η κεντρική ιδέα του MECN είναι να βρεθεί ένα υπο-δίκτυο, το οποίο θα έχει τον λιγότερο αριθμό κόμβων και θα απαιτεί λιγότερη ενέργεια για την επικοινωνία ανάμεσα σε δύο κόμβους. Ωστόσο η εύρεση ενός υπο-δικτύου με λιγότερες ακμές επιβαρύνει επιπλέον τον αλγόριθμο.

- *Self Organizing Protocol.[14]* Είναι ένα πρωτόκολλο για ετερογενή συστήματα αισθητήρων. Επιπλέον οι αισθητήρες μπορεί να είναι σταθεροί ή κινούμενοι. Μερικοί αισθητήρες συλλέγουν δεδομένα από το περιβάλλον και τα διαβιβάζουν σε ένα καθορισμένο σύνολο κόμβων που λειτουργούν ως δρομολογητές. Οι Κόμβοι-Router είναι σταθεροί και αποτελούν τον κεντρικό κορμό για την επικοινωνία. Τα δεδομένα διαβιβάζονται μέσω των δρομολογητών στους πολύ πιο ισχυρούς κόμβους BS. Κάθε κόμβος-αισθητήρας θα πρέπει να είναι σε θέση να επικοινωνεί με ένα router για να είναι μέρος του δικτύου. Είναι απαραίτητη η διευθυνσιοδότηση των αισθητήρων. Οι routers

αντιλαμβάνονται τους κόμβους που τους έχουν ανατεθεί και μπορούν να δημιουργηθούν καινούργιες ομάδες ή να συγχωνευτούν αν χρειάζεται. Ο αλγόριθμος αυτός προκαλεί ένα μικρό κόστος για τη διατήρηση των πινάκων δρομολόγησης και διατηρεί μια ισορροπημένη ιεραρχία δρομολόγησης. Ένα μειονέκτημα είναι ο φόρτος που δημιουργείται από την οργάνωση της ιεραρχίας.

- *Sensor Aggregates Routing.* Ο στόχος είναι να γίνει συλλογική παρακολούθηση μιας δραστηριότητας σε ένα συγκεκριμένο περιβάλλον. Μια συνάθροιση αισθητήρων(sensor aggregate) περιλαμβάνει τους κόμβους εκείνους που ανήκουν σε μια ομάδα εκτέλεσης μιας εργασίας. Η παράμετροι της ομάδας εξαρτώνται από την φύση της εργασίας και από τους πόρους που απαιτεί. Οι αισθητήρες χωρίζονται σε ομάδες ανάλογα με την ισχύ του σήματος τους, έτσι ώστε να υπάρχει μόνο μία κορυφή(peak) ανά συστάδα. Στη συνέχεια επιλέγονται οι leaders. Το κάθε peak μπορεί να αντιπροσωπεύει έναν κόμβο ή πολλούς κόμβους. Για να βρεθούν οι leaders οι γείτονες ανταλλάσουν μεταξύ τους πληροφορίες. Εάν ένας αισθητήρας, μετά την ανταλλαγή πακέτων με όλους τους one-hop γείτονές του, διαπιστώσει ότι έχει το πιο ισχυρό σήμα δηλώνει τον εαυτό του leader. Υπάρχουν τρείς αλγόριθμοι για εργασίες παρακολούθησης: DAM, EBAM, EMLAM. Το σύστημα λειτουργεί καλά στην παρακολούθηση πολλαπλών στόχων, όταν οι στόχοι δεν παρεμβάλλονται αλλά και όταν υπάρχουν μπορεί να ανακάμψει από παρεμβολές όταν οι στόχων απομακρυνθούν.

- *VGA.* Το πρωτόκολλο Virtual Grid Architecture(VGA) είναι ένα ενεργειακά αποδοτικό μοντέλο δρομολόγησης που χρησιμοποιεί data aggregation και in-network επεξεργασία για τη μεγιστοποίηση της διάρκειας ζωής του δικτύου. Οι αισθητήρες βρίσκονται σε σταθερό σημείο και είναι τοποθετημένοι έτσι ώστε να είναι καλύπτονται κατάλληλα οι περιοχές και να υπάρχει συμμετρία. Η πράξεις συνάθροισης(data aggregation) γίνονται σε δύο επίπεδα: πρώτα τοπικά και στη συνέχεια καθολικά. Το σύνολο των clusterheads, που ονομάζεται επίσης local Aggregators (LAS), εκτελούν τοπική συνάθροιση, ενώ ένα υποσύνολο αυτών των LAs χρησιμοποιούνται για την εκτέλεση καθολικής συνάθροιση. Ωστόσο, ο καθορισμός της βέλτιστης επιλογής των καθολικών σημείων συνάθροισης, που ονομάζεται Master Aggregators (MAs) είναι NP πρόβλημα.

- *HPAR.* Το μοντέλο Hierarchical Power-aware Routing λειτουργεί ως εξής. Χωρίζει το δίκτυο σε ομάδες αισθητήρων. Οι αισθητήρες ομαδοποιούνται έτσι ώστε σε κάθε ζώνη να υπάρχουν κόμβοι που βρίσκονται σε κοντινή απόσταση. Για τη δρομολόγησης, κάθε ζώνη έχει το δικαίωμα να αποφασίσει πώς δρομολογήσει ένα μήνυμα ιεραρχικά σε όλες την υπόλοιπες ζώνες, ώστε να μεγιστοποιηθεί ο χρόνος ζωής της μπαταρίας των κόμβων στο σύστημα. Τα μηνύματα δρομολογούνται σε μια διαδρομή που έχει το μέγιστο από τα ελάχιστα αποθέματα ενέργειας και λέγεται max-min path. Η λογική είναι ότι το μονοπάτι με μεγαλύτερο απόθεμα μπορεί να είναι λιγότερο αποδοτικό από το μονοπάτι με την ελάχιστη κατανάλωση ενέργειας. ο πυρήνας του αλγορίθμου βασίζεται στην διαφορά μεταξύ της ελαχιστοποίησης της συνολικής κατανάλωσης ενέργειας και τη μεγιστοποίηση της ελάχιστης υπολειπόμενης ισχύος του δικτύου. Ως εκ τούτου, ο αλγόριθμος προσπαθεί να ενισχύσει ένα max-min μονοπάτι, περιορίζοντας την κατανάλωση ενέργειας του ως εξής. Πρώτον, ο αλγόριθμος βρίσκει το μονοπάτι με τη μικρότερη κατανάλωση ενέργειας (Pmin) με τη χρήση του αλγορίθμου Dijkstra. Δεύτερον, ο αλγόριθμος βρίσκει ένα μονοπάτι που μεγιστοποιεί την ελάχιστη υπολειπόμενη ισχύ στο δίκτυο. Ο αλγόριθμος προσπαθεί να βρει τον βέλτιστο συνδυασμό των δύο κριτηρίων. Αυτό επιτυγχάνεται με την ελάχιστη κατανάλωση ενέργειας για το μήνυμα να είναι ίση zPmin με παράμετρο $z \geq 1$, για τον περιορισμό της κατανάλωσης ενέργειας στην αποστολή ενός μηνύματος σε zPmin. Ο αλγόριθμος καταναλώνει το πολύ zPmin μεγιστοποιώντας παράλληλα την ελάχιστη υπολειπόμενη ενέργεια.

- *TTDD.* Η προσέγγιση αυτή που λέγεται Two-Tier Data Dissemination παρέχει παράδοση

δεδομένων σε πολλαπλους κινούμενους σταθμούς βάσης. Κάθε πηγή δεδομένων δημιουργεί προληπτικά μια δομή πλέγματος που χρησιμοποιείται για να διαδώσει τα δεδομένα στους BS υποθέτοντας ότι οι κόμβοι-αισθητήρες είναι σταθεροί και έχουν γνώση της τοποθεσίας. Όταν συμβαίνει ένα γεγονός, αισθητήρες γύρω του επεξεργάζονται το σήμα και ένας από αυτούς γίνεται η πηγή για την παραγωγή αναφορών. Οι κόμβοι-αισθητήρες γνωρίζουν την αποστολή τους η οποία δεν θα αλλάζει συχνά. Για τη δημιουργία της δομής του πλέγματος, μια πηγή δεδομένων να επιλέγει τον εαυτό του ως αρχικό σημείο διέλευσης του πλέγματος, και στέλνει ένα μήνυμα-ανακοίνωση των δεδομένων σε κάθε ένα από τα τέσσερα παρακείμενα σημεία διέλευσης με τη χρήση άπληστου γεωγραφικού αλγορίθμου. Όταν το μήνυμα φτάσει σε ένα κόμβο που είναι πιο κοντά στο σημείο διέλευσης (καθορίζεται στο μήνυμα), θα σταματήσει. Κατά τη διάρκεια αυτής της διαδικασίας, κάθε ενδιάμεσος κόμβος αποθηκεύει πληροφορίες για την πηγή και προωθεί προς τα εμπρός το μήνυμα σε παρακείμενα σημεία διέλευσης, εκτός από εκείνο το οποίο το μήνυμα προέρχεται. Η διαδικασία αυτή συνεχίζεται έως ότου το μήνυμα φτάσει στα σύνορα του δικτύου. Οι κόμβοι που αποθηκεύουν τις πληροφορίες των πηγών επιλέγονται ως σημεία διάδοσης. Μετά από αυτή τη διαδικασία, η δομή του πλέγματος έχει κατασκευαστεί. Χρησιμοποιώντας το πλέγμα, ένας BS μπορεί προωθήσει(flood) ένα query, το οποίο θα διαβιβαστεί προς το πλησιέστερο σημείο διάδοσης (dissemination point) σε ένα τοπικό κελί απ'όπου και θα πάρει δεδομένα. Τα δεδομένα ακολουθούν την αντίστροφη διαδρομή προς τον BS. Αν και ο TTDD είναι αποδοτικός αλγόριθμος υπάρχουν κάποια ζητήματα για τις πληροφορίες που χρειάζεται η κατασκευή του πλέγματος. Τελικά το μονοπάτι που χρησιμοποιείται είναι μεγαλύτερο από το πιο σύντομο μονοπάτι. Θεωρείται όμως ότι το συνολικό κέρδος είναι μεγαλύτερο. Ωστόσο παράγεται επιπλέον φόρτος για τη διατήρηση του πλέγματος. Τέλος το *TTDD* υποθέτει ακριβή συστήματα εντοπισμού τα οποία δεν είναι ακόμα διαθέσιμα στα WSN.

3.3.3 Location based routing protocols

Σε αυτή τη κατηγορία routing η διευθυνσιοδότηση των αισθητήρων γίνεται σύμφωνα με τη θέση τους στον χώρο. Η απόσταση δύο γειτόνων υπολογίζεται από την ισχύ των σημάτων τους. Εναλλακτικά οι αισθητήρες μπορεί να είναι εξοπλισμένοι με ένα μικρό GPS και να μπορούν να επικοινωνούν με έναν δορυφόρο. Παρακάτω αναφέρονται τα πιο σημαντικά πρωτόκολλα αυτής της κατηγορίας(location και geographic based).

- *GAF[15].* Το Geographic Adaptive Fidelity είναι ένα energy-aware πρωτόκολλο που βασίζεται στη τοποθεσία, το οποίο έχει σχεδιαστεί κυρίως για ad hoc και δίκτυα κινητής τηλεφωνίας αλλά μπορεί να χρησιμοποιηθεί και σε δίκτυα αισθητήρων. Η περιοχή του δικτύου διαιρείται αρχικά σε καθορισμένες ζώνες και σχηματίζεται ένα εικονικό πλέγμα. Μέσα σε κάθε ζώνη του πλέγματος οι αισθητήρες συνεργάζονται και εναλλάσσουν ρόλους. Κάθε φορά ένας κόμβος επιλέγεται να συλλέγει δεδομένα και να μεταδίδει ενώ οι άλλοι απενεργοποιούνται. Οι κόμβοι που σχετίζονται με ένα συγκεκριμένο σημείο του πλέγματος θεωρείται ότι έχουν το ίδιο κόστος(από άποψη δρομολόγησης). Η ιδιότητα αυτή χρησιμοποιείται ώστε να μένουν ανενεργοί οι περισσότεροι κόμβοι σε μια ζώνη και να εξοικονομείται ενέργεια. Έτσι καθώς προσθέτουμε παραπάνω κόμβους αυξάνεται και η διάρκεια ζωής του δικτύου. Το GAF έχει τρεις καταστάσεις: την εύρεση των γειτόνων(discovery), ενεργό συμμετοχή στο routing, και sleep. Ένας clusterhead όταν ανιχνεύσει ένα γεγονός, μπορεί να ζητήσει από τους αισθητήρες-κόμβους της συστάδας του να βγουν από την κατάσταση sleep και να ξεκινήσουν τη συλλογή στοιχείων. Στη συνέχεια, ο clusterhead είναι υπεύθυνος για την παραλαβή των ανεπεξέργαστων δεδομένων από άλλους κόμβους στη συστάδα του και τη διαβίβαση τους στον BS. Ο GAF αυξάνει την διάρκεια ζωής του δικτύου και συμπεριφέρεται σαν ένα κανονικό πρωτόκολλο ad-hoc από άποψη καθυστερήσεων και απώλειας πακέτων. Μπορεί επίσης να θεωρηθεί και ως ιεραρχικό

πρωτόκολλο αφού δημιουργούνται clusters, που βασίζονται όμως στη θέση του αντικειμένου στον χώρο.

- *GEAR[16].* Το Geographic and Energy Aware Routing χρησιμοποιεί πληροφορίες για τη θέση των κόμβων για να κατευθύνει τα queries σε συγκεκριμένες περιοχές, καθώς τα queries πολλές φορές περιέχουν γεωγραφικά κριτήρια. Το πρωτόκολλο χρησιμοποιεί κριτήρια με βάση την ενέργεια και γεωγραφικούς ευριστικούς αλγορίθμους(πληροφορίες που παίρνει από τους γείτονες) για να δρομολογήσει ένα πακέτο σε κάποιον προορισμό. Η κεντρική ιδέα είναι να περιορίσει τα interest που χρησιμοποιούνται στο direct diffusion, στέλνοντας τα σε μια συγκεκριμένη περιοχή αντί για ολόκληρο το δίκτυο. Αυτό το καθιστά και πιο αποδοτικό(ενεργειακά) από το DD. Κάθε κόμβος κρατά μια τιμή για το εκτιμώμενο κόστος και μία τιμή που ονομάζεται κόστος εκμάθησης(learning cost), για έναν προορισμό μέσω των γειτόνων του. Το εκτιμώμενο κόστος είναι ένας συνδυασμός της υπολειπόμενης ενέργειας και της απόσταση μέχρι τον προορισμό. Το learning cost είναι το εκτιμώμενο κόστος που αντιπροσωπεύει τη δρομολόγηση γύρω από τις "τρύπες" στο δίκτυο. Μία τρύπα ορίζεται όταν ένας κόμβος δεν έχει άλλους κοντινότερους γείτονες προς τον προορισμό, εκτός από τον εαυτό του. Αν δεν υπάρχουν τρύπες, το εκτιμώμενο κόστος είναι ίσο με το learning cost. Το learning cost που υπολογίζεται μεταδίδεται ένα hop πίσω κάθε φορά που ένα πακέτο φτάνει τον προορισμό, έτσι ώστε η διαδρομή για το επόμενο πακέτο να διαμορφωθεί. Ο αλγόριθμος έχει δύο φάσης. Στην πρώτη, προωθούνται τα πακέτα προς την επιθυμητή περιοχή, όταν ένας κόμβος λάβει το πακέτο ελέγχει αν κάποιος γείτονας είναι κοντά στον προορισμό και του προωθεί το πακέτο. Σε διαφορετική περίπτωση υπάρχει τρύπα οπότε και επιλέγεται ένας γείτονας σύμφωνα με την συνάρτηση του learning cost. Στη δεύτερη φάση, τα πακέτα προωθούνται μέσα στην επιθυμητή περιοχή. Όταν το πακέτο φτάσει την επιθυμητή περιοχή γίνεται flooding(Restricted ή recursive geographic) μέχρι να βρεθεί ο σωστός κόμβος.

- *MFR, DIR, και GEDIR[17].* Τα πρωτόκολλα αυτά χρησιμοποιούν μεθόδους που βασίζονται στην απόσταση, την πρόοδο και την κατεύθυνση. Το βασικό ζήτημα είναι οι δύο κατευθύνσεις μιας διαδρομής(forword-backwards). Ο Geographic Distance Routing (GEDIR) είναι ένας άπληστος αλγόριθμος που προωθεί το πακέτο στον γείτονα εκείνο που η απόσταση από τον προορισμό ελαχιστοποιείται. Στον αλγόριθμο DIR επιλέγεται ο γείτονας που έχει την καλύτερη κατεύθυνση (γωνία) προς τον προορισμό. Στον Most Forward within Radius αλγόριθμο επιλέγεται ο γείτονας που ελαχιστοποιεί το αποτέλεσμα $\overline{DA}.\overline{DS}$ (dot product), όπου S η πηγή και D ο προορισμός και $\overline{DA}$, $\overline{DS}$ ευκλείδειες αποστάσεις. Εναλλακτικά μπορεί το κριτήριο να είναι η μεγιστοποίηση του $\overline{SD}.\overline{SA}$.Ο αλγόριθμος σταματάει να μεταδίδει ένα πακέτο όταν κάποιος κόμβος επιστρέψει το πακέτο στον κόμβο που του το έστειλε. Στους GEDIR και MFR δεν δημιουργούνται loop ενώ στον DIR μπορεί να υπάρξουν αν δεν υπάρχει γνώση της κίνησης στο παρελθόν. Παρόμοιοι αλγόριθμοι είναι και μέθοδος 2-hop greedy και η alternate greedy μέθοδος.

- *GOAFR[18].* Ο Greedy Other Adaptive Face Routing είναι ένας γεωμετρικός αλγόριθμος για ad hoc που συνδυάζει greedy και face routing. Ο άπληστος αλγόριθμος επιλέγει πάντα τον κόμβο που βρίσκεται πιο κοντά στον προορισμό. Ωστόσο μπορεί να βρεθεί σε αδιέξοδα ή loop. Ο αλγόριθμος face routing εγγυάται επιτυχία αν υπάρχει διαδρομή μεταξύ της πηγής και του προορισμού. Το κόστος όμως του αλγορίθμου στη χειρότερη περίπτωση εξαρτάται από το μέγεθος του δικτύου.

- *SPAN[19].* Στο πρωτόκολλο αυτό επιλέγονται κάποιο κόμβοι ως συντονιστές ανάλογα με τη θέση τους. Οι κόμβοι αυτοί αποτελούν το backbone του δικτύου και προωθούν τα μηνύματα. Ένας κόμβος μπορεί να γίνει συντονιστής αν δύο κόμβοι δεν μπορούν να επικοινωνήσουν μεταξύ τους είτε απευθείας είτε μέσω δύο συντονιστών(προσβασιμότητα 3 hop)

3.4 Δρομολόγηση βασισμένη στο Protocol Operation

3.4.1 Multipath routing protocols

Σε αυτή τη κατηγορία ανήκουν πρωτόκολλα που χρησιμοποιούν πολλαπλές διαδρομές αντί για ένα απλό μονοπάτι για να αυξήσουν την απόδοση του δικτύου. Η ανοχή σε σφάλματα εξαρτάται από το αν υπάρχουν εναλλακτικές διαδρομές, σε περίπτωση που το βασικό μονοπάτι αποτύχει να παραδώσει το πακέτο. Αυτό διασφαλίζεται με το να διατηρούνται πολλαπλά μονοπάτια για μια διαδρομή, με επιπλέον κόστος σε ενέργεια και σε ροή δεδομένων στο δίκτυο. Τα μονοπάτια αυτά συντηρούνται στέλνοντας περιοδικά μηνύματα.

Ένας από τους αλγορίθμους που χρησιμοποιούνται υπολογίζει μια διαδρομή επιλέγοντας κόμβους με το μεγαλύτερο απόθεμα ενέργειας. Το βασικό μονοπάτι θα χρησιμοποιείται μέχρι η ενέργεια του να πέσει κάτω από την ενέργεια που διαθέτει ένα εναλλακτικό μονοπάτι, οπότε και το αντικαθιστά.

Ένας άλλος αλγόριθμος προτείνει την χρήση εναλλακτικών μονοπατιών περιστασιακά για να αυξηθεί ο χρόνος ζωής του δικτύου. Η λογική είναι ότι η διαδρομή με το μεγαλύτερο απόθεμα ενέργειας δεν είναι απαραίτητα και η διαδρομή με το λιγότερο κόστος. Έτσι γίνονται αλλαγές στα μονοπάτια για υπάρχει μεγιστοποίηση της ενέργειας που απομένει.

Εναλλακτικά μπορεί το πακέτο να χωριστεί σε πολλά υπο-πακέτα και να σταλεί το καθένα από ένα διαφορετικό μονοπάτι. Φυσικά αυτό δεν είναι το βέλτιστο από άποψη ενέργειας και υπάρχει μεγαλύτερη πιθανότητα να χαθεί κάποιο πακέτο.

Τέλος μπορεί να χρησιμοποιηθεί μια παραλλαγή του Direct Difussion η οποία βρίσκει μονοπάτια τα οποία επικαλύπτονται.

Η λογική πίσω από όλους αυτούς τους αλγορίθμους είναι ότι οι εναλλακτικές διαδρομές βρίσκονται πολύ κοντά με την βασική και έχουν παρόμοιο κόστος. Γενικά η χρήση πολλαπλών μονοπατιών είναι μια λύση που αποδίδει ενεργειακά και μειώνει τις αποτυχίες σε περίπτωση που το βασικό μονοπάτι δεν μπορεί να λειτουργήσει.

3.4.2 Query based routing

Σε αυτό το είδος το query ξεκινάει από έναν κόμβο του δικτύου και αν ένας κόμβος έχει την απάντηση, στέλνει τα δεδομένα σε αυτόν τον κόμβο. Όλοι οι κόμβοι έχουν έναν πίνακα με τα queries που έχουν ζητηθεί και όταν παράγουν δεδομένα που τα ικανοποιούν, τα στέλνουν στον κόμβο που προήλθε το query. Κάτι παρόμοιο γίνεται με τα interests του Direct Difussion.

Το πρωτόκολλο rumor routing χρησιμοποιεί agents οι οποίοι δημιουργούν μονοπάτια για τα events που βρίσκουν. Όταν ο πράκτορας βρίσκει ένα μονοπάτι που οδηγεί σε ένα event το αποθηκεύει. Αν βρεθεί ένα μονοπάτι για υπάρχον event που είναι βέλτιστο τότε επιλέγεται η καλύτερη λύση. Κάθε κόμβος έχει έναν πίνακα με events, που ενημερώνει όταν ένα γεγονός συμβεί. Επίσης έχει μια λίστα με γείτονες και μπορεί να κατασκευάζει πράκτορες. Ο πράκτορας έχει χρόνο ζωής συγκεκριμένο αριθμό hop και φροντίζει να συγχρονίζει τον πίνακα των events του με τους αντίστοιχους πίνακες των κόμβων που επισκέπτεται. Ένας κόμβος θα παράγει ένα query μόνο όταν ξέρει το μονοπάτι για το αντίστοιχο event. Διαφορετικά στέλνει ένα query σε τυχαία κατεύθυνση. Αν και αυτό δεν επιστρέψει αποτελέσματα μετά από κάποιο διάστημα τότε ο κόμβος χρησιμοποιεί flood για στείλει το query σε όλο το δίκτυο.

3.4.3 Negotiation based routing protocols

Τα πρωτόκολλα αυτά χρησιμοποιούν περιγραφές δεδομένων υψηλού επιπέδου, για να εξαλείψουν τα περιττά/πλεονάζοντα δεδομένα, μέσω διαπραγμάτευσης. Τα πρωτόκολλα SPIN είναι ένα παράδειγμα αυτής της κατηγορίας. Η κεντρική ιδέα είναι ότι αν με τη χρήση flood θα επιστραφούν πλεονάζοντα δεδομένα από διαφορετικούς κόμβους τα οποία κοστίζουν σε πράξεις και ενέργεια. Στα πρωτόκολλα SPIN τα δεδομένα ενός κόμβου μεταδίδονται σε όλους τους άλλους θεωρώντας ότι οι κόμβοι αυτοί είναι πιθανοί σταθμοί-βάσης. Έτσι ο στόχος είναι να εξαλειφθούν τα διπλότυπα και η πλεονάζουσα πληροφορία, με διαπραγματεύσεις πριν φτάσουν στον επόμενο κόμβο ή BS.

3.4.4 QoS-based routing

Σε αυτή τη κατηγορία το δίκτυο πρέπει να κρατάει ισορροπία ανάμεσα στη κατανάλωση ενέργειας και στην ποιότητα δεδομένων. Πιο συγκεκριμένα πρέπει να ικανοποιεί κάποιες συνθήκες όσον αφορά τις καθυστερήσεις, την ενέργεια, το bandwith κτλ.

- *SAR [21].* Ο Sequential Assignment Routing. Είναι ένα από τα πρώτα πρωτόκολλα που εισήγαγε την έννοια του QoS στη δρομολόγηση. Η αποφάσεις που λαμβάνονται εξαρτώνται από τρείς παραμέτρους: ενέργεια, QoS σε κάθε μονοπάτι, και προτεραιότητα. Για να αποφευχθούν τα σφάλματα που μπορεί να προκύψουν σε ένα μονοπάτι, χρησιμοποιούνται πολλαπλά μονοπάτια και τοπικά μοντέλα επαναφοράς μονοπατιού. Στην ουσία δημιουργείται ένα δέντρο με ρίζα την πηγή και φύλλα τους προορισμούς, έτσι προκύπτουν πολλαπλά μονοπάτια(και δημιουργούνται και οι BS). Τα μονοπάτια προκύπτουν επιλέγοντας τους κόμβους με την μεγαλύτερη ενέργεια, έτσι ώστε στο τέλος όλοι οι αισθητήρες να ανήκουν σε κάποιο μονοπάτι(multi-path tree). Ο SAR στοχεύει σε καλύτερη απόδοση ενέργειας και μεγαλύτερη ανοχή σε σφάλματα. Ο αλγόριθμος υπολογίζει ένα βάρος QoS συνδυάζοντας τα τοπικά QoS με έναν συντελεστή της προτεραιότητας του πακέτου. Στόχος είναι να μειώνεται το μέσο βάρος κατά τη διάρκεια της λειτουργίας του δικτύου. Αν αλλάξει η τοπολογία ή υπάρξει βλάβη σε μονοπάτι, τα μονοπάτια επαναπροσδιορίζονται. Περιοδικά οι BS ελέγχουν για αλλαγές στην τοπολογία. Ο SAR χρησιμοποιεί σημαντικά λιγότερη ενέργεια από πολλά μοντέλα και έχει μηχανισμούς ανάνηψης από σφάλματα. Παρόλα αυτά, ο αλγόριθμος επιβαρύνει το δίκτυο, καθώς χρειάζονται routing tables, ειδικά όταν έχουμε μεγάλο αριθμό κόμβων.

- *SPEED [20].* Το πρωτόκολλο αυτό εγγυάται παράδοση και συγκεκριμένα σε πραγματικό χρόνο. Οι κόμβοι έχουν πληροφορίες για τους γείτονες τους και χρησιμοποιούν geographic forwarding για την εύρεση μονοπατιών. Το SPEED διατηρεί συγκεκριμένη ταχύτητα διάδοσης έτσι ώστε οι εφαρμογές να μπορούν να υπολογίσουν με ακρίβεια τις καθυστερήσεις. Επίσης το πρωτόκολλο έχει μηχανισμούς για την αποφυγή συμφόρησης στο δίκτυο. Ένας κόμβος στέλνει ένα πακέτο στους γείτονες του και από τις απαντήσεις (ACK), επιλέγεται ό γείτονας που ταιριάζει περισσότερο στα κριτήρια. Έτσι υπολογίζεται και η αναμενόμενη καθυστέρηση. Ο SPEED είναι αποδοτικός από άποψη καθυστερήσεων και απωλειών. Επίσης είναι ενεργειακά αποδοτικός λόγω της απλότητας του αλγορίθμου.

3.4.5 Coherent και non-coherent processing

Στα non-coherent πρωτόκολλα οι κόμβοι επεξεργάζονται τα δεδομένα πριν τα στείλουν σε άλλους κόμβους για περαιτέρω επεξεργασία. Οι κόμβοι που κάνουν την περαιτέρω επεξεργασία λέγονται aggregators. Στα Coherent πρωτόκολλα τα δεδομένα προωθούνται στους aggregators έπειτα από ελάχιστη επεξεργασία. Στην πρώτη περίπτωση επιδιώκεται βελτιστοποίηση μεταδίδοντας λιγότερα δεδομένα ενώ στην δεύτερη, πραγματοποιώντας λιγότερες πράξεις σε κάθε κόμβο. Στην non-coherent επεξεργασία υπάρχουν τρείς φάσεις. Πρώτον, συλλογή δεδομένων και επεξεργασία. Δεύτερον, ο κόμβος δηλώνει ότι συμμετέχει σε μια εργασία. Τρίτον, επιλογή κεντρικών κόμβων. Ο κόμβος βρίσκεται στη φάση ένα μέχρι να αποφασίσει να στείλει δεδομένα οπότε και περνάει στη δεύτερη φάση όπου ανακοινώνει την απόφαση του στους γείτονες του. Στην τρίτη φάση επιλέγεται ως κεντρικός κόμβος, αυτός με την μεγαλύτερη υπολογιστική ισχύ και τα μεγαλύτερα αποθέματα ενέργειας.

- *SWE.* Το πρωτόκολλο αυτό είναι επιλέγει έναν aggregator για σύνθετους υπολογισμούς σύμφωνα με την υπολογιστική του ικανότητα και την ενεργειακή του κατάσταση. Όταν όλοι οι κόμβοι στέλνουν δεδομένα στον aggregator, το κόστος θα είναι μεγάλο. Ένας τρόπος να μειωθεί είναι να υπάρχει όριο στο πλήθος των κόμβων που μπορούν να στείλουν δεδομένα ταυτόχρονα.

Κεφάλαιο 4

Query processing

4.1 SQL & query processing

Ένα δίκτυο αισθητήρων στην ουσία αποτελεί μια κατανεμημένη βάση δεδομένων στην οποία ένας χρήστης ή μια εφαρμογή εκτελεί πράξεις για την απόκτηση στοχευμένων δεδομένων. Τα δεδομένα αυτά μπορεί να είναι ξεχωριστά για κάθε κόμβο και να μην έχουν καμία συσχέτιση, καθώς μπορεί οι αισθητήρες να παρατηρούν διαφορετικά γεγονότα ή μπορεί να βρίσκονται σε μακρινή απόσταση. Μπορεί όμως τα δεδομένα που συλλέγονται να είναι σε μεγάλο βαθμό σχετικά αν όχι όμοια. Στόχος μιας εφαρμογής είναι να λάβει όσο το δυνατόν πιο ακριβή δεδομένα με το λιγότερο κόστος για το δίκτυο και σε όσο το δυνατόν συντομότερο χρονικό διάστημα γίνεται.

Ακριβώς επειδή στο επίπεδο εφαρμογής φαίνονται σαν μια ενιαία αποθήκη, πολλά συστήματα αισθητήρων χρησιμοποιούν sql query (επερωτήματα) για την απόκτηση της επιθυμητής πληροφορίας. Φυσικά η γλώσσα που χρησιμοποιείται είναι προσαρμοσμένη στο σύστημα αισθητήρων(TinyDB & Cougar), αφού στην πραγματικότητα διαφέρει πολύ από τις βάσεις δεδομένων. Η διαφορά είναι ότι συνήθως μας ενδιαφέρουν συγκεκριμένες πληροφορίες ή καλύτερα συγκεκριμένες κατηγορίες πληροφορίας. Επίσης στα περισσότερα δίκτυα αισθητήρων υπάρχουν λιγότερες μεταβλητές, γνωρίσματα και σχεδόν πάντα μόνο ένας πίνακας(sensor). Παρακάτω αναφέρονται μερικές διαφορές της κανονικής sql με μια δηλωτική γλώσσα που χρησιμοποιείται στο TinyDB, την TinySQL.

- Στο FROM μπορεί να μπει μόνο ο sensors, για το λόγω αυτό δεν είναι και απαραίτητο.

- Στο WHERE μπορεί να υπάρχουν μόνο αριθμητικοί τελεστές και όχι λογικοί και τα SIMILAR, LIKE

- Δεν υπάρχουν φωλιασμένα query

- Υπάρχει το EPOCH DURATION που δηλωνει το πόσο συχνά θα εκτελείται το query

ακολουθεί ένα παράδειγμα σύνταξης query:

```
SELECT R.temperature
FROM SENSORS R
WHERE R.location <10
```

οι βασικές εντολές που αφορούν συνήθως τα δίκτυα αισθητήρων είναι κλασσικές εντολές sql όπως SELECT,JOIN και μια κατηγορία πράξεων γνωστή ως aggregation.

4.2 Κατηγορίες query

Υπάρχουν τέσσερις κατηγορίες query:

1. event-based queries
2. lifetime-based queries
3. snapshot queries
4. Historical queries

Στην πρώτη κατηγορία τα query περιγράφουν τις διαδικασίες που πρέπει να γίνουν όταν συμβεί κάποιο γεγονός. Για παράδειγμα: σε περίπτωση φωτιάς σε δάσος επιστρέφονται δεδομένα όπως ταχύτητα αέρα, θερμοκρασία. Τα query αυτά είναι πολύ σημαντικά καθώς δίνουν τη δυνατότητα στο σύστημα να είναι σε αδράνεια μέχρι να συμβεί κάποιο γεγονός, αντί να ελέγχει τα δεδομένα συνεχώς. Στην δεύτερη κατηγορία το query ζητάει από το σύστημα να εκτελέσει μια λειτουργία για κάποιο χρονικό διάστημα. Π.χ. Τον υπολογισμό της μέσης θερμοκρασία για τις επόμενες 5 ημέρες. Σε αυτές τις μετρήσεις δεν μας ενδιαφέρουν οι μικρο-αλλαγές αλλά η συμπεριφορά του συστήματος σε βάθος χρόνου(συνήθως οι μεγάλες εργασίες λέγονται tasks). Στην τρίτη κατηγορία, ανήκουν τα query τα οποία αφορούν δεδομένα για ένα χρονικό σημείο. Πχ. Επέστρεψε τις τιμές των αισθητήρων για τη θερμοκρασία στον δεύτερο όροφο. Στην τελευταία κατηγορία τα δεδομένα αφορούν αποθηκευμένες τιμές που έχουν συλλεχθεί σε ένα χρονικό διάστημα στον παρελθόν. πχ. Επέστρεψε την μέγιστη θερμοκρασία το 2010.

Χαρακτηριστικά δεδομένων

Streaming data. Συνήθως οι αισθητήρες παράγουν συνεχώς(σε τακτά χρονικά διαστήματα) δεδομένα χωρίς να τους ζητηθούν.

Επεξεργασία σε πραγματικό χρόνο. Τα δεδομένα συνήθως απαιτούν πληροφορία σε πραγματικό χρόνο(real time events). Επίσης συνήθως είναι προτιμότερο να επεξεργάζονται αμέσως παρά να αποθηκεύονται ακατέργαστα(raw data) λόγω των περιορισμένων αποθηκευτικών πόρων.

Σφάλματα στην επικοινωνία. Επειδή τα δεδομένα μεταφέρονται ασύρματα και πολλές φορές σε πολλά βήματα(multi-hop) περιέχουν μια πιθανότητα σφάλματος.

Αβεβαιότητα. Θόρυβος από το περιβάλλον ή δυσλειτουργία του αισθητήρα μπορεί να αλλοιώσουν τα δεδομένα.

Κριτήρια απόδοσης

Υπάρχουν δύο κριτήρια για τη απόδοση ενός συστήματος. η χρήση πόρων, δηλαδή η ενέργεια που απαιτείται για να εκτελεστεί ένα query και ο χρόνος απόδοσης, το διάστημα δηλαδή από τη στιγμή που το query θα μπει στο σύστημα μέχρι τη στιγμή που θα επιστρέψει αποτελέσματα στον χρήστη ή εφαρμογή.

4.3 Query processor

Ο Query processor δέχεται SQL επερωτήματα, επιλέγει ένα σχέδιο για την εκτέλεση, και στη συνέχεια εκτελεί το επιλεγμένο σχέδιο. Ο χρήστης ή ένα πρόγραμμα αλληλεπιδρά με τον Query processor, και ο Query processor με τη σειρά του αλληλεπιδρά με τους αισθητήρες. Ο Query processor απομονώνει τον χρήστη από τις λεπτομέρειες της εκτέλεσης: Ο χρήστης περιγράφει το αποτέλεσμα,

και Query processor καθορίζει τον τρόπο αυτό το αποτέλεσμα θα υπολογιστεί.

Στάδια επεξεργασίας

Υπάρχουν δύο στάδια επεξεργασίας επερωτημάτων. Το στάδιο της βελτιστοποίησης και το στάδιο της εκτέλεσης.

Στο στάδιο της βελτιστοποίησης ο query processor μελετάει τις παραμέτρους του επερωτήματος και προσπαθεί να επιλέξει το ταχύτερο και αποδοτικότερο πλάνο εκτέλεσης. Ανάλογα με το είδος του query, την τοπολογία του δικτύου και τους αλγόριθμους βελτιστοποίησης που χρησιμοποιεί επιλέγει πιθανά σενάρια βελτιστοποίησης:

- αν στο δίκτυο χρησιμοποιούνται index τότε επιλέγει ποιά θα χρησιμοποιηθούν
- τη σειρά με την οποία θα εκτελεστούν οι join
- με ποια σειρά θα εφαρμοστούν περιορισμοί όπως αυτοί που ορίζονται στο WHERE
- ποιοι αλγόριθμοι θα δώσουν την καλύτερη απόδοση, βασισμένος σε πληροφορίες για το κόστος και στατιστικά.

Υπάρχουν δύο είδη βελτιστοποίησης: syntax-based όπου το πλάνο που δημιουργείται εξαρτάται από την σύνταξη που έχει το αρχικό query και cost-based επιλέγεται το καλύτερο πλάνο με την βέλτιστη απόδοση για το σύστημα(λιγότερες πράξεις, λιγότερη επικοινωνία κτλ).

Στο στάδιο της εκτέλεσης ο query processor εκτελεί το πλάνο που δημιουργήθηκε στην προηγούμενη φάση. Ο στόχος είναι να εκτελέσει το σχέδιο γρήγορα, επιστρέφοντας την απάντηση στο χρήστη ή την εφαρμογή σε μικρό χρονικό διάστημα.

Query plan. Το query plan αποτελείτε από flow blocks και στην ουσία ρυθμίζει πως θα εκτελεστεί το query. Σύμφωνα με αυτό συντονίζονται οι κόμβοι που σχετίζονται με το query και πως θα κινηθεί η πληροφορία από αυτούς. Επίσης κανονίζει αν οι υπολογισμοί θα γίνουν συγκεντρωτικά ή κατανεμημένα. Στη κατανεμημένη περίπτωση περιγράφει σε πόσα επίπεδα θα γίνουν οι υπολογισμοί και πια ποσότητα πληροφορίας θα περιλαμβάνουν.

Μέθοδοι επεξεργασίας ερωτημάτων

Συγκεντρωτική

Οι κόμβοι όταν λάβουν ένα query συγκεντρώνουν τα δεδομένα που αντιστοιχούν σε αυτό(αν υπάρχουν) και τα στέλνουν στον σταθμό βάσης είτε απευθείας είτε μέσω ενδιάμεσων κόμβων, ανάλογα με την τοπολογία του δικτύου και το πρωτόκολλο δρομολόγησης. Πολλές φορές οι κόμβοι δεν έχουν ολόκληρο το πλάνο εκτέλεσης του query αλλά μόνο ένα μικρό κομμάτι που αφορά τα δεδομένα τους. Οι κόμβοι δεν κάνουν καμία επεξεργασία πάνω στα δεδομένα αλλά τα διαβιβάζουν στον σταθμό βάσης. Ο σταθμός βάσης, που συνήθως είναι ένας κόμβος με μεγαλύτερη υπολογιστική ισχύ και μεγαλύτερα ενεργειακά αποθέματα, συλλέγει τα δεδομένα από όλους τους κόμβους που έχουν στείλει αποτελέσματα. Στη συνέχεια εκτελεί τις απαραίτητες πράξεις πάνω στα δεδομένα, σύμφωνα με το πλάνο εκτέλεσης και τελικά επιστρέφει το αποτέλεσμα στον χρήστη. Το βασικότερο πρόβλημα είναι ο μεγάλος αριθμός πακέτων που ταυτόχρονα έχουν μικρό μέγεθος. Αυτό επιβαρύνει το δίκτυο με παραπάνω κίνηση, και τους κόμβους ξεχωριστά που πρέπει να παράγουν τα πακέτα.

Κατανεμημένη

Στόχος είναι να μειωθεί ο φόρτος στον σταθμό βάσης μεταφέροντας μέρος των υπολογισμών στο υπόλοιπο δίκτυο. Αυτό μπορεί να γίνει με δύο τρόπους. Ο ένας είναι οι κόμβοι να εκτελούν κάποιες πράξεις(πχ aggregate) πριν μεταδώσουν τα δεδομένα. Έτσι όχι μόνο μειώνεται ο φόρτος για τον σταθμό βάσης αλλά μειώνεται σημαντικά και η πληροφορία που μεταδίδεται στο δίκτυο χρησιμοποιώντας λιγότερο bandwith. Ο άλλος τρόπος είναι να δώσουμε σε ορισμένους κόμβους την ευθύνη να συλλέγουν δεδομένα από γείτονες τους και να κάνουν αυτοί τους υπολογισμούς για λογαριασμό τους. Με αυτόν τον τρόπο μπορεί να προκύψει καλύτερο αποτέλεσμα αφού πράξεις όπως αφαίρεση διπλότυπων και aggregate γίνονται για τα δεδομένα πολλών κόμβων. Στη συνέχεια οι κόμβοι αυτοί προωθούν τα δεδομένα που έχουν επεξεργαστεί μερικώς στον σταθμό βάσης όπου και θα υποστούν τελική επεξεργασία. Με τον τρόπο αυτό μειώνεται και η πιθανότητα σύγκρουσης μέσα στο δίκτυο. Επίσης επειδή οι κόμβοι αυτοί έχουν πληροφορία από πολλούς αισθητήρες στέλνουν συγκεντρωτικά πακέτα κάτι το οποίο είναι σαφώς πιο αποδοτικό από την αποστολή πολλών μικρών πακέτων. Αυτό γίνεται γιατί τα δεσμεύεται συνεχώς το κανάλι και μεταδίδονται συνέχεια περιττά header.

4.4 Aggregation

Οι συναρτήσεις aggregate(συνάθροιση) είναι συναρτήσεις οπού πολλές τιμές χρησιμοποιούνται για είσοδο για να σχηματίσουν μια τιμή σύμφωνα με κάποια κριτήρια. Η τιμή αυτή είναι αντιπροσωπευτική για το σύνολο των δεδομένων. Τέτοιες συναρτήσεις είναι:

avarage. Η μέση τιμή από ένα σύνολο.

count. Το πλήθος των στοιχείων σε έναν σύνολο τιμών.

Maximum. Η μέγιστη τιμή σε ένα σύνολο τιμών

minimum. Η ελάχιστη τιμή σε ένα σύνολο τιμών

median. Η τιμή που βρίσκεται στη μέση ενός συνόλου τιμών αν αυτές μπουν σε διάταξη

mode. Είναι η τιμή χ που μια συνάρτηση πιθανότητας παίρνει τη μέγιστη τιμή της.

Sum. Το άθροισμα των τιμών ενός συνόλου τιμών.

Max n Τα n μεγαλύτερα στοιχεία σε ένα σύνολο τιμών

Min n. Τα n μικρότερα στοιχεία σε ένα σύνολο τιμών.

Χαρακτηριστικά συναρτήσεων συνάθροισης

Οι συναρτήσεις aggregate είναι ευαίσθητες σε διπλοεγγραφές, όταν δηλαδή μια τιμή υπάρχει δύο φορές σε ένα σύνολο δεδομένων. Τις περισσότερες φορές η ύπαρξη μιας τιμής πάνω από μια φορά είναι περιττή, αν όχι ανεπιθύμητη καθώς αλλοιώνουν το αποτέλεσμα. Για παράδειγμα αν μια τιμή χ υπάρχει δύο φορές, η average συγκλίνει προς την χ, ενώ αντίθετα η max δεν επηρεάζεται. Επειδή οι συναρτήσεις είναι περιληπτικές προσφέρουν ανοχή σε περίπτωση απώλειας πακέτων. Ένα άλλο

χαρακτηριστικό είναι η μονοτονία, καθώς μπορούν να γίνουν έλεγχοι όσο νωρίτερα γίνεται(πχ min σε δεδομένα με αύξουσα σειρά). Επίσης κάθε συνάρτηση απαιτεί διαφορετικό ποσό πληροφορίας. Η average για παράδειγμα χρειάζεται μόνο τις τιμές sum και count ενώ η median απαιτεί όλο το σύνολο τιμών.

Simple Aggregate Query

Ένα απλό Aggregate Query είναι μια επερώτηση συνάθροισης χωρίς Group By ή Having. Οι κατηγορία αυτη αποτελεί μια δημοφιλή κατηγορία επερωτημάτων. Περιγράφονται οι τρείς βασικές στρατηγικές αυτής της κατηγορίας:

Direct Delivery. Τα δεδομένα κάθε κόμβου στέλνονται σε έναν κεντρικό κόμβο σύμφωνα με ένα multi-hop πρωτόκολλο δρομολόγησης. Υπολογισμοί γίνονται μόνο σε αυτόν τον κόμβο.

Packet Merging. Το κόστος της αποστολής πολλών μικρών πακέτων είναι πολύ μεγαλύτερο από το να σταλεί ένα μεγάλο πακέτο. Έτσι είναι πιο αποδοτικό να συγκεντρώνονται πακέτα και να στέλνονται μαζικά.

Partial Aggregation. Για distributive και algebraic aggregation όπως min και max, οι υπολογισμοί μπορούν να γίνονται στους ενδιάμεσους κόμβους. Στη συνέχεια διαβιβάζεται μόνο το αποτέλεσμα αυτών στους σταθμούς-βάσης.

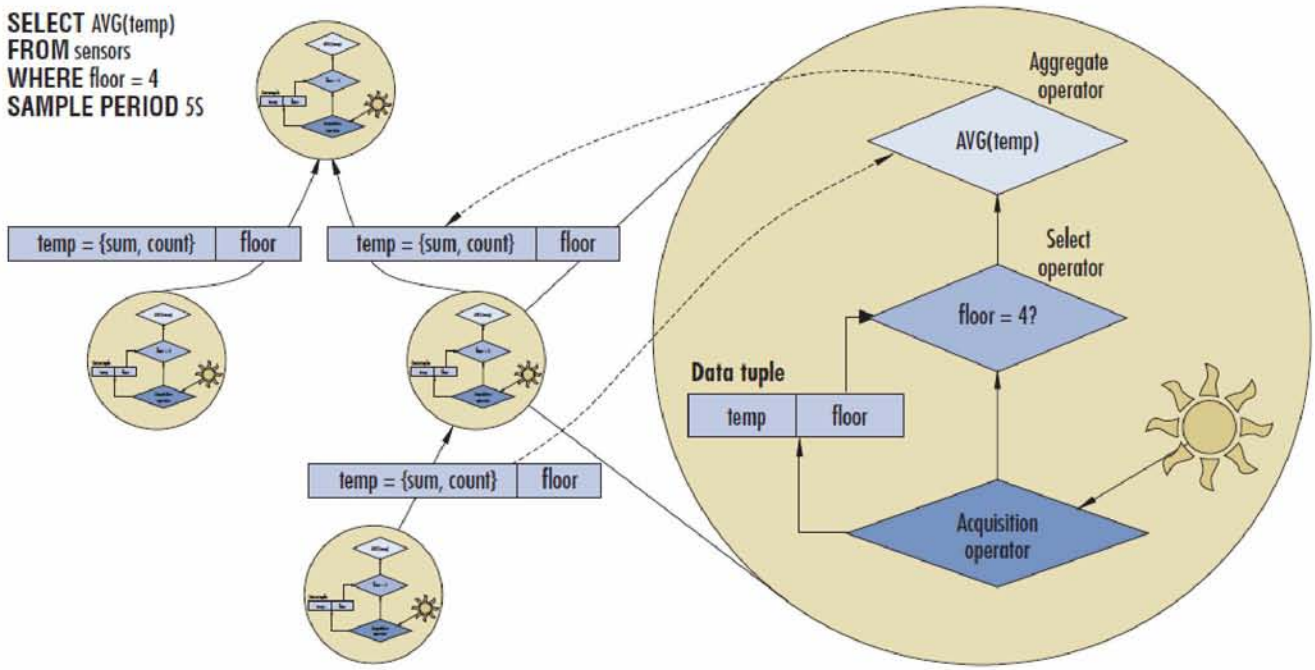
Query processing

Είναι κατανοητό ότι ο χρήστης δεν μπορεί να γνωρίζει για τα γεγονότα στο δίκτυο. Θα πρέπει να ενημερώσει τους κόμβους για την πληροφορία που τον ενδιαφέρει. Επίσης οι αισθητήρες δεν μπορούν να γνωρίζουν εκ τον προτέρων πια πληροφορία μπορεί να ενδιαφέρει τον χρήστη. Έτσι υπάρχουν τρεις προσεγγίσεις για αυτό το ζήτημα.

Push-based query processing. Σύμφωνα με την προσέγγιση αυτή ο κάθε αισθητήρας "διαφημίζει" τα δεδομένα του, τα οποία μπορεί να μεταδίδονται τοπικά ή απευθείας στον σταθμό βάσης για περαιτέρω επεξεργασία.

Pull-based query processing. Σε αυτή τη προσέγγιση ο χρήστης ενημερώνει μια ομάδα κόμβων για την πληροφορία που αναζητά. Αυτό μπορεί να είναι η τιμή μιας εγγραφής ή ένα συμβάν που θέλει να ερευνήσει.

Push–pull query processing. Αυτή η προσέγγιση αποτελεί συνδυασμό των δυο παραπάνω έτσι ώστε και ο χρήστης και οι κόμβοι να εμπλέκονται στην επεξεργασία.



4.5 Αλγόριθμοι

Σε αυτήν την ενότητα περιγράφονται συστήματα και οι αλγόριθμοι τους. Μερικά από αυτά είναι σε πειραματικό στάδιο ενώ άλλα έχουν μέχρι και εμπορικές εφαρμογές.

4.5.1

Tiny Aggregation (TAG)

Το TAG [22] συνδυάζει κατανεμημένη επεξεργασία και πράξεις συνάθροισης μέσα στο δίκτυο. Χρησιμοποιεί απλή δηλωτική γλώσσα διασύνδεσης για συλλογή δεδομένων και εκτέλεση query. Για τον σκοπό αυτό δημιουργείται ένα δέντρο επικοινωνίας. Τα επερωτήματα ξεκινούν από τον χρήστη και κατασκευάζεται ένα δέντρο δρομολόγησης με ρίζα τον σταθμό βάσης. Η μεταφορά δεδομένων γίνεται μέσω αυτού του δέντρου από τα φύλλα στη ρίζα. Το μοντέλο έχει δύο φάσεις, την φάση της διανομής και την φάση της συλλογής, που περιγράφονται παρακάτω.

Φάση διανομής. Σε αυτή τη φάση τα επερωτήματα συνάθροισης διαδίδονται στο δίκτυο από τον σταθμό βάσης. Το αποτέλεσμα είναι ένα δέντρο με ρίζα τον σταθμό βάσης. Κάθε κόμβος συνδέεται με έναν κόμβο-πατέρα, ο οποίος βρίσκεται πιο κοντά στον σταθμό-βάσης και χρησιμοποιείται σαν ενδιάμεσος για την επικοινωνία.

Φάση συλλογής. Σε αυτή τη φάση οι απαντήσεις από τα παιδιά στέλνονται στους γονείς, σύμφωνα με το δέντρο που δημιουργήθηκε στην προηγούμενη φάση. Η επικοινωνία βασίζεται σε εποχές και κάθε επίπεδο επικοινωνεί μόνο σε συγκεκριμένη εποχή(epoch), όπως έχει σχεδιαστεί.

Όταν παράγεται ένα query σχετίζεται με μια εποχή. Η εποχή αυτή ορίζεται ως ο χρόνος που αναμένεται να επιστραφούν τα αποτελέσματα. Επειδή το δίκτυο αποτελείται από επίπεδα, κάθε επίπεδο θέτει ένα deadline στα παιδιά του μέσα στο οποίο πρέπει να εκτελέσουν την εργασία. Το deadline αυτό εξαρτάται από το epoch και από την θέση στην ιεραρχία.

Τη στιγμή που ένας κόμβος i λάβει ένα query r, επιλέγει τον κόμβο που του το έστειλε ως πατέρα. Το r υπονοεί τον χρόνο μέσα στον οποίο πρέπει να απαντήσει στον πατέρα. Αν ο κόμβος είναι

ενδιάμεσος υπολογίζει τον χρόνο που χρειάζεται για τις εργασίες του και στέλνει στα παιδιά το query με deadline μικρότερο του r. Αν ένας κόμβος λάβει το ίδιο query από διαφορετικούς κόμβους, επιλέγει για πατέρα αυτόν με το καλύτερο κανάλι επικοινωνίας ή ότι άλλο κριτήριο έχει οριστεί από το πρωτόκολλο.

Στην φάση της επιλογής τα δεδομένα έχουν συλλεχθεί από τα φύλλα στο περιθώριο της εποχής. Σε αυτό το σημείο το deadline χρησιμοποιείται για να συγχρονίσει τα παιδιά με τους γονείς. Οι γονείς ενεργοποιούν τους πομποδέκτες ώστε να παραλάβουν τα δεδομένα. Αφού παραλάβουν τα δεδομένα γίνεται συνάθροιση τους και στέλνονται στο ανώτερο επίπεδο.

Σε περίπτωση αποσύνδεσης κόμβου το routing table ενημερώνεται, έτσι ώστε τα παιδιά να διαλέξουν καινούργιους πατέρες με το καλύτερο κανάλι. Αν ένας κόμβος αντιμετωπίζει συχνά προβλήματα επικοινωνίας με τον πατέρα του, παρατηρεί τις μεταδόσεις των γειτόνων του για να επιλέξει πατέρα με καλύτερο κανάλι και το ίδιο ή καλύτερο βάθος στο δέντρο.

Ένα από τα πλεονεκτήματα του TAG είναι η μειωμένη κίνηση στο δίκτυο λόγω του συγχρονισμού. Με το κατάλληλο μέγεθος πακέτου και τις σωστές συναρτήσεις aggregate κάθε κόμβος μπορεί να μεταδίδει μόνο ένα πακέτο ανά εποχή. Ακόμα λόγω της συγχρονισμένης επικοινωνίας μειώνεται η κατανάλωση ενέργειας, αφού οι κόμβοι μπορούν να κλείσουν τους πομπούς τους. Επίσης επειδή δεν χρειάζεται όλα τα δεδομένα αλλά μόνο αυτά που προκύπτουν από συναρτήσεις aggregate έχει μεγάλη ανοχή σε σφάλματα.

Από την άλλη, επειδή πολλοί κόμβοι μπορεί να στείλουν μαζί δεδομένα υπάρχει πιθανότητα σύγκρουσης. Ο συγχρονισμός προκαλεί πρόσθετες καθυστερήσεις στο σύστημα οι οποίες είναι ανάλογες με το βάθος του δέντρου. Τέλος αν και ο TAG χρησιμοποιεί κατανεμημένη συνάθροιση για να μειώσει την κίνηση στο δίκτυο, τα queries στέλνονται σε ολόκληρο το δίκτυο ακόμα και στους κόμβους που δεν θα απαντήσουν. Αυτό αυξάνει την κίνηση του δικτύου.

4.5.2 TinyDB

Το TinyDB[23] βασίζεται σε μια acquisitional query processing προσέγγιση, όπου το κόστος για την παραγωγή δεδομένων θεωρείται μέρος της επεξεργασίας επερωτημάτων. Ελέγχοντας την τον ρυθμό συλλογής δεδομένων ανάλογα με τις απαιτήσεις του χρήστη, μειώνεται σημαντικά η ενέργεια.

Όπως και τα περισσότερα συστήματα αισθητήρων το TinyDB βλέπει τα δεδομένα ως έναν πίνακα με μια μόνο στήλη για κάθε τύπο αισθητήρα. Εκτός από περιοδική παρακολούθηση, προσφέρει και event-based με έναν μηχανισμό που ξεκινά την συλλογή δεδομένων. Το δέντρο δρομολόγησης και εδώ δημιουργείται κατά τη διάδοση του query και υπάρχει η έννοια της εποχής(epoch). Και εδώ υπάρχει in-network επεξεργασία και συνάθροιση όσο η πληροφορία πλησιάζει τη ρίζα. Ο TinyDB χρησιμοποιεί μια προσαρμοσμένη SQL, ο χρήστης εισάγει το query και ο κόμβος-ρίζα είναι υπεύθυνος για τη διανομή στο δίκτυο. Γίνεται επίσης βελτιστοποίηση του query όσον αφορά τη σωστή σειρά της συλλογής δεδομένων, τα SELECT και JOIN. Το TinyDB εξειδικεύει τα query σε periodical, event-based και lifetime-based, για να εξοικονομήσει ενέργεια.

Η διάδοση του query γίνεται με ελεγχόμενο flooding, σαν αποτέλεσμα του οποίου δημιουργείται το δέντρο δρομολόγησης. Επιπλέον εξασφαλίζει ότι το query δεν θα μεταδοθεί κάτω από κάποιον πατέρα αν αυτός και όλοι οι γείτονες του δεν πρόκειται να απαντήσουν στην επερώτηση. Το δέντρο που παράγεται λέγεται semantic routing tree (SRT) και ως αποτέλεσμα έχει να στέλνει τα queries μόνο στα κομμάτια του δικτύου που χρειάζεται.

Για να κατασκευαστεί ένα αποτελεσματικό SRT, κάθε πατέρας πρέπει να έχει πληροφορίες για παιδιά του έτσι ώστε να μπορεί να απορρίπτει τα άσχετα queries. Αυτό γίνεται σε δύο φάσεις. Στην πρώτη φάση μια αίτηση για SRT διανέμεται στο δίκτυο, που περιγράφει την πληροφορία που αναζητά το query. Σε αυτή τη φάση οι κόμβοι επιλέγουν τους γονείς τους και προωθούν τις δικές τους πληροφορίες για το query αλλά και των παιδιών τους. Στην δεύτερη φάση το πραγματικό query

διαδίδεται στους κόμβους που έχουν σχετικά δεδομένα.

Το μεγαλύτερο πλεονέκτημα του TinyDB είναι ο τρόπος με τον οποίο σχηματίζεται το δέντρο δρομολόγησης. Εξοικονομείται με αυτόν τον τρόπο σημαντική ποσότητα ενέργειας από τους κόμβους που δεν συμμετέχουν στο query. Όμως το SRT βασίζεται σε flooding, οπότε το συγκεκριμένο μοντέλο ταιριάζει κυρίως σε περιβάλλοντα που αλλάζουν δυναμικά. Σαν αποτέλεσμα τα SRT πρέπει να ανανεώνονται συνέχεια.

4.5.3 Fjords

Το μοντέλο Framework in Java for Operators on Remote Data Streams (Fjords) [24] αναπτύχθηκε για να χειρίζεται stream δεδομένων από αισθητήρες, στα οποία εκτελεί in-network πράξεις. Ορισμένα κομμάτια από διαφορετικά stream μπορούν να ενωθούν για πιο αποδοτική μετάδοση δεδομένων, σε ένα δίκτυο που υποστηρίζει pull-based και push-based επερωτήματα. Εκτός από την ύπαρξη χειριστών για την επεξεργασία επερωτημάτων, το μοντέλο περιλαμβάνει και sensor proxies για την επικοινωνία των χειριστών και των αισθητήρων που εκτελούν συλλογή δεδομένων. Ως stream ορίζεται ένα συνεχές σύνολο από δεδομένα που προωθείται(pushed) στο δίκτυο από τους αισθητήρες. Τα δεδομένα παράγονται συνεχόμενα με συγκεκριμένο ρυθμό και μεταφέρονται στο δίκτυο. Τα streams έχουν άπειρο μήκος οπότε πράξεις πάνω σε ολόκληρα streams δεν μπορούν να γίνουν. Μπορεί όμως να γίνει επεξεργασία ενός συνόλου δεδομένων on-the-fly ή για ένα κομμάτι που βρίσκεται αποθηκευμένο τοπικά. Τα streams χειρίζονται από μια ουρά που ορίζει έναν σωλήνα ανάμεσα σε δύο σημεία. Οι σωλήνες αυτοί μπορεί να αφορούν δύο σημεία μέσα σε έναν απλό κόμβο ή σε διαφορετικούς κόμβους.

Οι χειριστές πραγματοποιούν επεξεργασία των επερωτημάτων μέσω αυτών των ουρών. Ένας χειριστής χαρακτηρίζεται από ένα σύνολο ουρών εισόδου Qi και ένα σύνολο ουρών εξόδου Qo. Ανάλογα με την εργασία που εκτελεί μπορεί να επιλέξει έναν συνδυασμό ουρών εισόδου για να συλλέξει δεδομένα και να εξάγει από κάποιες ουρές εξόδου. Επεξεργασία γίνεται μόνο από χειριστές, γιαυτό και οι ουρές εξόδου οδηγούν σε άλλους χειριστές, εξασφαλίζοντας έτσι αξιοπιστία.

Εφόσον οι ουρές συμπεριφέρονται σαν σωλήνες ανάμεσα στους χειριστές του δικτύου, υποστηρίζονται push-based και pull-based επερωτήματα. Για push επερώτημα, ο χειριστής παρέχει δεδομένα συνεχώς σε μια ουρά, και κάποιος άλλος χειριστής μπορεί να τα παραλάβει. Αντίστοιχα για pull επερωτήματα, ο χειριστής ζητά δεδομένα από έναν χειριστή, ο οποίος ανταποκρίνεται τοποθετώντας δεδομένα στην ουρά. Επειδή το μοντέλο χρησιμοποιεί ροές δεδομένων, δεν είναι δυνατόν να αποκτηθούν απευθείας συγκεκριμένα δεδομένα. Για το λόγο αυτό ορίζονται ορισμένοι χειριστές για πράξεις συνάθροισης, όπως average, count και sort οι οποίοι λειτουργούν αυξητικά, χωρίς να απαιτείται ολόκληρο το stream για τον υπολογισμό.

Η αρχιτεκτονική αυτή μπορεί να χρησιμοποιηθεί από πολλούς χρήστες οι οποίοι μπορεί να στέλνουν παρόμοια query. Αν το κάθε query αντιμετωπιστεί ξεχωριστά θα χρειαστούν σημαντικοί πόροι σε κάθε κόμβο. Για το λόγο αυτό το Fjords χρησιμοποιεί proxies που λειτουργούν ως διασύνδεση ανάμεσα στους αισθητήρες και το κομμάτι του δικτύου που εκτελεί την επεξεργασία. Οι proxies εκτελούν μια σειρά από σημαντικές εργασίες όπως προσαρμογή του ρυθμού δειγματοληψίας, οδηγεί τους κόμβους στη δημιουργία δειγμάτων, ομαδοποιεί τα δείγματα σε tuples, κατευθύνει τα tuples στα κατάλληλα queries.

Οι δυνατότητες των αισθητήρων είναι περιορισμένες. Όμως πολλοί χρήστες μπορεί να ενδιαφέρονται για συγκεκριμένες πληροφορίες από έναν κόμβο. Οι proxies διασφαλίζουν ότι οι κόμβοι δεν θα σπαταλήσουν πόρους για κάθε query. Τα queries συναθροίζονται έτσι ώστε μόνο ένα αντίγραφο μιας παρατήρησης να παράγεται στον αισθητήρα, αλλά το αντίγραφο αυτό να στέλνεται σε όλα τα query που σχετίζονται με αυτό.

Το Fjords είναι ένα αποτελεσματικό μοντέλο για ροές δεδομένων. Συνεχόμενα streams μπορούν να συναθροισθούν σε ένα χρονικό διάστημα. Επίσης συναθροίζοντας τα query που έχουν ομοιότητες και

μειώνεται σημαντικά ο φόρτος του δικτύου αφού οι αισθητήρες δεν χρειάζεται να αντιμετωπίζουν το κάθε query ξεχωριστά. Ακόμα επειδή ολόκληρη η πληροφορία που φεύγει με ροή από τους αισθητήρες μπορεί να φτάσει στον σταθμό βάσης, τα ακατέργαστα δεδομένα μπορούν να αποθηκευτούν για περαιτέρω επεξεργασία(post processing). Το Fjords όμως δεν υποστηρίζει event-based επερωτήματα.

4.5.4 COUGAR

Το COUGAR[25] παρέχει ένα πλαίσιο κατανεμημένης συλλογής πληροφορίας, για ξεχωριστά δεδομένα, μέσα από τα query που ορίζει ο χρήστης. Τα δεδομένα χωρίζονται σε δύο κατηγορίες: τα αποθηκευμένα και τα δεδομένα των αισθητήρων. Τα αποθηκευμένα δεδομένα(stored data), αντιστοιχούν σε πληροφορίες για τους αισθητήρες όπως ο τύπος, η τοποθεσία, και ιδιότητες του φαινομένου που παρατηρούν. Τα αποθηκευμένα δεδομένα βρίσκονται στον σταθμό-βάσης και μοντελοποιούνται σχεσιακά. Τα δεδομένα των αισθητήρων(sensor data) αντιστοιχούν σε πληροφορίες που έχουν συλλεχθεί από τους αισθητήρες σύμφωνα με τα query. Τα δεδομένα αυτά παρουσιάζονται ως time series. Πιο συγκεκριμένα οι αισθητήρες θεωρούνται συγχρονισμένοι, και τα δεδομένα τους θεωρούνται αποτέλεσμα μιας συνάρτησης επεξεργασίας σήματος, σε μια συγκεκριμένη τοποθεσία και σε συγκεκριμένο χρόνο. Έτσι τα δεδομένα παρουσιάζονται σε μορφή 3-tuple που περιλαμβάνει ένα σετ δεδομένων, έναν ταξινομημένο τομέα και ταξινομημένα δεδομένα σύμφωνα με τον κάθε τομέα.

Υπάρχουν δύο τύποι query, ένας για κάθε τύπο δεδομένων που βασίζονται σε σχεσιακούς και ακολουθιακούς τελεστές. Οι σχεσιακοί τελεστές αφορούν τα αποθηκευμένα δεδομένα, ενώ οι ακολουθιακοί τα δεδομένα των αισθητήρων. Ακόμα τρεις τελεστές ορίζονται για να συνδέουν τα δύο είδη:

- Ακολουθίες μπορούν να χρησιμοποιηθούν ως είσοδο σε μια πράξη για να αφαιρέσουν την πληροφορία της τοποθεσίας και να δημιουργήσουν μια σχέση.
- Μια ακολουθία και μια σχέση μπορούν να χρησιμοποιηθούν για να παράγουν μια ακολουθία.
- Μια ακολουθία μπορεί να επεξεργαστεί με συναρτήσεις συνάθροισης για να αφαιρέσει την ιδιότητα της τοποθεσίας.

Οι αισθητήρες παρατηρούν φαινόμενα και τα δεδομένα που συλλέγουν βασίζονται σε συγκεκριμένες συναρτήσεις επεξεργασίας σήματος όπως δειγματοληψίας, σύγκρισης, integration. Η έξοδος των αισθητήρων παρουσιάζεται ως συνάρτηση αφηρημένου τύπου δεδομένων (ADT) έτσι ώστε ένα ADT αντικείμενο στην βάση δεδομένων να αντιστοιχεί σε έναν αισθητήρα. Με αυτό το σχήμα το COUGAR μπορεί να χρησιμοποιεί SQL γλώσσα για να εκτελεί query στους αισθητήρες.

Στο σταθμό βάσης υπάρχει μια εφαρμογή βελτίωσης των queries(query optimizer) το οποίο δημιουργεί και κατανέμει στο εσωτερικό δίκτυο πλάνα(query plan) από τα queries που λαμβάνει από το εξωτερικό δίκτυο. Τα πλάνα αυτά δημιουργούνται σύμφωνα με τις πληροφορίες κατάλογου(catalog information) και τα χαρακτηριστικά του query. Το πλάνο καθορίζει τη ροή δεδομένων ανάμεσα στους κόμβους και τους υπολογισμούς που γίνονται σε κάθε κόμβο ξεχωριστά. Το πλάνο διαδίδεται μόνο στους κόμβους που αφορά το συγκεκριμένο query.

Επίσης το COUGAR παρέχει ένα επίπεδο query που βρίσκεται ανάμεσα στο επίπεδο δικτύου και το επίπεδο εφαρμογής. Έτσι είναι δυνατή η αλληλεπίδραση ανάμεσα στο πρωτόκολλο δρομολόγησης και στο πρωτόκολλο κατανεμημένων query. Ο στόχος του επιπέδου query είναι να κάνει πιο αφηρημένη την λειτουργία μια μεγάλης κατηγορίας εφαρμογών και να τις μετατρέψει σε κοινά δηλωτικά επερωτήματα. Επειδή όμως το επίπεδο query έχει διαφορετικές απαιτήσεις δρομολόγησης, τα υπάρχοντα πρωτόκολλα δρομολόγησης χρειάζονται τροποποιήσεις.

Η τυπική αρχιτεκτονική του COUGAR αποτελείται από κόμβους-αισθητήρες και κόμβους-αρχηγούς(leader). Οι αισθητήρες στέλνουν τα δεδομένα τους στον leader, ο οποίος πραγματοποιεί βέλτιστη συνάθροιση. Ανάλογα με το query ένας συγκεκριμένος κόμβος αναλαμβάνει τον ρόλο του leader. Η επιλογή του leader μπορεί να γίνει είτε με την ανάθεση σε κάποιους κόμβους με μεγάλη

υπολογιστική ισχύ είτε μπορεί να γίνει κατανεμημένα στο δίκτυο από έναν κατανεμημένο αλγόριθμο επιλογής.

Όταν ξεκινάει ένα query ο query optimizer προσπαθεί να ενώσει το query με παρόμοια υπάρχοντα queries. Αν υποθέσουμε ότι το query είναι το μοναδικό που υπάρχει στο δίκτυο τότε ο query optimizer δημιουργεί ένα καινούργιο πλάνο. Παράγονται δύο πλάνα, ένα για τον κόμβο leader και ένα για τους υπόλοιπους κόμβους που συμμετέχουν στο query. Οι απλοί κόμβοι περιλαμβάνουν έναν μηχανισμό ανάγνωσης δεδομένων και αποστολής στον leader. Ακόμα το πλάνο τους περιλαμβάνει aggregation operator για να επεξεργαστεί τα δεδομένα από άλλους αισθητήρες.

Αρχικά το πλάνο διανέμεται σε όλους τους proxy των σχετικών κόμβων. Ο proxy καταχωρεί το query, δημιουργεί ένα τοπικό δέντρο ελέγχου, ενεργοποιεί σχετικούς αισθητήρες και επιστρέφει εγγραφές που αφορούν το συγκεκριμένο πλάνο. Ο leader θα αποθηκεύσει τις εγγραφές μόνο εάν η μέση τιμή τους ικανοποιεί το κατώφλι του query.

Επίσης αν και η βέλτιστη συνάθροιση γίνεται στον leader, οι κόμβοι μπορούν να εκτελέσουν μερική συνάθροιση σύμφωνα με τις πληροφορίες που παίρνουν από τους γείτονες τους. Κάθε κόμβος εκτελεί μερική συνάθροιση που βασίζεται σε δύο πηγές. Πρώτον ο κάθε κόμβος έχει πληροφορίες για τα δεδομένα που παράγει ο ίδιος. Δεύτερον, αν ένας κόμβος βρίσκεται στην διαδρομή ενός άλλου κόμβου με τον leader λαμβάνει τα δεδομένα του. Αυτές οι πληροφορίες μπορούν να χρησιμοποιηθούν για να συναθροιστούν τα δεδομένα μερικώς με κατανεμημένες και αλγεβρικές πράξεις όπως SUM, AVG, MAX. MIN. Στη συνέχεια κάθε κόμβος στέλνει τα επεξεργασμένα δεδομένα του στον leader.

Στο επόμενο στάδιο ο leader συναθροίζει βέλτιστα τα μερικώς επεξεργασμένα δεδομένα και τα στέλνει στον σταθμό βάσης μόνο αν ταιριάζουν με την περιγραφή του query που έχει στείλει ο χρήστης. Είναι δηλαδή το πρώτο στάδιο φιλτραρίσματος για την κατάλληλη πληροφορία. Το πρωτόκολλο COUGAR απαιτεί ορισμένες μετατροπές σε μερικά επίπεδα. Οι ενδιάμεσοι κόμβοι θα πρέπει να έχουν πρόσβαση στα δεδομένα από τα πακέτα που παίρνουν από τους γείτονες τους. Όμως η αναμετάδοση πακέτων γίνεται στο επίπεδο δικτύου από τις πληροφορίες που έχουν τα header. Γιαυτό απαιτείται από τους ενδιάμεσους κόμβους να "υποκλέπτουν"(intercept) αυτά τα πακέτα. Αυτό γίνεται με κάποια φίλτρα τα οποία επιτρέπουν στο επίπεδο δικτύου να γίνεται η λήψη αυτών των πακέτων με διάφορους μηχανισμούς. Η συνάθροιση δεδομένων γίνεται στο επίπεδο query.

Σε ένα δίκτυο αισθητήρων μπορούν να υπάρξουν πολλά διαφορετικά είδη αισθητήρων και αυτό σημαίνει ότι μπορούν να υποστηριχθούν πολλά είδη εφαρμογών. Οι διάφορες γλώσσες που θα χρησιμοποιηθούν για τα queries θα πρέπει να υποστηρίζουν βασικές πράξεις aggregation(max, average). Γι αυτό το λόγο θα πρέπει η γλώσσες να βασίζονται στις ιδιότητες των δεδομένων και απλά να καθορίζουν ένα γενικό πλαίσιο για τις πράξεις που θα γίνουν πάνω σε αυτά.

Τα queries μπορεί να είναι σύνθετα και να απαιτούν πολύπλοκους υπολογισμούς και αρκετές μεταφορές δεδομένων. Ο query optimizer θα πρέπει να δημιουργεί το βέλτιστο πλάνο για ένα query με βάση την ενέργεια. Έτσι καθορίζει ποία δεδομένα θα μεταφερθούν αλλά και τι υπολογισμούς θα κάνει ο κάθε κόμβος.

Ο query optimizer χρειάζεται κάποια μετα-δεδομένα έτσι ώστε να παράγει το βέλτιστο πλάνο. Τα δεδομένα αυτά περιέχονται σε έναν κατάλογο κυρίως σε κάποιο server και περιλαμβάνουν δεδομένα όπως η κατάσταση των κόμβων, το φόρτο εργασίας τους κτλ. Ο κατάλογος αυτός μπορεί να ενημερώνεται με κάθε εκτέλεση query ή με κάποιο άλλο ανεξάρτητο αλγόριθμο.

Ακόμα το COUGAR απαιτεί να κατασκευάζονται μονοπάτια από τον αισθητήρα στον leader για ένα query. Έτσι είναι απαραίτητη η τροποποίηση του επιπέδου δικτύου για την δημιουργία μονοπατιών και τη συντήρηση τους. Για τον κάθε leader το δέντρο δρομολόγησης πρέπει να δημιουργηθεί ξεκινώντας από τον τελικό κόμβο συνάθροισης. Τέλος, επειδή τα πακέτα που περιέχουν συναθροισμένα αποτελέσματα είναι πιο σημαντικά από τα απλά δεδομένα του κάθε κόμβου, πρέπει να υπάρχει μια τεχνική συντήρησης του μονοπατιού που να λαμβάνει υπόψη το βάθος του δέντρου.

Το COUGAR προσφέρει αποδοτικούς τελεστές που συνδυάζουν τα αποθηκευμένα δεδομένα με τα δεδομένα των αισθητήρων. Έτσι η απόδοση της επεξεργασίας επερωτημάτων βελτιώνεται καθώς

στατικά δεδομένα που σχετίζονται με τους κόμβους αποθηκεύονται κεντρικά, χωρίς να υπάρχει απαίτηση για συνεχή ενημέρωση. Επίσης η συσταδοποίηση του συστήματος προσφέρει κατανεμημένη εργασία, όπου ένα έργο κατανέμεται στους leader κόμβους του δικτύου. Με αυτόν τον τρόπο συναθροισμένα αποτελέσματα παράγονται πιο αποδοτικά.

Το COUGAR επικεντρώνεται σε μακροπρόθεσμα επερωτήματα σε μια push-based αρχιτεκτονική. Γιαυτό το λόγω είναι κατάλληλο για την εφαρμογές παρακολούθησης. Από την άλλη, pull-based και event-based επερωτήματα δεν υποστηρίζονται. Επίσης λόγω της δομής του πρωτοκόλλου, υπάρχει μεγάλος φόρτος στους leaders, αφού αυτοί εκτελούν τις περισσότερες πράξεις. Για να λυθεί αυτό θα πρέπει να γίνονται συνεχώς αλλαγές των leader, το οποίο αυξάνει την κίνηση στο δίκτυο λόγω των αλγορίθμων επιλογής.

4.5.5 StonesDB

Μια μεγάλη κατηγορία τεχνικών και αλγορίθμων υπάρχει για τις επερωτήσεις σε live data. Τόσο το TinyDB όσο και το COUGAR έχουν μηχανισμούς ώστε η επεξεργασία αυτών των δεδομένων να γίνεται όσο πιο κοντά στους αισθητήρες που εμπλέκονται και έτσι μόνο το αποτέλεσμα να χρειάζεται να μεταδοθεί. Οι τεχνικές AQP (Acquisitional Query Processing) επιλέγουν από ποιούς κόμβους θα ζητηθεί πληροφορία και πότε. Επίσης καθορίζεται με ποιά σειρά θα αποκτηθεί η πληροφορία έτσι ώστε να υπάρχει βέλτιστη ενεργειακή απόδοση.

Από την άλλη μεριά, για τον χειρισμό δεδομένων που βρίσκονται ήδη στην μνήμη υπάρχουν λιγότερες λύσεις. Υπάρχουν δύο μοντέλα για την επεξεργασία τέτοιων query.

Το πρώτο μοντέλο αντιμετωπίζει τα δεδομένα των αισθητήρων σαν μια συνεχή ροή, η οποία συναθροίζεται μέσα στο δίκτυο και στη συνέχεια μεταδίδεται και αρχειοθετείται έξω από το δίκτυο. Όταν τα δεδομένα συλλεχθούν μπορούν να αποθηκευτούν σε κλασικές βάσεις δεδομένων και να επεξεργαστούν χρησιμοποιώντας τυπικές μεθόδους. Τα δίκτυα αυτά είναι γνωστά ως "dump data collection" και χρησιμοποιούνται για παρακολούθηση φαινομένων. Όμως, αν και ένα τέτοιο δίκτυο είναι εύκολο να αναπτυχθεί, η εφαρμογές πρέπει να είναι βραχυπρόθεσμες, όταν υπάρχει μεγάλος ρυθμός παραγωγής δεδομένων(πχ κάμερα) λόγω των μεγάλων απαιτήσεων σε ενέργεια.

Ένα δεύτερο μοντέλο για επεξεργασία παλαιότερων δεδομένων(historical data) αντιμετωπίζει τους αισθητήρες ως σαν μια βάση που με archival επεξεργασία επερωτημάτων, όπου όλα τα query διαχέονται στο δίκτυο μέχρι τους αισθητήρες που υπάρχει αρχειοθετημένη η πληροφορία τοπικά. Αυτή η αρχιτεκτονική θεωρείται ενεργειακά αποδοτική για αναζήτηση και επεξεργασία αρχειοθετημένων δεδομένων, αφού η επεξεργασία επερωτημάτων πραγματοποιείται στην πηγή και μεταδίδονται μόνο τα αποτελέσματα αντί για ολόκληρα τα δεδομένα. Το μοντέλο αυτό όμως δεν θεωρείται πρακτικό για τρεις λόγους.

Πρώτον, οι αισθητήρες έχουν περιορισμένη υπολογιστική ικανότητα, κάτι το οποίο αποκλείει πολύπλοκες πράξεις σε απομακρυσμένους αισθητήρες. Δεύτερον, η μνήμη των αισθητήρων είναι περιορισμένη(συνήθως μερικά MB) άρα ένα μικρό μέρος της πληροφορίας μπορεί να αρχειοθετηθεί. Τρίτον, οι μνήμες flash θεωρούνται λιγότερο αποδοτικές ενεργειακά, σε σύγκριση με τους μικρο-ελεγκτές και την χαμηλής κατανάλωσης ραδιο-επικοινωνία των αισθητήρων, μειώνοντας έτσι τα οφέλη της τοπικής αποθήκευσης.

StonesDB

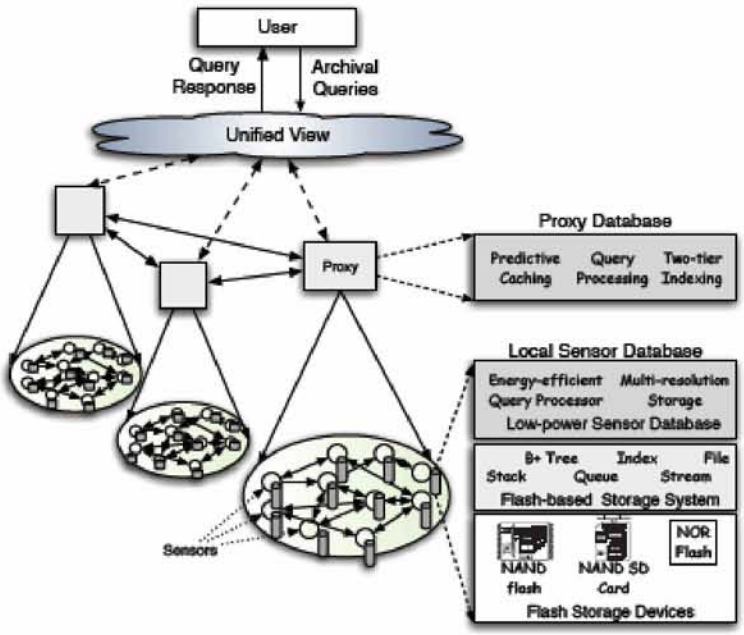
Το StonesDB[26] είναι μια αρχιτεκτονική που δίνει έμφαση στην τοπική αποθήκευση και στην

επεξεργασία στους αισθητήρες. Στόχος είναι η βελτιστοποίηση της απόδοσης ενέργειας. Το StonesDB είναι σχεδιασμένο έτσι ώστε να εκτελεί σημαντική επεξεργασία επερωτημάτων μέσα στο δίκτυο και υποστηρίζει παραδοσιακά queries αλλά και καινούργιους τύπους query που είναι συχνή στην ανάλυση των δικτύων αισθητήρων. Σε αντίθεση με τα υπόλοιπα συστήματα που προσπαθούν να βελτιώσουν την in-network επεξεργασία, με αλγορίθμους, indexing και query forwarding, το StonesDB προσπαθεί να το πετύχει με την κατασκευή μια πραγματικής in-network βάση δεδομένων.

Το StonesDB χρησιμοποιεί μια αρχιτεκτονική δύο επιπέδων(two-tier), στο χαμηλότερο επίπεδο της οποίας βρίσκονται οι αισθητήρες που έχουν χαμηλή ενεργειακή υποστήριξη και στο ανώτερο επίπεδο βρίσκονται οι ενεργειακά ισχυρότεροι proxies.

Τα δεδομένα από τους αισθητήρες αποθηκεύονται τοπικά και μια περίληψη(summary) τους στέλνεται στον proxy. Τα query πρέπει να απαντηθούν με έναν συνδυασμό της πληροφορίας που βρίσκεται στον proxy και της πληροφορίας που βρίσκεται στους αισθητήρες. Οι στόχοι του μοντέλου είναι:

- Η εκμετάλλευση της τοπικής μνήμης flash, θεωρώντας ότι η επικοινωνία είναι πιο ακριβή από την αποθήκευση.
- Βελτιστοποίηση της βάσης δεδομένων ώστε να είναι ενεργειακά αποδοτική
- Εκμετάλλευση των proxies ώστε να χρησιμοποιούνται για όσο περισσότερες λειτουργίες γίνεται για να μην επιβαρύνονται οι φτωχότεροι σε ενέργεια αισθητήρες.
- Η υποστήριξη μεγάλου εύρους query, οι οποίες εκτός από τις κλασικές SQL εντολές περιλαμβάνουν και εντολές data-mining
- Η δυνατότητα ύπαρξης διαφορετικών τύπων αισθητήρων.



Το StonesDB είναι σχεδιασμένο να υποστηρίζει ένα ευρύ φάσμα επερωτημάτων σε historical δεδομένα τα οποία είναι αποθηκευμένα σε ολόκληρο το δίκτυο. Τα επερωτήματα ανήκουν σε δύο οικογένειες.

Στην πρώτη οικογένεια περιλαμβάνονται SQL query τα οποία μπορεί να περιλαμβάνουν πράξεις σύγκρισης πάνω σε συγκεκριμένες τιμές ή χρόνους και μια ποικιλία από χωρικές και χρονικές συναθροίσεις. Πιο συγκεκριμένα χρησιμοποιείται in-network επεξεργασία για πράξεις όπως σύγκριση

τιμών, απλές πράξεις συνάθροισης(min, max, avg) και σύνθετες πράξεις συνάθροισης(median, top-k) και πράξεις συνάθροισης που ορίζονται από τον χρήστη.

Στην δεύτερη οικογένεια, ανήκουν τα επερωτήματα data-mining που πραγματοποιούν επεξεργασία σήματος, αναγνώριση προτύπων, χωρική και χρονική ανάλυση. Οι νέοι τύποι επερωτημάτων που εισάγει το StonesDB είναι:

- Time-series Analysis Queries. Αυτό το είδος query ανιχνεύει τάσεις δεδομένων και τυχών ανωμαλίες σε μία συλλογή αρχειοθετημένων δεδομένων. Μπορούν να προσδιορίσουν ανοδικές και φθίνουσες τάσεις καθώς και να αναγνωρίσουν ακραίες τιμές και ασυνέχειες σε δεδομένα. Αυτό το είδος query είναι σημαντικό για αναγνώριση ανωμαλιών(π.χ. ανίχνευση φωτιάς).

- Similarity Search Queries. Η κατηγορία αυτή προσδιορίζει την ύπαρξη δεδομένων που ταιριάζουν με κάποια πρότυπα. Τα queries αυτά είναι σημαντικά για εφαρμογές όπως παρακολούθηση από κάμερες.

- Classification Queries. Είναι μια επέκταση των Similarity Search Queries. Τα συγκεκριμένα query όχι μόνο βρίσκουν όμοια δεδομένα αλλά τα χωρίζουν και σε κατηγορίες(π.χ. εύρεση όλων των τύπων οχημάτων που πέρασαν από ένα δρόμο).

- Signal Processing Queries. Εκτελούνται κλασικές πράξεις σε σήματα όπως FFT, μετασχηματισμός με wavelet, και φίλτρα.

Στόχος είναι να υπάρξει ενεργειακά φθηνή υποστήριξη και για τις δύο οικογένειες ερωτημάτων. Αυτό γίνεται με εφαρμογή κατάλληλων μεθόδων αποθήκευσης, τεχνικές επεξεργασίας των query κτλ. Επίσης σημαντικό ρόλο παίζει αν τα Queries εκτελούνται περιοδικά ή συνεχόμενα.

Η διπλού επιπέδου δομή του StonesDB αποτελείται από μια τοπική βάση δεδομένων που υπάρχει σε κάθε κόμβο-αισθητήρα και ένα κατανεμημένο επίπεδο διαχείρισης δεδομένων στο οποίο βρίσκονται οι proxy και αλληλεπιδρά με επίπεδο βάσης δεδομένων. Η τοπική βάση δεδομένων σε StonesDB έχει τρεις βασικές συνιστώσες. Μια μηχανή επερωτημάτων που παράγει τα σχέδια επερωτημάτων και παρουσιάζει τα αποτελέσματα. Αλγόριθμους κατασκευής περιλήψεων και γήρανσης. Ένα σχέδιο αποθήκευσης το οποίο διευκολύνει το indexing και του αλγόριθμους γήρανσης. Το επίπεδο κατανεμημένης διαχείρισης δεδομένων έχει δύο συνιστώσες. Περιλαμβάνει μια cache που αποθηκεύονται τα summaries και υλοποιεί μια μηχανή επεξεργασίας που καθορίζει πως θα χειρίζεται το κάθε query.

Τα δεδομένα μπορούν να αποθηκευτούν με τρεις τρόπους. Stream, που είναι συνεχής ροή δεδομένων με τη σειρά που δημιουργήθηκαν. Index, τα οποία παράγονται πάνω στα stream έτσι ώστε να μην χρειάζεται να διαβαστεί ολόκληρο για να βρεθεί η κατάλληλη πληροφορία. Summary, τα οποία είναι περίληψη δεδομένων από streams.

Η εξοικονόμηση ενέργειας στην αποθήκευση γίνεται με δύο τρόπους. Η πρώτη είναι με μεθόδους διαμερισμένης πρόσβασης, όπου τα δεδομένα χωρίζονται σε μικρά τμήματα(λογικά ή φυσικά) καθένα από τα οποία έχει το δικό του index και summary. Η δεύτερη είναι το αποκλειστικό indexing, με την έννοια ότι τα index ορίζονται αποκλειστικά μια φορά χωρίς να ανανεώνονται.

Ο αλγόριθμος με τον οποίο δημιουργούνται τα summary εξαρτάται από το είδος των query, από τις απαιτήσεις που έχει ο χρήστης και φυσικά από τον τύπο των δεδομένων.

Η διαδικασία γήρανσης των δεδομένων ξεκινάει όταν η μνήμη γεμίσει, για να δημιουργηθεί χώρος για νέα δεδομένα. Κάθε partition μετά από κάποιο διάστημα υπόκειται σε αυτή τη διαδικασία ανάλογα με το πόσο σημαντικά είναι τα δεδομένα του(η γενικότερα του stream που ανήκει).

Η μηχανή επερωτημάτων σε αντίθεση με άλλα συστήματα που προσπαθούν να βελτιώσουν τον

χρόνο αντίδρασης, θέτει ως στόχο της την βέλτιστη χρήση ενέργειας. Η λογική είναι απλή: πρώτα δημιουργούνται τα πιθανά πλάνα εκτέλεσης και στη συνέχεια επιλέγεται αυτό με την βέλτιστη χρήση ενέργειας. Επειδή υπάρχουν δεδομένα που είναι αποθηκευμένα με τη μορφή summary η aged είναι πολύ δύσκολο να βρεθούν με απλές μεθόδους πρόσβασης. Έτσι το αρχικό επερώτημα θα πρέπει να χωριστεί σε sub-queries για κάθε ένα από τα οποία θα πρέπει να υπάρχει μια μέθοδος πρόσβασης που να καλύπτει τα επιθυμητά δεδομένα. Ένα βασικό ζήτημα δηλαδή για τον query processor είναι να επιλέξει σωστά το πως θα χωριστεί το query έτσι ώστε να καλύπτονται όλα τα δεδομένα της απάντησης. Επίσης ένα βασικό ζήτημα έχει να κάνει με τον τρόπο τα δεδομένα από τα sub-queries θα συγχωνεύονται. Ειδική περίπτωση αποτελεί το αποτέλεσμα να είναι συνδυασμός raw data και summaries. Τέλος ο query processor θα πρέπει να αποφασίζει ποια επερωτήματα θα χωρίζονται. Ας σημειωθεί ότι ορισμένα επερωτήματα όπως medium η percentile δεν μπορούν χωριστούν και να επιστρέψουν εγγυημένα σωστή απάντηση. Τα δύο τελευταία ζητήματα είναι πιθανοτικά.

4.5.6 DQEB - Dynamic query-tree Energy Balancing Protocol.

Τα περισσότερα πρωτόκολλα περιλαμβάνουν τη κατασκευή ενός στατικού δέντρου. Υποθέτουν ότι η ενέργεια θα είναι σταθερή σε όλη τη διάρκεια της λειτουργίας. . Στην πραγματικότητα οι ενδιάμεσοι κόμβοι έχουν μεγαλύτερη κατανάλωση ενέργειας από τους κόμβους φύλλα. αυτό γίνεται επειδή συνήθως αναλαμβάνουν πρόσθετους υπολογισμούς και διαβιβάζουν δεδομένα άλλων κόμβων.

Το DQEB[27] προσπαθεί να λύσει το πρόβλημα αυτό τροποποιώντας δυναμικά τη δομή του δέντρου ανάλογα με την ενέργεια των κόμβων. Σε κάθε κόμβο ανατίθεται ένα βάρος το οποίο αυξάνεται με την μείωση της ενέργειας. Όσο μειώνεται η ενέργεια ο κόμβος μεταφέρεται προς τα κάτω στο δέντρο. Το βέλτιστο είναι τα βάρη των ενδιάμεσων κόμβων να είναι ελάχιστα.

Όταν το βάρος ενός ενδιάμεσου κόμβου ξεπεράσει ένα κατώφλι τότε ζητά από τα παιδιά του εναλλακτικό πατέρα και γίνεται ο ίδιος κόμβος φύλλο. Ο νέος πατέρας μπορεί να είναι κάποιος κόμβος-απόγονος ή μπορεί να είναι κόμβος που δεν είναι απόγονος αλλά ανήκει στο ίδιο επίπεδο με τους απογόνους του. Όταν ο κόμβος γίνει φύλλο αυξάνεται η διάρκεια ζωής του και σε περίπτωση που εξαντληθεί η ενέργεια του έχει λιγότερο κόστος στο δίκτυο απ'οτι αν ήταν ενδιάμεσος κόμβος.

4.6 Πιθανοτικά μοντέλα

Στα περισσότερα δίκτυα αισθητήρων είναι δύσκολο να υπάρχουν συνέχεια ακριβή δεδομένα λόγω της συνεχής αλλαγής των δεδομένων και των περιορισμένων πόρων. Έτσι πολλές φορές ένα query παίρνει απάντηση που βασίζεται σε λανθασμένα ή παλιά δεδομένα. Αν όμως θεωρήσουμε ένα συντελεστή λάθους τότε το αποτέλεσμα μπορεί να αλλάξει. Έτσι το αποτέλεσμα εκτιμάται πιθανοτικά για την εγκυρότατα του.

Πολλές φορές υπάρχει μια μεγάλη κεντρική βάση δεδομένων που αποθηκεύει δεδομένα από κάποιους αισθητήρες τα οποία ενημερώνονται από αυτούς περιοδικά. Έτσι όταν ένας χρήστης εκτελεί ένα query απευθύνεται στη βάση δεδομένων και όχι απευθείας στους αισθητήρες. Αυτό συνεπάγεται ότι ανά πάσα στιγμή η βάση μπορεί να έχει δεδομένα τα οποία δεν έχουν ενημερωθεί από τους αισθητήρες.

Υπάρχουν δεδομένα των οποίων οι τιμή μεταβάλλεται συνεχώς(dynamic attribute) όπως η θερμοκρασία και η μόνη στιγμή που το σύστημα έχει ακριβή γνώση είναι όταν η βάση ενημερώνεται από τους αισθητήρες. Από την άλλη ο ρυθμός με τον οποίον αλλάζουν κάποια δεδομένα σε κανονικές συνθήκες είναι γνωστός(πχ σε ένα χώρο η θερμοκρασία μπορεί να αλλάζει έναν βαθμό σε πέντε λεπτά) όπως επίσης και η ελάχιστη ή μέγιστη τιμή, κάτι το οποίο βελτιώνει την ακρίβεια των δεδομένων.

Όταν υπάρχει μια πιθανοτική εκτίμηση της απάντησης ο χρήστης μπορεί να αποφασίσει εάν τα

δεδομένα είναι χρήσιμα, κάτι το οποίο είναι πολύ καλύτερο από το να πάρει λανθασμένη ή και καθόλου απάντηση.

Η πιθανοτική εκτίμηση είναι ποιό σημαντική σε aggregation queries από ότι σε range queries και αυτό γιατί κάθε αντικείμενο στα range queries εκτιμάται σύμφωνα με την τιμή του και τη πιθανότητα λάθους του, ενώ σε aggregation queries λαμβάνεται υπόψη και η πιθανότητα λάθους των άλλων αντικειμένων(πχ υπολογισμός min ή max). Για κάθε είδος query υπάρχει διαφορετική πιθανότητα λάθους καθώς και διαφορετικός τρόπος εκτίμησης της.

Μια κατηγοριοποίηση των probabilistic queries μπορεί να γίνει με τις παρακάτω δύο κατηγορίες: Πρώτον, τα queries μπορούν να κατηγοριοποιηθούν ανάλογα με τη φύση της απάντησης. Η κατηγορία αυτή(entity-based queries) επιστρέφει ένα σετ αντικειμένων που ικανοποιεί το query. Αντίθετα η άλλη κατηγορία επιστρέφει για απάντηση μια απλή τιμή(value-based query) όπως για παράδειγμα μια τιμή σε κάποιον αισθητήρα ή η μέση τιμή ενός συνόλου. Ακολουθεί ο ορισμός μερικών βασικών πιθανοτικών επερωτημάτων.

1. value-based non-aggregate

Αυτός είναι ο πιο απλός τύπος query. Επιστρέφει μια τιμή και δεν περιλαμβάνει πράξεις συνάθροισης.ένα παράδειγμα αυτής την κατηγορίας είναι το probabilistic single value query(VSingleQ).

Probabilistic Single Value Query(VSingleQ):

Για ένα αντικείμενο *Tk*, η VSingleQ επιστρέφει *l*, *u*, $\{p(x) \mid x \in [l, u]\}$ οπου $l, u \in R$ με $l \leq u$, και $p(x)$ την pdf του *Tk.a* ώστε $p(x) = 0$ όταν $x < l$ or $x > u$.

2. Entity-based Non-Aggregate Class

Επιστρέφει μία ομάδα δεδομένων που ικανοποιούν μια συνθήκη ανεξάρτητα από τα υπόλοιπα δεδομένα.

Probabilistic Range Query (ERQ) :

Για ένα κλειστό διάστημα [*l, u*], όπου $l, u \in \Re$ και $l \leq u$, η ERQ
Επιστρέφει ένα set από tuples (*Ti, pi*), όπου *pi* είναι η μη-μηδενική πιθανότητα το $Ti.a \in [l, u]$.

3. Entity-based Aggregate Class

Επιστρέφει ένα σύνολο δεδομένων που ικανοποιεί μια συνθήκη aggregate. Υπάρχουν δύο τύποι query:

A)Probabilistic Minimum(Maximum) Query(EMinQ (EMaxQ))

Επιστρέφει ένα σύνολο αντικειμένων (T_i,P_i) όπου P_i είναι η μη μηδενική πιθανότητα ότι το T_i είναι η ελάχιστη(ή μέγιστη) τιμή από το σύνολο αντικειμένων T_i.

B)Probabilistic Nearest Neighbor Query(ENNQ)

Δεδομένης μία τιμής $q \in \Re$ επιστρέφει ένα σύνολο αντικειμένων (T_i,P_i) όπου Pi είναι η μη μηδενική πιθανότητα ότι το $|T_i-q|$ είναι η ελάχιστη τιμή από το σύνολο αντικειμένων T

Και για του δύο τύπους ισχύει $\sum_{T_i \in \Re} p_i = 1$

4. Value-based Aggregate Class

Η τελευταία κατηγορία περιλαμβάνει πράξεις aggregate που επιστρέφουν μια τιμή. Σε αυτήν τη κατηγορία ανήκουν:

***A)* Probabilistic Average (Sum) Query(VAvgQ (VSumQ))**
Επιστρέφει l,u$\{p(x) \mid x\}$ όπου l=<u, X είναι τυχαία μεταβλητή για την μέση τιμή των τιμών a των αντικειμένων του T και p(x) είναι η pdf του X που ικανοποιεί p(x)=0 αν x<u ή x>l

B)ProbabilisticMinimum(Maximum) Value Query (VMinQ (VMaxQ))
Επιστρέφει l,u$\{p(x) \mid x\}$ όπου l=<u, X είναι τυχαία μεταβλητή για το ελάχιστο(ή το μέγιστο) για τις τιμές a των αντικειμένων του T και p(x) είναι η pdf του X που ικανοποιεί p(x)=0 αν x<u ή x>l

Οι δύο παραπάνω κατηγορίες επιστρέφουν απαντήσεις σε μορφή πιθανοτικής κατανομής p(x) για το (l,u) έτσι ώστε $\int_{l}^{u} p(x)dx = 1$

Σε αντίθεση με τις ντετερμινιστικές ΒΔ οπου ένα στοιχείο είτε υπάρχει στη ΒΔ είτε οχι, το αν υπάρχει στη ΒΔ είναι πιθανοτικό γεγονός.
Μας ενδιαφέρουν κυρίως τα σχεσιακά πιθανοτικά μοντέλα. Υπάρχουν δύο τύποι πιθανοτικών μοντέλων:

Ντετερμινιστική βάση δεδομένων αλλά οι απαντήσεις σε query πιθανοτικά

Πιθανοτική βάση δεδομένων και οι απαντήσεις σε query πιθανοτικά

probabilistic data κυρίως όταν έχουμε ανακριβή δεδομένα (probabilistic db, top-k answers, KR, probabilistic reasoning, random graphs)

Ακολουθούν μερικά πιθανοτικά μοντέλα

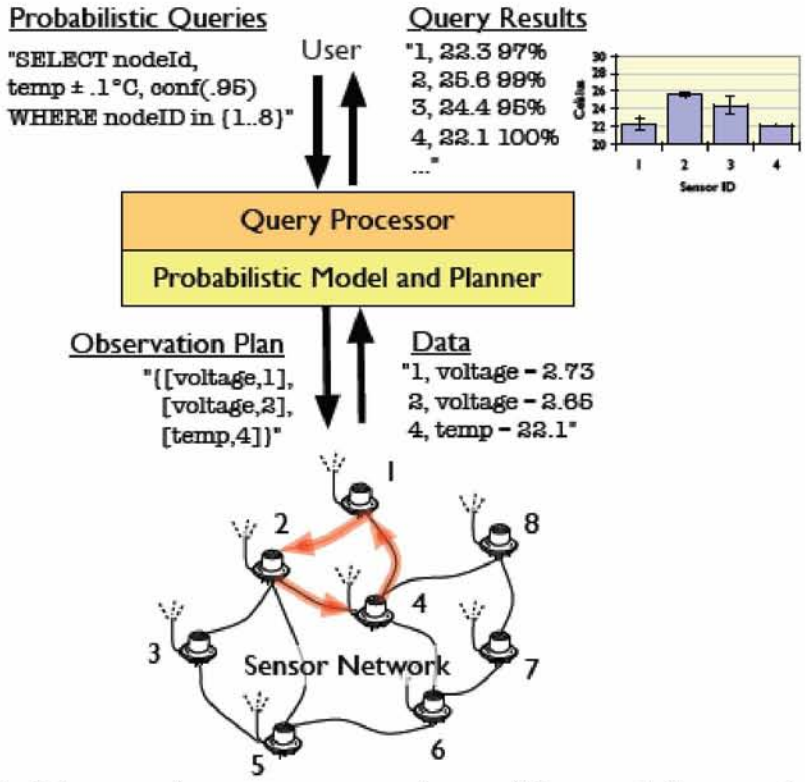
4.6.1 BBQ

Το BBQ[28] θεωρεί μια δηλωτική μηχανή επεξεργασίας επερωτημάτων και μοντελοποιεί το δίκτυο σαν μια pdf $p(X_1, X_2, ..., X_n)$ με X την πιθανότητα μια τιμής σε έναν αισθητήρα. Το μοντέλο είναι χρησιμοποιείται για να προβλέψει τα δεδομένα που παράγονται σε πραγματικό χρόνο. Το μοντέλο χρησιμοποιεί πληροφορίες των κόμβων όταν υπάρχει μεγάλη αβεβαιότητα στις προβλέψεις του. Το BBQ χρησιμοποιεί χρονικά μεταβαλλόμενες μεταβλητές Gauss.

Όταν ο χρήστης ζητά μια πληροφορία με ένα ποσοστό εμπιστοσύνης, το μοντέλο υπολογίζει πιο σχέδιο είναι καλύτερο για την απάντηση και στέλνει αυτό το σχέδιο στο δίκτυο. Τα αποτελέσματα που θα επιστρέψει το δίκτυο δεν χρησιμοποιούνται απλά σαν απάντηση στον χρήστη, αλλά ενημερώνουν και τις τιμές πρόβλεψης του μοντέλου. Αν ο χρήστης ζητήσει ένα επερώτημα με εμπιστοσύνη 100% τότε το μοντέλο είναι υποχρεωμένο να ζητήσει τιμές από όλου τους κόμβους, σε περίπτωση όμως που ζητήσει κάποιο χαμηλότερο ποσοστό εμπιστοσύνης, το μοντέλο μπορεί να απαντήσει μόνο με τις εκτιμήσεις που έχει για κάθε κόμβο.

Το BBQ προσπαθεί να κάνει σωστές εκτιμήσεις χρησιμοποιώντας σχέσεις μεταξύ των αισθητήρων, έτσι ώστε να μην χρειάζεται να ζητά συνέχεια δεδομένα από τους κόμβους για να ενημερώσει τις προβλέψεις του. Πχ δύο δωμάτια μπορεί να έχουν πάντα σταθερή διαφορά θερμοκρασίας 1-2 βαθμούς. Οπότε αν είναι γνωστή η τιμή του ενός μπορεί να προβλεφθεί η τιμή του άλλου με μεγάλο ποσοστό εμπιστοσύνης. Η pdf για d παραμέτρους μπορεί να εκφραστεί από δύο συνιστώσες: έναν πίνακα dxd ο οποίος αποθηκεύει το covariance ανάμεσα στους κόμβους. Και έναν

μονοδιάστατο πίνακα μεγέθους d που αποθηκεύεται η μέση τιμή για κάθε κόμβο.



Όταν ο χρήστης ζητά ένα επερώτημα με εμπιστοσύνη τουλάχιστον 1-δ, μπορούμε να χρησιμοποιήσουμε την pdf για να υπολογίσουμε τις αναμενόμενες τιμές μ_i για κάθε παράμετρο του query. Ακόμα με την pdf μπορούμε να υπολογίσουμε την πιθανότητα το X_i να είναι μέσα στο ε από τον μέσο όρο $P(X_i E[\mu_i-\varepsilon, \mu_i+\varepsilon])$. αν η πιθανότητα αυτή είναι μέσα στις απαιτήσεις του χρήστη, τότε μπορούμε να επιστρέψουμε απάντηση με τη μορφή μέσου όρου μ_i.

Το BBQ χρησιμοποιεί πλάνα παρατήρησης για να ενισχύσει την εμπιστοσύνη των προβλέψεων. Το πλάνο παρατήρησης είναι μια λίστα με κόμβους για κάθε έναν από τους οποίους υπάρχει μια λίστα με γνωρίσματα που πρέπει να παρατηρηθούν. Η λογική είναι να επιλέγονται κόμβοι που θα αυξήσουν την εμπιστοσύνη. Το πλάνο ξεκινάει από τον BS και καταλήγει σε αυτόν.

Το μοντέλο επικεντρώνεται κυρίως στην πιθανοτική απάντηση των παρακάτω επερωτημάτων:

Range queries. Θέλουμε να βρούμε αν η X_i βρίσκεται στο διάστημα $[\alpha , \beta]$. αντί να ανατρέξουμε σε κάθε κόμβο για την τιμή και μετά να ελέγξουμε αν βρίσκεται στο διάστημα χρησιμοποιούμε πιθανοτική προσέγγιση. Αυτό γίνεται σε δύο στάδια. Αρχικά προβάλουμε την pdf $p(X_1, X_2, ..., X_n)$ μόνο για ένα X_i.:

$$p(x_i) = \int p(x_1, x_2, ..., x_n) dx_1, dx_2, ..., dx_n$$

έτσι παίρνουμε την pdf μόνο για το X_i. Στη συνέχεια μπορούμε να υπολογίσουμε το $P(X_i E [\alpha, \beta])$

$$P(X_i E [\alpha, \beta]) = \int_{a_i}^{b_i} p(x_i) dx_i$$

αν η πιθανότητα που επιστρέψει είναι μεγάλη τότε έχουμε απάντηση με μεγάλη εμπιστοσύνη. Αλλιώς η εμπιστοσύνη είναι μικρή.

Value queries. Αν ο χρήστης ενδιαφέρεται για μια την τιμή ενός συγκεκριμένου γνωρίσματος μπορούμε να υπολογίσουμε έναν μέσο όρο της τιμής σε συνδυασμό με μια παρατήρηση ο:

$$\bar{x}_i = \int x_i p(x_i \mid o) dx_i$$

AVERAGE aggregates
Σε αυτή την περίπτωση μπορεί να υπολογιστεί απάντηση με τρόπο παρόμοιο με τις προηγούμενες. Θεωρούμε μια τυχαία μεταβλητή Y ως μέση τιμή για ένα σετ γνωρισμάτων A και υπολογίζουμε την από κοινού pdf των γνωρισμάτων στο A

4.6.2 PRESTO

Το PRESTO [29] είναι μια αρχιτεκτονική δύο επιπέδων που συνδυάζει αισθητήρες και proxies, οι οποίοι συνεργάζονται μεταξύ τους για την απόκτηση της πληροφορίας και την επεξεργασία των επερωτημάτων. Οι proxies παράγουν χρονικά μοντέλα για τάσεις δεδομένων που έχουν παρατηρηθεί και στέλνουν τις παραμέτρους του μοντέλου στους αισθητήρες. Οι αισθητήρες με τη σειρά τους ελέγχουν τα δεδομένα τους και μεταδίδουν μόνο αυτά που αποκλίνουν από το μοντέλο. Το μοντέλο PRESTO εισάγει δύο κεντρικές ιδέες:

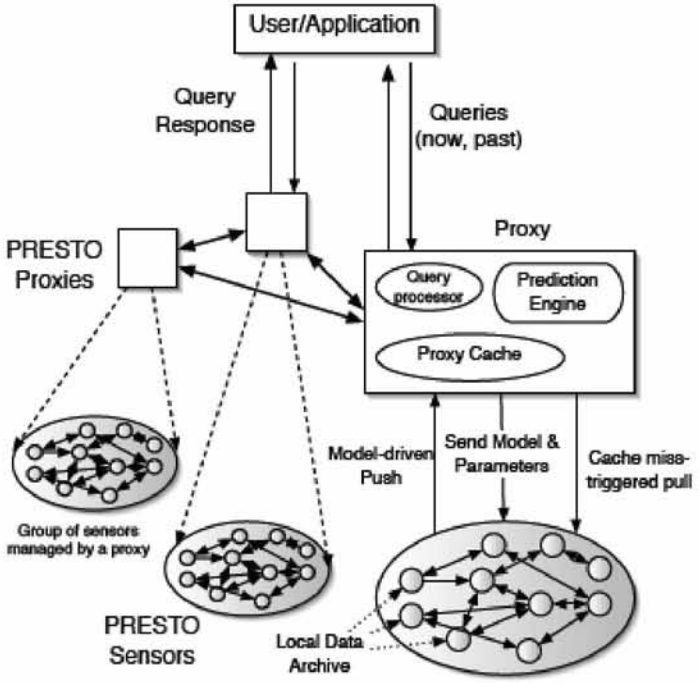
Model-driven Push. Οι proxy του PRESTO δημιουργούν μοντέλα τα οποία βρίσκουν τις συσχετίσεις δεδομένων στα δεδομένα που παρατηρούνται σε κάθε κόμβο. Όπως αναφέρθηκε οι αισθητήρες μεταδίδουν μόνο τις τιμές που δεν ταιριάζουν στα μοντέλα, ανιχνεύοντας έτσι ανωμαλίες. Μια τέτοια προσέγγιση(push-based) μειώνει τον φόρτο επικοινωνίας μεταδίδοντας μόνο τις αποκλίνουσες τιμές, εξασφαλίζοντας ταυτόχρονα ότι αυτές πάντα θα παρατηρούνται.

Historical queries. Το PRESTO υποστηρίζει επερωτήματα σε αρχειοθετημένα δεδομένα χρησιμοποιώντας έναν συνδυασμό πρόβλεψης, παρεμβολής και τοπικής αρχειοθέτησης. Συσχετίζοντας τις τιμές πρόβλεψης με βαθμούς εμπιστοσύνης και cached δεδομένα από προβλέψεις στο παρελθόν, ένας proxy μπορεί να απαντήσει σε τέτοια query χρησιμοποιώντας μόνο τα cached δεδομένα αν είναι μέσα στα περιθώρια λάθους. Επίσης χρησιμοποιεί μεθόδους παρεμβολής για να βελτιώσει παλιές εκτιμήσεις κάθε φορά που λαμβάνει καινούργια δεδομένα.

Προσαρμογή σε αλλαγές των δεδομένων και των query. Ο PRESTO περιοδικά εκπαιδεύει τα μοντέλα του έτσι ώστε να είναι σύμφωνα με τις αλλαγές των δεδομένων και των query. Αλλαγές στην ακρίβεια των query μπορούν να γίνουν μεταβάλλοντας τα κατώφλια σύμφωνα με τα οποία οι αισθητήρες στέλνουν τιμές.

Το μοντέλο χρησιμοποιεί συσχετίσεις από παρατηρήσεις στο παρελθόν για να προβλέψει την πιθανή τιμή σε κάποια στιγμή t στο μέλλον. Το μοντέλο και οι παράμετροι του προωθούνται στους αισθητήρες. Ο αισθητήρας στην συνέχεια συγκρίνει αυτήν τη τιμή με την πραγματική τιμή και αν η διαφορά ξεπερνά κάποιο κατώφλι, το μοντέλο έχει αποτύχει και η κανονική τιμή στέλνεται στον proxy. Σε αντίθετη περίπτωση το μοντέλο θεωρείται ακριβές για τη χρονική στιγμή t. Ο αισθητήρας αποθηκεύει την τιμή αυτή στη μνήμη του και δεν τη μεταδίδει. Έτσι όσο το μοντέλο προβλέπει σωστά δεν χρειάζεται καμία επικοινωνία. Τα μοντέλα παράγουν ένα διάστημα εμπιστοσύνης στο οποίο οι τιμές δεν πρέπει να ξεπερνούν το περιθώριο λάθους. Αν το query απαιτεί μεγαλύτερη ακρίβεια ο proxy ζητάει τις πραγματικές τιμές από τον αισθητήρα. Κάθε query αποθηκεύεται στην cache και έτσι ο

proxy μπορεί να παράγει και απαντήσεις για historical data.



Ένας proxy αποτελείται από τέσσερα στοιχεία:

1. **μηχανή πρόβλεψης και μοντελοποίησης.** Στόχος της μηχανής είναι να δημιουργήσει το μοντέλο πρόβλεψης μελλοντικών τιμών, με βάση τις τιμές από παλαιότερες παρατηρήσεις. Αυτό γίνεται με την χρήση συσχετίσεων και βασίζεται στο μοντέλο ARIMA.

2. **Επεξεργαστή επερωτημάτων.** Εκτός από τις πιθανές τιμές τα μοντέλα περιέχουν και ένα διάστημα εμπιστοσύνης. Ο επεξεργαστής επερωτημάτων συγκρίνει αυτό το διάστημα με το περιθώριο σφάλματος που υπάρχει στο query. Αν το διάστημα εμπιστοσύνης είναι στο περιθώριο σφάλματος τότε θεωρείται ότι θα απαντηθεί με σχετική ακρίβεια. Αλλιώς τα πραγματικά δεδομένα ζητούνται από τους αισθητήρες για να επιστραφούν στον χρήστη. Έτσι πολλά επερωτήματα θα απαντηθούν σωστά, αν και οι πραγματικές τιμές δεν θα έχουν φτάσει ποτέ στον proxy.

3. **τοπική cache.** Θεωρούμε ότι δεν υπάρχει πρόβλημα με τα αποθηκευτικά μέσα στους proxy(microdrives η hard-drives). Αν το αποτέλεσμα από ένα παλαιότερο query μπορεί να χρησιμοποιηθεί για ένα καινούργιο τότε ανακαλείται από τη cache. Σε περίπτωση που απαντηθεί από τα δεδομένα των αισθητήρων οι καινούργιες τιμές αποθηκεύονται στην cache. Τα δεδομένα αυτά χρησιμοποιούνται για να βελτιώσουν την ακρίβεια του μοντέλου με την χρήση παρεμβολής. Χρησιμοποιούνται δύο είδη παρεμβολής: προς τα πίσω και προς τα εμπρός. Η παρεμβολή προς τα εμπρός χρησιμοποιεί την εξίσωση

$$X_t = X_{t-1} + X_{t-S} - X_{t-S-1} + \theta e_{t-1} - \Theta e_{t-S} + \theta\Theta e_{t-S-1}$$

για να προβλέψει τις τιμές. Όπου θ και Θ είναι γνωστοί παράμετροι του μοντέλου, X_{t-1} η προηγούμενη παρατήρηση, X_{t-S} και X_{t-S-1} οι τιμές από παλαιότερο query και e είναι το σφάλμα

πρόβλεψης.

Για να τις επαληθεύσει τα διαστήματα εμπιστοσύνης σύμφωνα με τις νέες τιμές χρησιμοποιεί την παρακάτω εξίσωση:

$$\lambda(l) = \pm u_{\varepsilon/2}(1 + \sum_{j=1}^{l-1}(1-\theta)^2)^{1/2}\sigma$$

Όπου είναι η κανονική κατανομή στο ε/2, σ η διακύμανση για την λάθος πρόβλεψη.

Στην παρεμβολή προς τα πίσω προσαρμόζει τις παλαιότερες προβλέψεις σύμφωνα με τον τύπο

$$X_t^{'} = X_t - \frac{t - T^{'}}{T - T'}e_T$$

όπου X_t η αρχική πρόβλεψη, $X_t^{'}$ η αναθεωρημένη πρόβλεψη, Τ ο χρόνος παρατήρησης της καινούργιας τιμής, και Τ' ο χρόνος της τελευταίας παρατήρησης πριν την ανανέωση.

4. **ανιχνευτή βλαβών.** Επειδή η επικοινωνία είναι περιορισμένη οι βλάβες στο δίκτυο ανιχνεύονται με καθυστέρηση. Ο proxy αντιλαμβάνεται μια βλάβη όταν οι αιτήσεις του προς έναν κόμβο δεν απαντηθούν. Αυτό σημαίνει ότι οι κόμβοι που τα δεδομένα τους συμφωνούν με τα μοντέλα χρειάζονται μεγαλύτερο διάστημα για να εντοπιστούν σε περίπτωση βλάβης από αυτούς που συνεχώς εμφανίζουν διαφορετικά δεδομένα από τα προβλεπόμενα και είναι αναγκασμένοι να επικοινωνούν συχνά με τους proxy.

Κεφάλαιο 5

Simulation

Στα πλαίσια της εργασίας αναπτύχθηκε ένας αλγόριθμος πιθανοτικής επεξεργασίας επερωτημάτων με σκοπό την βελτίωση της απόδοσης σε συστήματα που καλούνται να εκτελέσουν συγκεκριμένες πράξεις πολλές φορές υπάρχει μια γενική γνώση των δεδομένων.

5.1 Το πρόβλημα

Ας θεωρήσουμε ένα σύστημα στο οποίο υπάρχει μια γενική γνώση για την ροή της πληροφορίας και στόχος είναι να χρησιμοποιηθεί η γνώση αυτή ώστε να γίνουν αποδοτικοί αλγόριθμοι για συγκεκριμένες πράξεις. Για παράδειγμα σε ένα σύστημα ελέγχου για την κυκλοφορία σε έναν αυτοκινητόδρομο τα δεδομένα είναι πολύ πιθανό να χαρακτηρίζονται από κάποιες συμπεριφορές. Η μέγιστη κίνηση κατά τη διάρκεια της μέρας θα μπορούσε να είναι το μεσημέρι σε έναν κόμβο-έξοδο για τα προάστια μόλις επειδή τότε φεύγουν οι περισσότεροι εργαζόμενοι για τα σπίτια τους. Επίσης είναι πολύ πιθανό η ελάχιστη κίνηση να είναι σε έναν κόμβο στις τέσσερις το βράδυ. Άρα ένα σύστημα θα μπορούσε να γλιτώσει ένα μεγάλο μέρος του φόρτου του αν καταφέρει να εκμεταλλευτεί τέτοιες πληροφορίες.

5.2 Το μοντέλο

Θεωρούμε ένα κατανεμημένο ιεραρχικό μοντέλο(δέντρο) που αποτελείται από τρία επίπεδα. Στο πάνω επίπεδο υπάρχει ο κόμβος ρίζα ο οποίος είναι και ο κόμβος διασύνδεσης για το σύστημα και ο κόμβος από τον οποίο ξεκινά ένα query και υπολογίζεται η τελική απάντηση. Στο ενδιάμεσο επίπεδο υπάρχουν κόμβοι οι οποίοι δεν εκτελούν λειτουργίες αισθητήρα αλλά εκτελούν μέρος των υπολογισμών και συνδέουν τα δύο άλλα επίπεδα. Κάθε ενδιάμεσος κόμβος είναι πατέρας για ένα σύνολο κόμβων-φύλλων. Στο τελευταίο επίπεδο βρίσκονται οι κόμβοι αισθητήρες που εκτελούν την συλλογή των δεδομένων, την τοπική αποθήκευση τους και μέρος των υπολογισμών. Στα πλαίσια της προσομοίωσης θεωρούμε ότι ο αριθμός των κόμβων είναι σταθερός σε κάθε επίπεδο και δεν μπορεί ούτε να αυξηθεί αλλά ούτε να μειωθεί από βλάβες. Επίσης δεν μπορούν να υπάρξουν λάθη επικοινωνίας και συλλογή λανθασμένων δεδομένων.

5.3 Ο αλγόριθμος

Όταν ζητηθεί από το σύστημα να απαντηθεί ένα επερώτημα οι κόμβοι αποθηκεύουν πληροφορίες της απάντησης για να τις χρησιμοποιήσουν σε μελλοντικά παρόμοια επερωτήματα. Στην προσομοίωση χρησιμοποιήθηκε η συνάρτηση συνάθροισης max(θα μπορούσε αντίστοιχα και η min) αλλά η λογική είναι ίδια για ένα μεγάλο μέρος επερωτημάτων. Την πρώτη φορά που ζητείται ένα επερώτημα η απάντηση γίνεται ντετερμινιστικά Ο κόμβος ρίζα ζητάει από τους κόμβους του ενδιάμεσου επιπέδου να βρουν τον max και αυτοί με τη σειρά τους από τους κόμβους αισθητήρες. Ο κάθε ενδιάμεσος κόμβος παίρνει τις τιμές max που έχουν στείλει οι κόμβοι-φύλλα που του αντιστοιχούν και με τη σειρά του στέλνει τον μεγαλύτερο από αυτούς στην ρίζα. Η ρίζα συγκρίνει τα αποτελέσματα που έστειλαν οι ενδιάμεσοι κόμβοι και επιστρέφει τον αποτέλεσμα στον χρήστη. Ταυτόχρονα τόσο οι ενδιάμεσοι κόμβοι όσο και η ρίζα αποθηκεύουν τον κόμβο που έχει επιστρέψει το σωστό αποτέλεσμα.

Έτσι την επόμενη φορά που θα ζητηθεί ο max η ρίζα θα τον ζητήσει από τον ενδιάμεσο κόμβο που είχε επιστρέψει το αποτέλεσμα την προηγούμενη φορά και με τη σειρά του ο κόμβος αυτός θα το

ζητήσει από τον αισθητήρα που το είχε επιστρέψει. Αυτό γίνεται γιατί θεωρούμε ότι υπάρχει μεγάλη πιθανότητα ο ίδιος κόμβος να επιστρέψει ξανά τον μέγιστο αριθμό στο δίκτυο ή ακόμα ο αριθμός που είχε επιστραφεί στο προηγούμενο query να παραμένει ακόμα max. Αν σκεφτεί κανείς ότι σε ένα σύστημα με δεκάδες ή εκατοντάδες κόμβους αρκεί να ελέγξουμε μόνο έναν κόμβο για την σωστή απάντηση, η επεξεργασία του query μειώνεται σε πολύ μεγάλο βαθμό.

Φυσικά ακόμα και σε ένα σύστημα στο οποίο η συμπεριφορά είναι γνωστή είναι αυτονόητο ότι πρέπει να γίνονται έλεγχοι για τη σωστή λειτουργία. Ακόμα και αν ένας κόμβος επιστρέφει για ένα μεγάλο διάστημα τον max αυτό μπορεί να αλλάξει. Θα πρέπει να υπάρχει ένας μηχανισμός που να προσαρμόζει το σύστημα στα νέα δεδομένα.

Για τον λόγο αυτό ανά τακτά χρονικά διαστήματα εκτελούμε τον ντετερμινιστικό αλγόριθμο για να ελέγξουμε για τυχόν αλλαγές στο σύστημα. Όταν γίνεται αυτό ο αλγόριθμος ελέγχει αν οι τιμή του ντετερμινιστικού αλγορίθμου συμπίπτει με την τιμή που επιστρέφει ο κόμβος για τον οποίο το σύστημα είναι εκπαιδευμένος. Οι αποτυχίες καταγράφονται έτσι ώστε να γνωρίζουμε κάθε στιγμή τι ποσοστό επιτυχίας υπάρχει. Αν συνεχόμενοι έλεγχοι αποτύχουν αυτό σημαίνει ότι πρέπει να προσαρμόσουμε τον αλγόριθμο.

Για τον λόγο αυτό ο χρήστης θέτει ένα ποσοστό σφάλματος για το query. Αν για παράδειγμα απαιτεί το 70% των απαντήσεων να είναι σωστές τότε το σύστημα θα πρέπει να αποτύχει το πολύ στο 30% των ελέγχων. Αν ξεπεράσει το κατώφλι αυτό τρέχει ο ντετερμινιστικός αλγόριθμος και προσαρμόζει τα αποθηκευμένα δεδομένα με αυτά που πραγματικά ισχύουν.

Επίσης αν το κατώφλι αυτό συνεχώς παραβιάζεται τότε θα πρέπει να γίνονται συχνότεροι έλεγχοι και συχνότερη προσαρμογή των αποθηκευμένων δεδομένων. Έτσι κάθε φορά που το ποσοστό επιτυχίας πέφτει κάτω από το κατώφλι οι έλεγχοι γίνονται πιο συχνοί(πχ διπλασιάζονται) και κατά συνέπεια η προσαρμογή των δεδομένων γίνεται συχνότερα αφού η εμφανίζονται συνεχώς σφάλματα. Έτσι το σύστημα αντιλαμβάνεται τις αλλαγές στην πληροφορία και εργάζεται εντονότερα για να προσαρμοστεί στα νέα δεδομένα. Όταν το σύστημα έρθει σε κάποια ισορροπία οι έλεγχοι μειώνονται(πχ υποδιπλασιάζονται) αφού έχουμε λιγότερα σφάλματα και λιγότερη ανάγκη για προσαρμογή. Στην χειρότερη περίπτωση δηλαδή έχουμε συνεχώς διαφορετικό κόμβο που επιστρέφει την μέγιστη τιμή οπότε και ελέγχουμε και προσαρμόζουμε σε κάθε επερώτημα το σύστημα μας. Στην ιδανική περίπτωση πάντα ο ίδιος κόμβος επιστρέφει το σωστό αποτέλεσμα οπότε και οι έλεγχοι γίνονται αραιά.

Στη συνέχεια ακολουθούν κάποια αποτελέσματα από μετρήσεις που έγιναν για να βρεθεί η απόδοση του συστήματος.

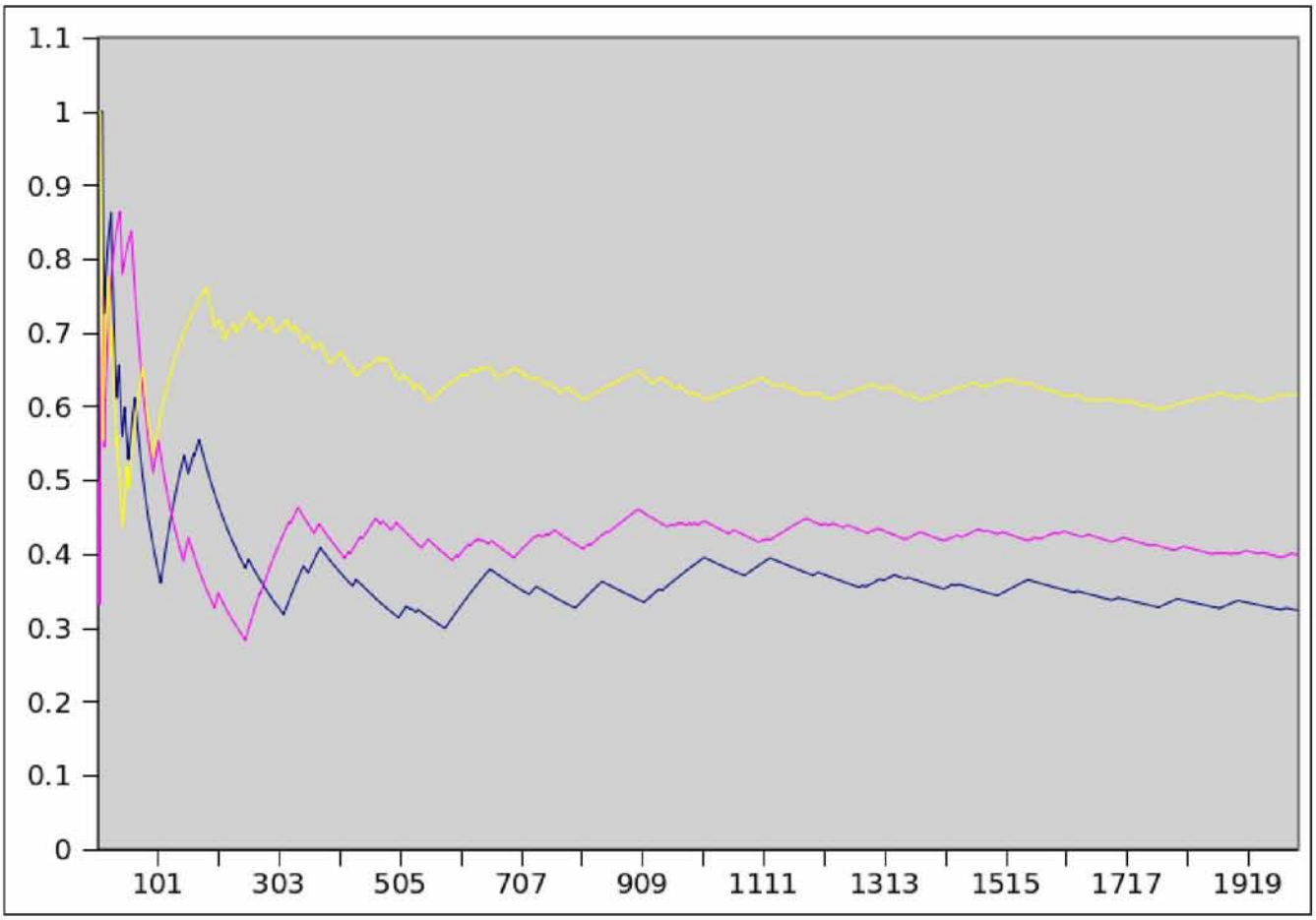
5.4 Μετρήσεις

Οι μετρήσεις έδειξαν ότι το σύστημα έχει παρόμοια απόδοση(όσον αφορά τα ποσοστά επιτυχίας) με διαφορετικό αριθμό κόμβων(εδώ φαίνονται μετρήσεις για 100 και 1000 κόμβους). Κάθε κόμβος έχει τον ίδιο αριθμό πληροφορίας, εδώ 200 μετρήσεις από τις οποίες 100 είναι πρόσφατες και 100 αρχειοθετημένες. Ο αριθμός των ενδιάμεσων κόμβων ορίζεται έτσι ώστε οι αισθητήρες να είναι δεκαπλάσιοι. Επίσης ως χαμηλότερο ρυθμό ελέγχων θέτουμε κάθε ένα query, μέγιστο κάθε 130 query, και αρχικό ρυθμό 50. σημειώνεται ότι το σύστημα τρέχει για πάνω από 200000 κύκλους(συλλογή δεδομένων σε κάθε κύκλο) και γίνονται περίπου 2000 query στο διάστημα αυτό.

5.4.1 Σενάριο 1

η πληροφορία είναι τυχαία, χωρίς να υπάρχει καμία συγκεκριμένη συμπεριφορά(κάθε κόμβος μπορεί να παράγει οποτεδήποτε τον max). Η μετρήσεις έγιναν για ποσοστό εμπιστοσύνης 50%, 70% και 90%. Το ποσοστό 100% δεν υπάρχει σε κανένα σενάριο αφού πρόκειται για τον ντετερμινιστικό

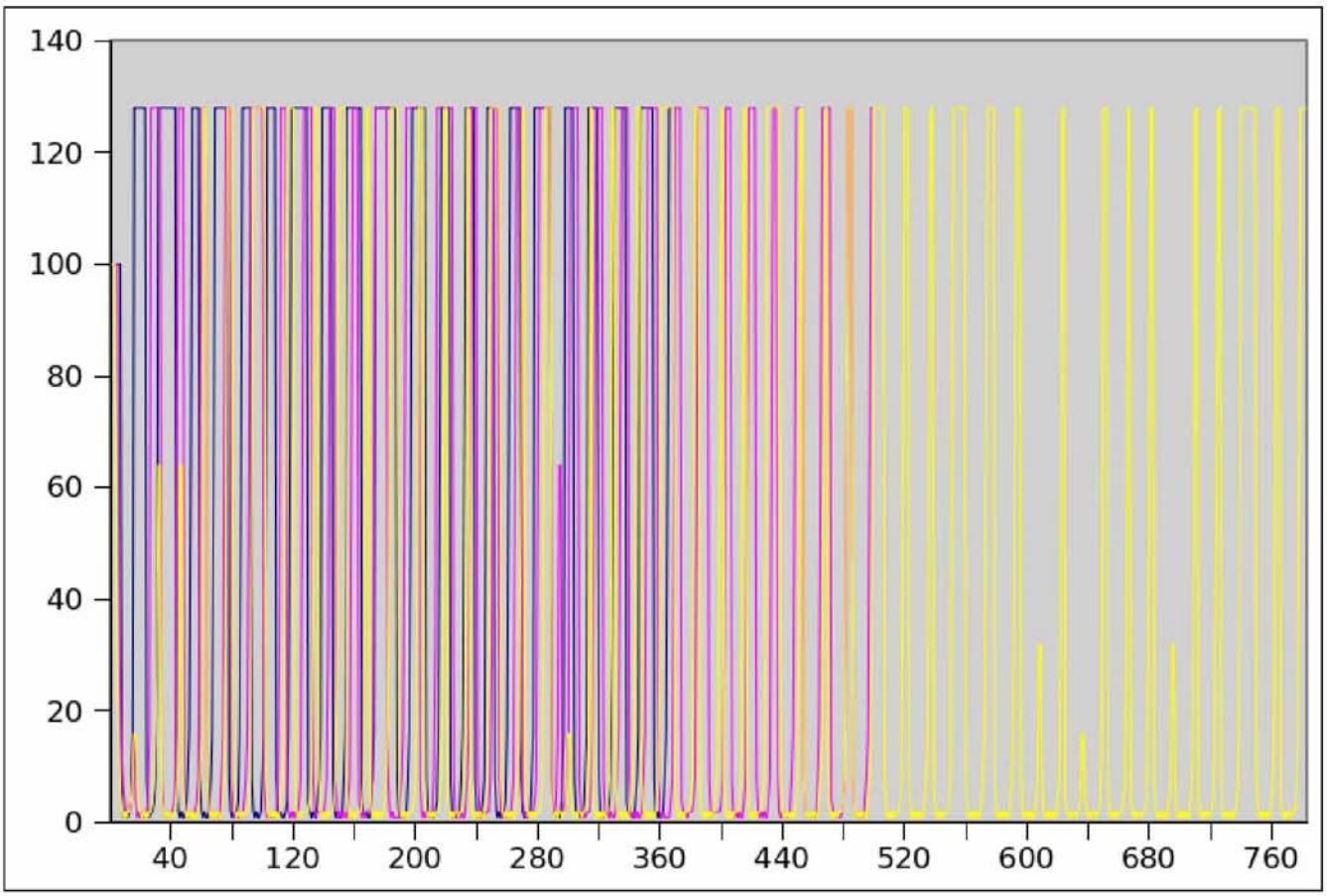
αλγόριθμο.



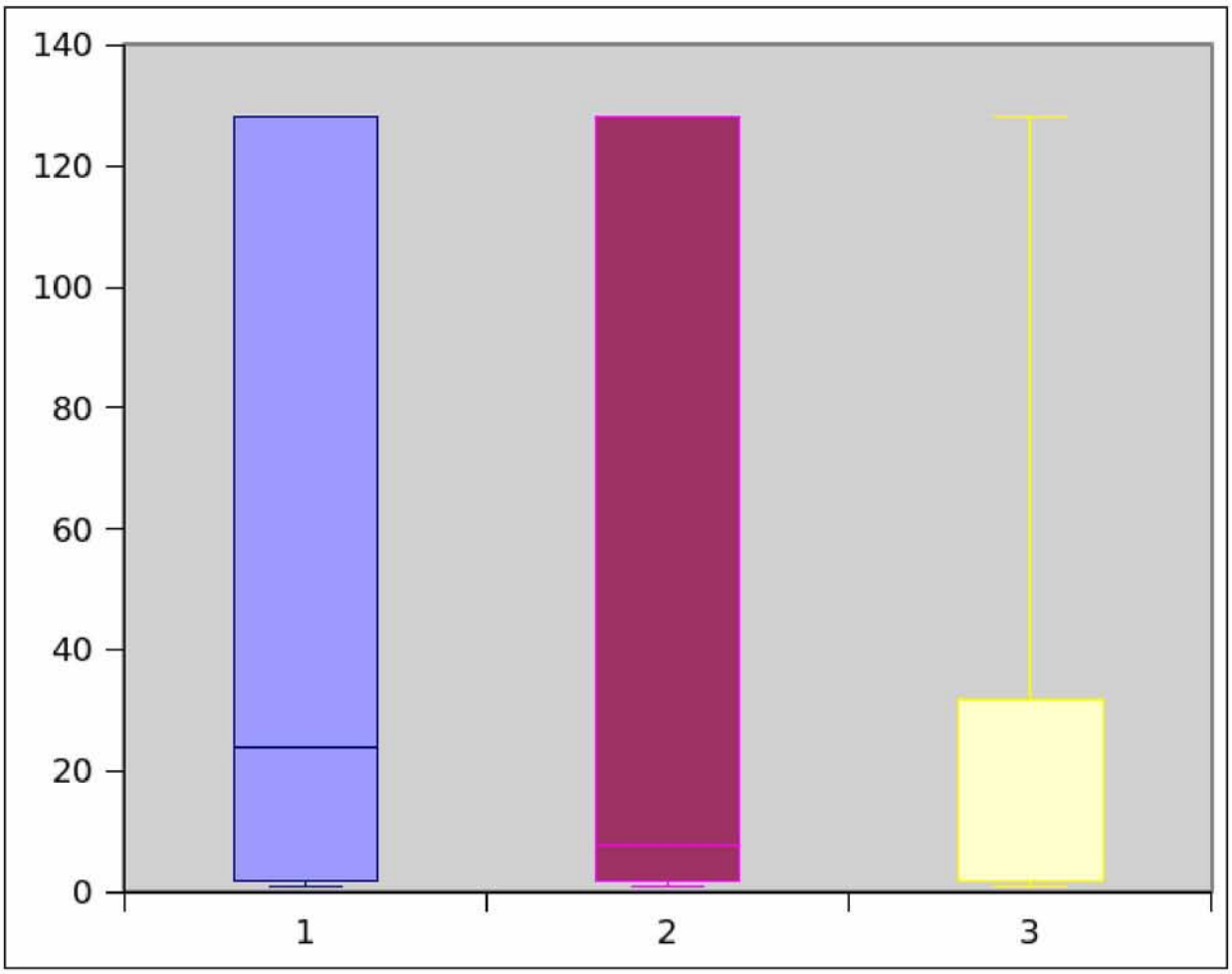
Στο παραπάνω διάγραμμα φαίνεται η πρόοδος για το ποσοστό επιτυχίας σε κάθε περίπτωση.

Τα αποτελέσματα είναι μάλλον απογοητευτικά με ποσοστό επιτυχίας 65% ενώ η εμπιστοσύνη έχει οριστεί 90% και με 50% και 70% να δίνουν λιγότερο από τις μισές φορές σωστό αποτέλεσμα.

Αντίστοιχα οι έλεγχοι και η προσαρμογή τιμών γίνεται πολύ συχνά για καταφέρει το σύστημα να κρατήσει το ποσοστό επιτυχίας όπως ορίζεται από τον χρήστη.



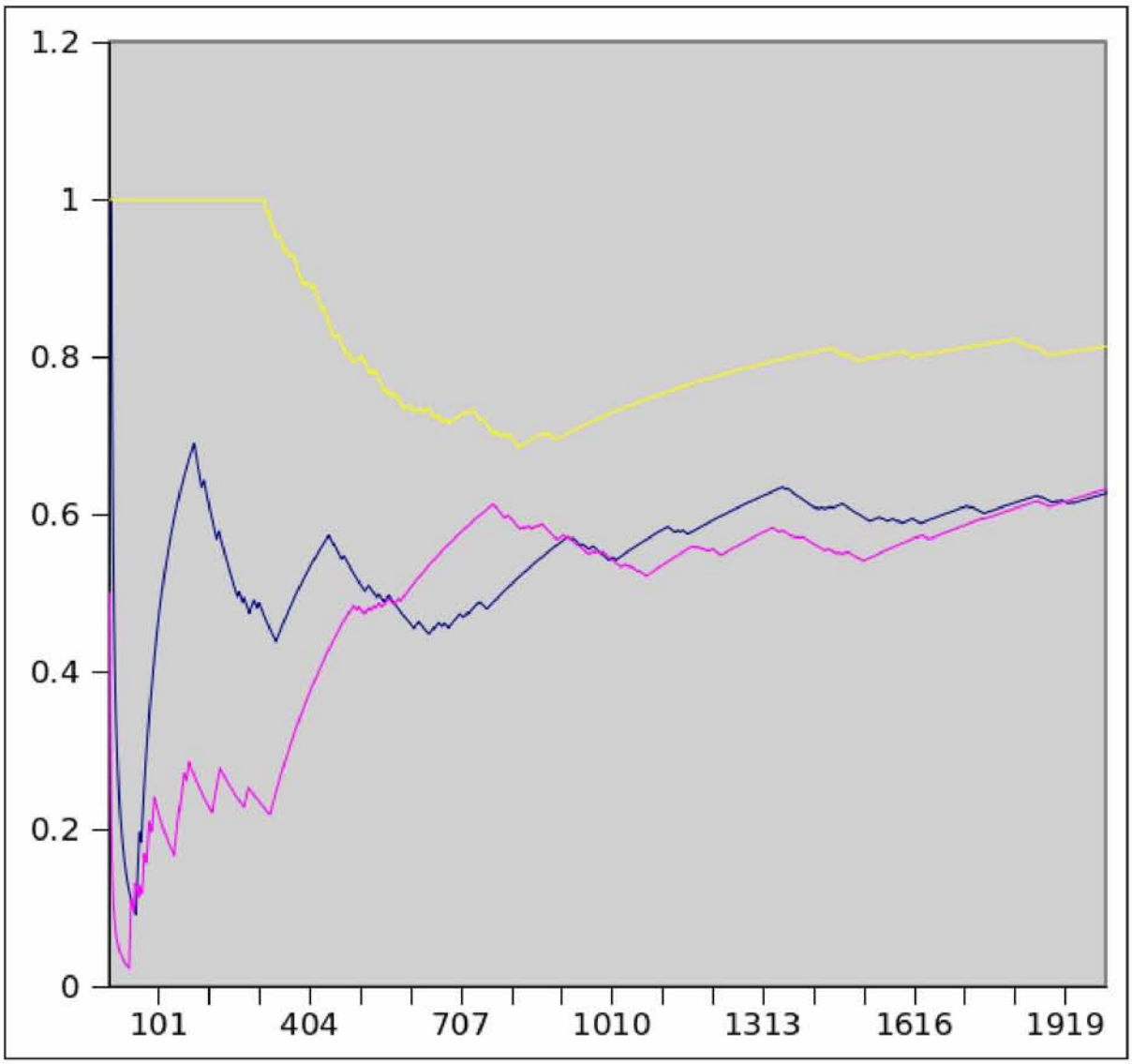
Όπως φαίνεται στο παραπάνω διάγραμμα για να επιτευχθεί ποσοστό επιτυχίας 50% γίνονται 367 έλεγχοι(αυτομάτως αυξάνονται και οι προσαρμογές), για ποσοστό επιτυχίας 70% περίπου 499 και για 90% 783.



Στο παραπάνω διάγραμμα φαίνεται η διακύμανση του ρυθμού ελέγχου. Όπως είναι φυσικό στα 50% και 70% έχουμε μεγαλύτερο εύρος και μέσο όρο(24 και 5 αντίστοιχα) ενώ για ποσοστό 90% ο ρυθμός είναι συνήθως κοντά στο 1.

5.4.2 Σενάριο 2

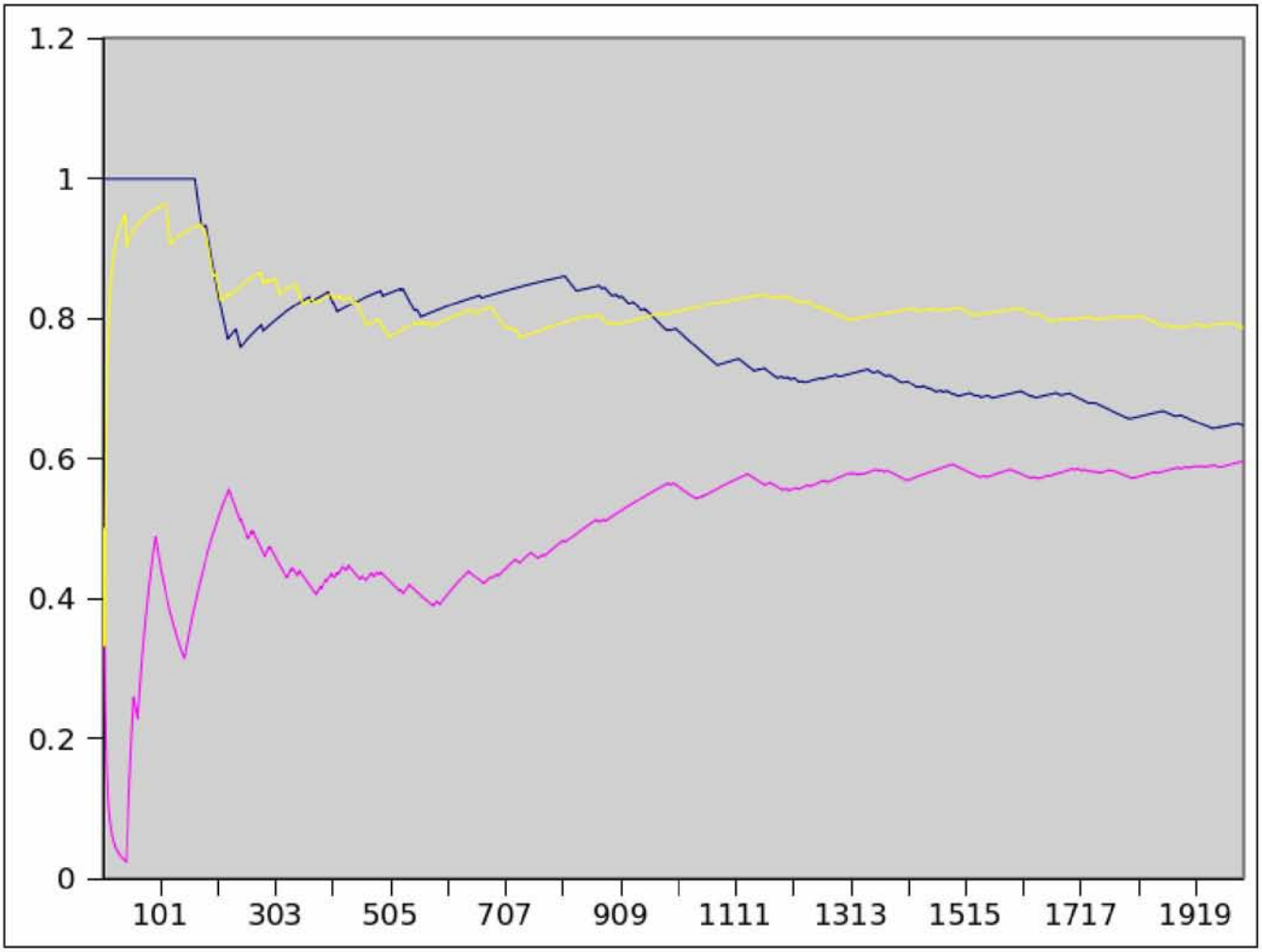
θεωρούμε ότι ένας κόμβος παράγει τον μέγιστο με μεγαλύτερη πιθανότητα από τους υπόλοιπους(10% παραπάνω). Γίνονται οι ίδιες μετρήσεις με πριν.



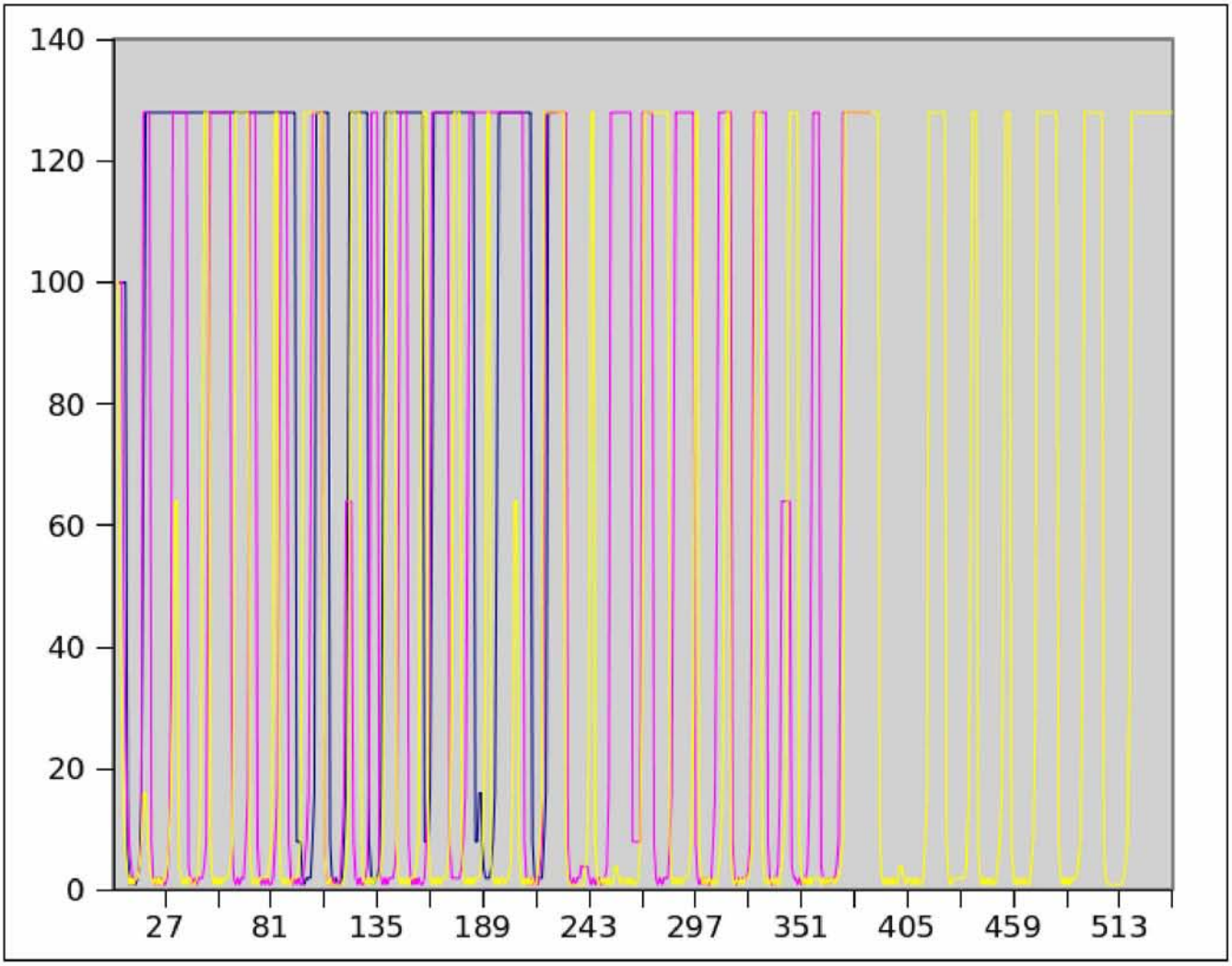
Όπως φαίνεται στο παραπάνω διάγραμμα η επίδοση εδώ είναι καλύτερη απ'οτι πριν αλλά παραμένει κάτω από το ποσοστό επιτυχίας που έχουμε θέσει για μεγάλες τιμές. Συγκεκριμένα έχουμε 81% ενώ απαιτείται 90%, 63% ενώ απαιτείται 70%. για ποσοστό 50% όμως το ποσοστό επιτυχίας είναι 62%. αυτό είναι φυσιολογικό αφού έχουμε βάλει ένα άνω όριο στον ρυθμό ελέγχου το οποίο είναι ίδιο για όλες τις μετρήσεις.

5.4.3 Σενάριο 3

Εδώ θεωρούμε δύο αισθητήρες που έχουν μεγαλύτερη πιθανότητα να παράγουν τον μέγιστο. Οι δύο κόμβοι παράγουν τον μέγιστο με την ίδια πιθανότητα(10% παραπάνω από τους υπόλοιπους).



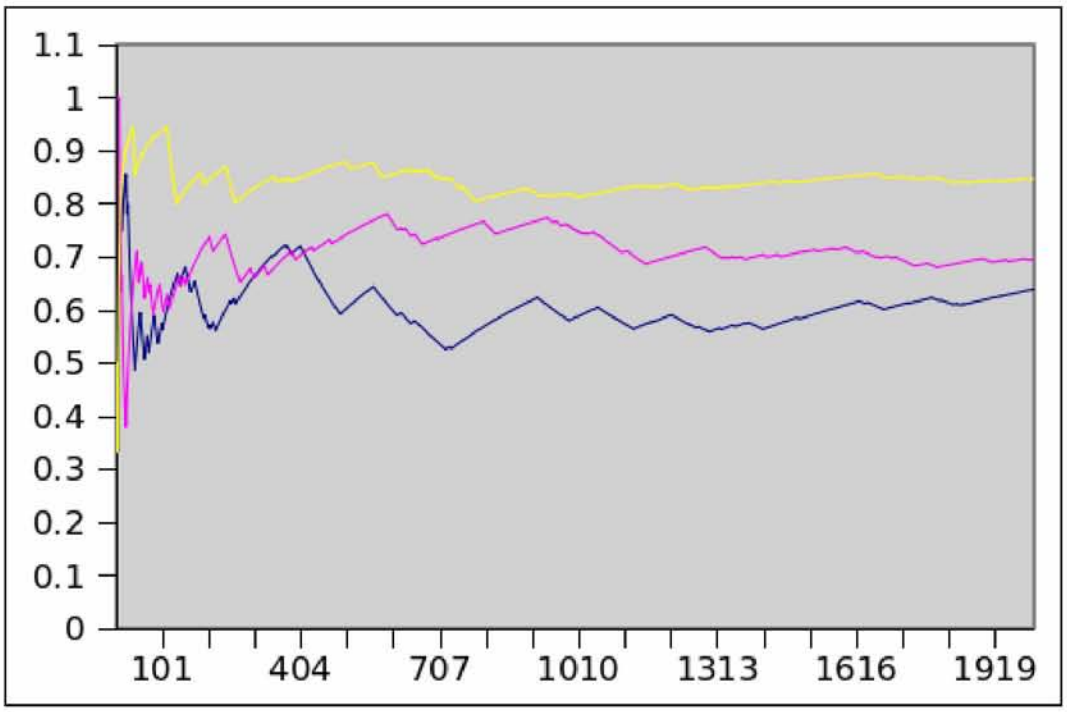
Τα αποτελέσματα και εδώ εμφανίζουν την ίδια συμπεριφορά με πριν. Ωστόσο αυτή τη φορά για 70% ποσοστό επιτυχίας παίρνουμε πιο σωστά αποτελέσματα(65%). Για 90% υπάρχει ακόμα μεγάλη απόκλιση(78%). Για ποσοστό 50% συνεχίζουμε να έχουμε μεγαλύτερο ποσοστό επιτυχίας από το αναμενόμενο.



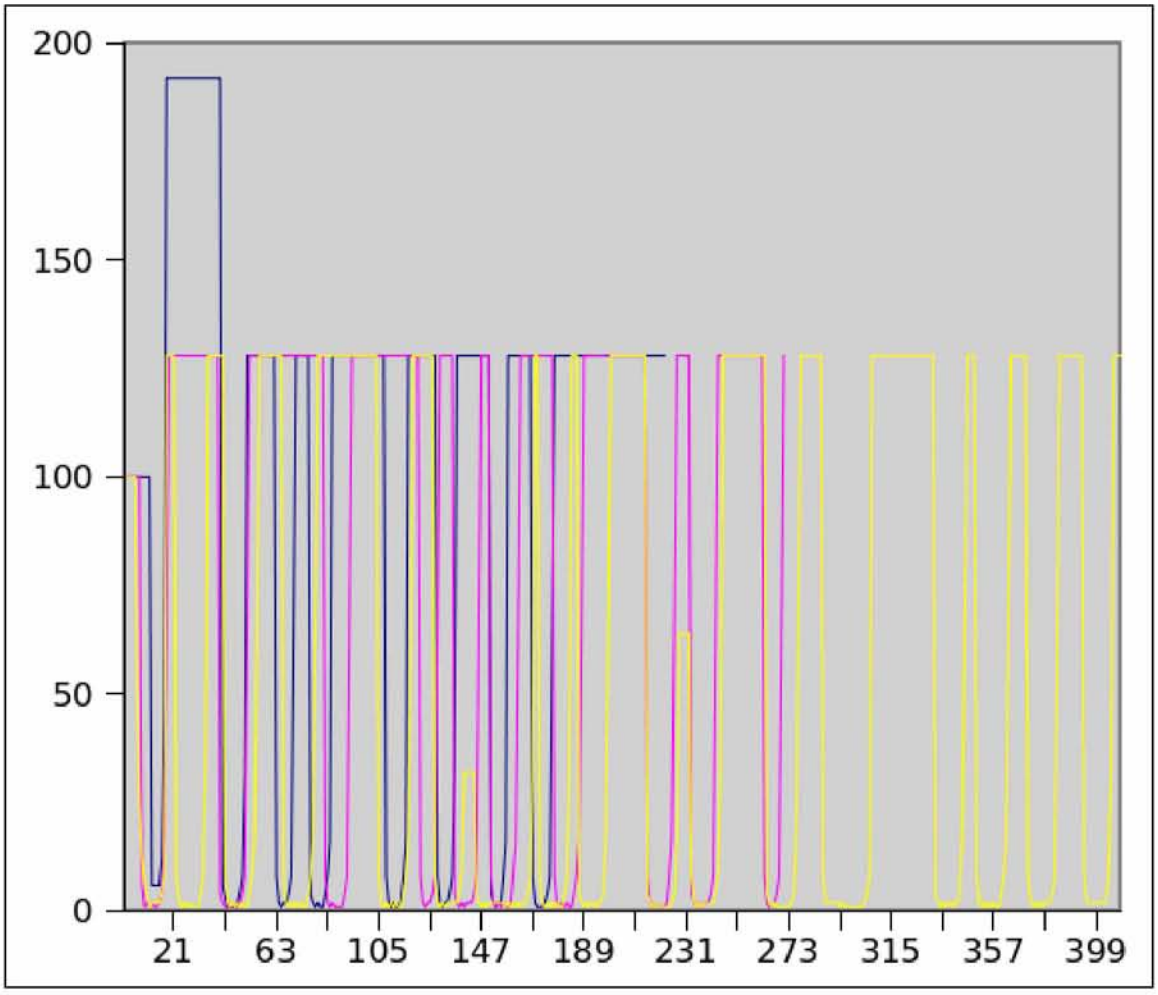
Ο ρυθμός ελέγχου συνεχίζει να είναι μεγαλύτερος όσο αυξάνει το ποσοστό επιτυχίας που απαιτείται.

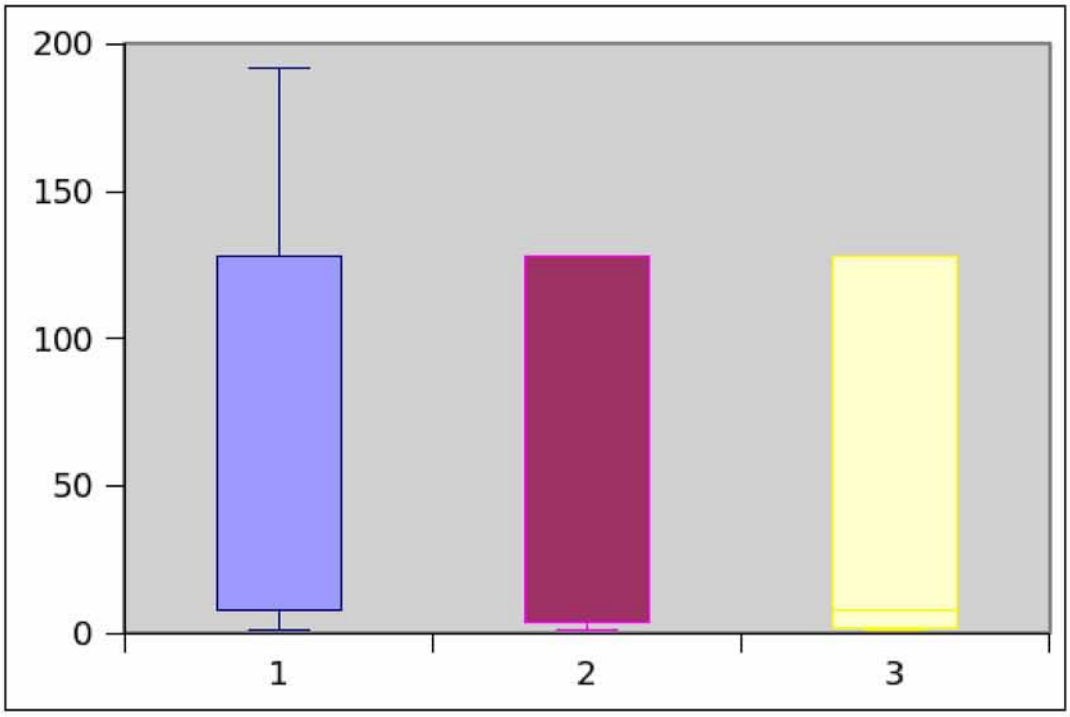
5.4.4 Σενάριο 4

Δύο κόμβοι εμφανίζουν πολύ μεγαλύτερη πιθανότητα να παράγουν το μέγιστο.



Πλέον τα αποτελέσματα πλησιάζουν τις τιμές που έχει ορίσει ο χρήστης. Για 90% έχουμε 85% σωστές απαντήσεις, για 70% έχουμε 69% και για 50% συνεχίζουμε να εμφανίζουμε πολύ μεγαλύτερο από το αναμενόμενο ποσοστό επιτυχίας(64%).

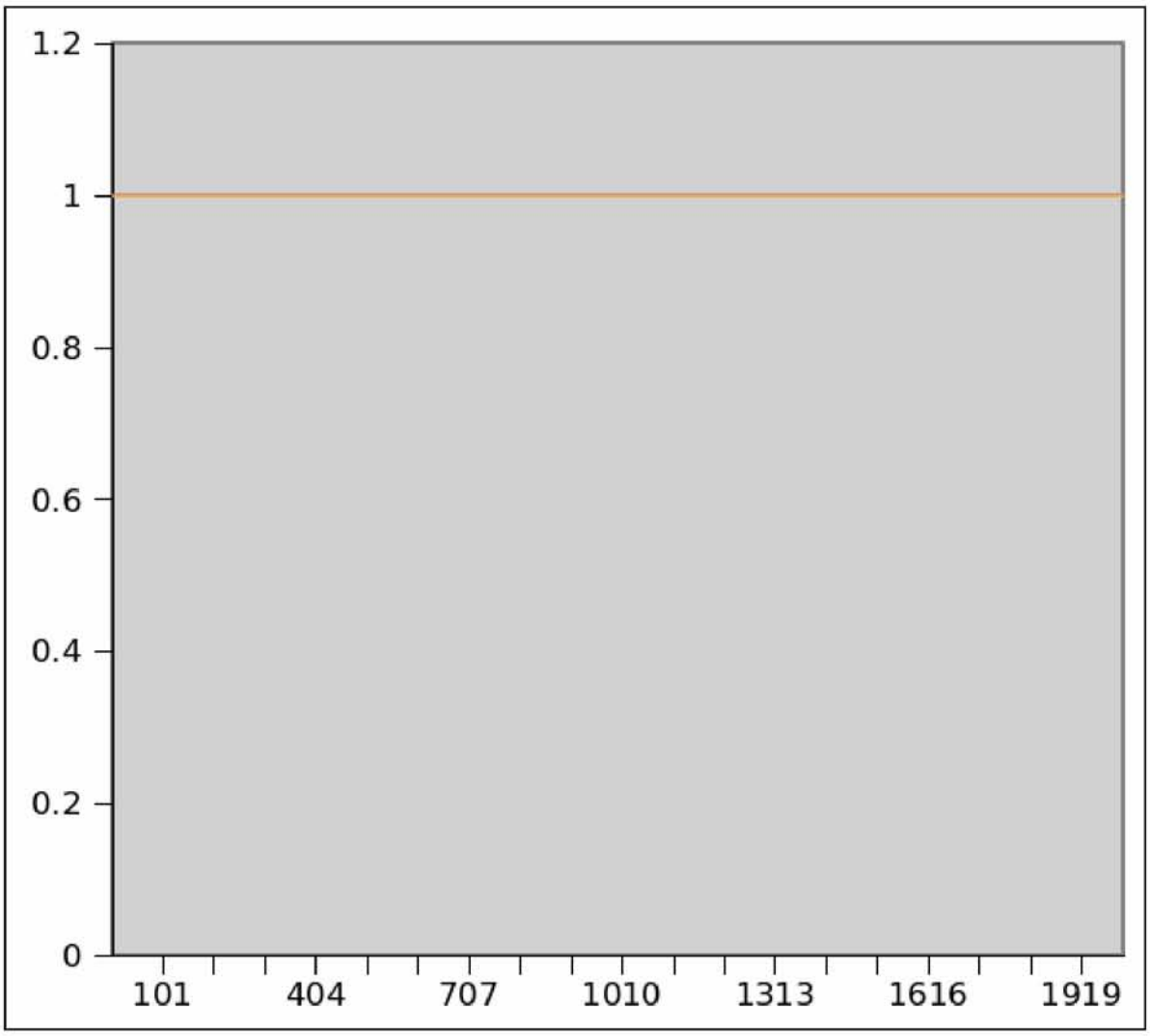




Επίσης ο οι αλλαγές στον ρυθμό ελέγχου έχουν μειωθεί σημαντικά, 225 για 50%, 271 για 70% και 409 για 90%. και ο μέσος όρος πλέον πλησιάζει το άνω όριο. Δηλαδή οι εναλλαγές δεν είναι τόσο συχνές οπότε δεν υπάρχει και ανάγκη για έλεγχος.

5.4.5 Σενάριο 5

Ένας κόμβος παρουσιάζει πολύ μεγαλύτερη πιθανότητα να εμφανίσει το μέγιστο.



Στην ιδανική αυτή περίπτωση η απάντηση θα είναι πάντα σωστή για οποιοδήποτε ποσοστό επιτυχίας που ορίζει ο χρήστης και ο ρυθμός ελέγχου θα είναι ο μικρότερος που ορίζεται καθώς το σύστημα δεν έχει αλλαγές οπότε δεν είναι απαραίτητοι και οι έλεγχοι.

5.6 Συμπέρασμα

Όπως ήταν αναμενόμενο το σύστημα εμφανίζει ικανοποιητική συμπεριφορά όταν οι αλλαγές στην πληροφορία είναι αναμενόμενες και σπάνιες. Το σύστημα στην χειρότερη περίπτωση λειτουργεί ντετερμινιστικά(σφάλμα 0%). όπως φαίνεται σε όλα τα σενάρια η απόδοση μειώνεται όσο μειώνεται το απαιτούμενο ποσοστό σφάλματος και το σύστημα τείνει να συμπεριφέρεται ντετερμινιστικά όταν αυτό συμβαίνει. Επίσης για μεγάλο περιθώριο σφάλματος η συμπεριφορά δεν αλλάζει ιδιαίτερα(πχ 30% και 40% δίνουν παρόμοια αποτελέσματα), κάτι που είναι φυσιολογικό αφού έχουμε θέσει κάτω όριο για τον ρυθμό ελέγχων. Η συνολική εκτίμηση είναι ότι σε ένα σύστημα χωρίς ιδιαίτερες

διακυμάνσεις ο αλγόριθμος μπορεί να αποδώσει στην μέση περίπτωση. Το ζήτημα που πρέπει να απασχολεί είναι το τι ακρίβεια ζητά ο χρήστης και τι απόδοση θέλει να έχει το σύστημα.

5.7 Βελτίωση

ο αλγόριθμος θα μπορούσε να βελτιωθεί περαιτέρω με κάποιους τρόπους:

Οι αισθητήρες θα μπορούν να προσαρμόζουν τον ρυθμό συλλογής των δεδομένων ανάλογα με την χρησιμότητα που έχουν στα query. Η γενική ιδέα είναι ότι ένας κόμβος που δεν έχει δώσει ποτέ απάντηση για το query δεν υπάρχει λόγος να παράγει δεδομένα με τον ίδιο ρυθμό που παράγει ένας κόμβος ο οποίος συνεχώς δίνει απαντήσεις.

Το σύστημα θα μπορεί να εκμεταλλευτεί τις σχέσεις μεταξύ των κόμβων. Πολύ συχνά μπορεί να παρατηρηθεί το φαινόμενο τα δεδομένα κάποιον κόμβων να σχετίζονται. Είτε αυτό σημαίνει ότι παράγουν κοντινές τιμές είτε ότι μια αλλαγή σε έναν κόμβο να συνεπάγεται αλλαγή και σε κάποιους άλλους. Στο παράδειγμα με τον αυτοκινητόδρομο η αύξηση κίνησης σε έναν κόμβο πολλές φορές μπορεί να ακολουθεί αύξηση και στον επόμενο κόμβο ή ένα τρακάρισμα να επηρεάζει πολλούς κόμβους μαζί. Αν το σύστημα καταφέρει να καταγράψει τέτοιες συσχετίσεις τότε μπορεί να βελτιώσει περισσότερο τις μεθόδους ελέγχου του.

Παράρτημα Α
Κώδικας

Data.java

```
package sense;
import java.util.Random;



/**
 *
 * @author Loukas Tzelis
 */
public class data implements Comparable{
    private int collected_data;
    private int timestamp;
    private int id;
    private static int counter =   0;
    public data(int timestamp,int id){
        this.timestamp  =   timestamp;
        this.id         =   id;
        collected_data  =   (new Random()).nextInt(100000);
        if (id ==   Network.total_nodes.length-1 ||id ==
Network.total_nodes.length-20){collected_data  =   (new
Random()).nextInt(Integer.MAX_VALUE);}
       // this line forces two nodes to produce much bigger
values,can be removed for non biased data

    }

    int getData(){
        return collected_data;
    }

    int getTimestamp(){
```

```
        return timestamp;
    }

    int getId(){
        return id;
    }

    public int compareTo(Object obj){
        data tmp = (data)obj;

        counter++;
        if(this.collected_data < tmp.collected_data)
            {
            /* instance lt received */
            return -1;
        }
        else if(this.collected_data > tmp.collected_data)
        {

            return 1;
        }
        else if(obj==null || this== null){return 1;}

            return 0;
    }

}
```

GeneralNode.java

```
package sense;

/**
 *
 * @author Loukas Tzelis
 */
public class GeneralNode {
    private int id;
    private int father;

    public GeneralNode(int id,int father){
        this.id     =   id;
        this.father =   father;
    }

    public int getId(){
        return id;
    }
```

```
    public int getFather(){
        return father;
    }




}
```

SensorNode.java

```
package sense;
import java.util.*;
/**
 *
 * @author Loukas Tzelis
 */
public class SensorNode extends GeneralNode{
    private data[] cur_data;
 //   private data[] old_data;
    private int   array_pos;
    private int   old_array_pos;
    private int count=0;
    private int neighbours[];


    public SensorNode(int id,int father, int left_brother, int
right_brother){
        super(id,father);

        cur_data    =   new data[200];
        neighbours  =   new int[2];
        //old_data    =   new data[100];
        array_pos   =   0;
        old_array_pos=  100;
        neighbours[0]   =   left_brother;
        neighbours[1]   =   right_brother;
       // for(int i=0;i<100;i++){cur_data[i]=-1;old_data[i]=-1;}
        //System.out.println("simplenode init");
    }

    void collectData(int timestamp){

        if (array_pos == 100){
            ageData();

        }
        cur_data[array_pos]   =     new data(timestamp,this.getId());
```

```
        array_pos++;
    }

    void ageData(){
        if (old_array_pos   ==  200) {old_array_pos =   100;}

        for(int i=0;i<10;i++){
            cur_data[old_array_pos] =  cur_data[i*10];
            old_array_pos++;
        }
        for(int i=0;i<100;i++){
            cur_data[i] =null;
        }
        array_pos       =   0;
    }

    //-------------------single value query---------------------
    public data getData(int timestamp){
        for(int i=0;i<cur_data.length;i++){
                if (cur_data[i].getTimestamp()   ==  timestamp)
return cur_data[i];
        }

        return null;
    }

    public data findValue(int value){
        for(int i=0;i<cur_data.length;i++){
            if (cur_data[i].getData()   ==  value) return
cur_data[i];
        }
        return null;
    }

    public int getNumOfData(){
        return cur_data.length;
    }


    public data getMax(){
        int max=Integer.MIN_VALUE;
        int pos =   -1;

        for (int i = 0; i < cur_data.length ; i++) {
            if (cur_data[i]!=null &&  cur_data[i].getData() > max)
{max = cur_data[i].getData();  pos=i;}

        }
```

```
        return cur_data[pos];

    }



    public data getMin(){
        int min=Integer.MAX_VALUE;
        int pos =   -1;
        for (int i = 0; i < cur_data.length; i++) {

            if (cur_data[i]!=null &&   cur_data[i].getData() <
min){min = cur_data[i].getData(); pos=i;}

        }

        return cur_data[pos];

    }

    public int getNeighbour(int pos){
        return neighbours[pos];

    }

    public void printData(){
        System.out.println("printing "+getId()+" after "+array_pos);
        for (int i = 0; i < array_pos; i++) {
            System.out.println("\t"+"value "+cur_data[i].getData()+"
time "+cur_data[i].getTimestamp());

        }
    }

}
```

SuperNode.java

```
package sense;

import java.util.*;

/**
 *
 * @author Loukas Tzelis
 */
public class SuperNode extends GeneralNode{
    private int[] nodes;
    private int[] trusted_nodes;
```

```
    public SuperNode(int id, int father, int num_of_children){
        super(id,father);
       // System.out.println("supernode init");
        nodes           =   new int[num_of_children];
        trusted_nodes   =   new int[10];
        //System.out.println("SuperNode "+nodes.length);
        nodeInit();
    }

    private void nodeInit(){
        for(int i=0;i<nodes.length;i++){nodes[i]=-1;}
        for(int j=0;j<10;j++){
            trusted_nodes[j]=nodes[nodes.length/2];
        }

    }

    public int[] getChilds(){
        return nodes;
    }

    public int isChild(int id){
        for(int i=0;i<nodes.length;i++){
            if (nodes[i]     ==  id) return i;
        }
        return -1;
    }

    public boolean setChild(int id){
        for(int i=0;i<nodes.length;i++){
            if (nodes[i] == -1){ nodes[i]    =   id; return true;}
        }
        return false;
    }

    public int getChildId(int pos){
        return nodes[pos];
    }

    public int numOfChildren(){
        return nodes.length;
    }

    public void setTrusted(int pos,int id){
        trusted_nodes[pos]  =   id;
    }

//reurns max stored by algorithm
```

```
    public data getMax(){
        data max;
        int pos;
        if (trusted_nodes[2]==-1)
{trusted_nodes[2]=nodes[nodes.length/2];}
       if (Network.total_nodes[trusted_nodes[2]] instanceof
SuperNode) {max =
((SuperNode)Network.total_nodes[trusted_nodes[2]]).getMax();}
       else {max    =
((SensorNode)Network.total_nodes[trusted_nodes[2]]).getMax();}

        return max;
    }
//gets actual max
    public data getSimpleMax(){
        data max=null;
        data temp;
        for (int i = 0; i < nodes.length; i++) {
            if(Network.total_nodes[nodes[i]] instanceof SuperNode){
                temp   =
((SuperNode)Network.total_nodes[nodes[i]]).getSimpleMax();
                if (max ==  null ||temp.getData()>max.getData()) max
=   temp;
            }
            else{
                temp   =
((SensorNode)Network.total_nodes[nodes[i]]).getMax();
                if (max ==  null || temp.getData()>max.getData()) max
=   temp;

            }

        }
        return max;

    }

}
```

Network.java

```
package sense;
import java.io.*;

/**
 *
 * @author Loukas Tzelis
 */
```

```
public class Network {
    private int num_of_nodes;
    private int num_of_super_nodes;
    private int cur_circle;
    private SuperNode root;
    public static GeneralNode[] total_nodes;
    public QueryStats stats;
    public int verify;
    private int trust;
    //filewriters for data used in simulation
    public FileWriter fstream_very;
    public BufferedWriter out_very;

    public FileWriter fstream_corr;
    public BufferedWriter out_corr;

    public FileWriter fstream_prob;
    public BufferedWriter out_prob;

    public FileWriter fstream_perc;
    public BufferedWriter out_perc;



    public Network(int num_of_nodes,int num_of_super_nodes, int
trust){
        System.out.println("initializing network...");
        cur_circle              =   1;
        this.num_of_nodes       =   num_of_nodes;
        this.num_of_super_nodes =   num_of_super_nodes;
        this.trust              =   trust;
        total_nodes             =   new GeneralNode[num_of_nodes +
num_of_super_nodes];

        stats   =   new QueryStats();
        verify  =   50;
        try{
          fstream_very = new FileWriter("outvery.txt");
          out_very = new BufferedWriter(fstream_very);

          fstream_corr = new FileWriter("outcorr.txt");
          out_corr = new BufferedWriter(fstream_corr);

          fstream_prob = new FileWriter("outprob.txt");
          out_prob = new BufferedWriter(fstream_prob);

          fstream_perc = new FileWriter("outperc.txt");
          out_perc = new BufferedWriter(fstream_perc);
        }
```

```
        catch (Exception e){//Catch exception if any
  System.err.println("Error: " + e.getMessage());
  }


        initializer();
    }
//inintializes the network
    private void initializer(){

        int cur_node    =   num_of_super_nodes;

        for(int i=0;i<num_of_super_nodes;i++){
            total_nodes[i]    =   new SuperNode(i,-
99,num_of_nodes/num_of_super_nodes);
            for(int j=0;j<num_of_nodes/num_of_super_nodes;j++){

                if (j   == 0) {total_nodes[cur_node]   =   new
SensorNode(cur_node,i,-1,cur_node+1);}
                else if (j  ==  num_of_nodes/num_of_super_nodes-1 ||
i ==  num_of_nodes ) {total_nodes[cur_node]=new
SensorNode(cur_node,i,cur_node-1,-1);}
                else {total_nodes[cur_node]   =   new
SensorNode(cur_node,i,cur_node-1,cur_node+1);}


if(((SuperNode)total_nodes[i]).setChild(total_nodes[cur_node].getId()
)){
                cur_node++;}
               // System.out.println("node created: "+i+" "+j+ " cur
"+cur_node);
            }
        }
        root    =   new SuperNode(-99,-99,num_of_super_nodes);
        for(int i=0;i<num_of_super_nodes;i++){
            root.setChild(total_nodes[i].getId());

        }
        runCircle();
        normalize();
        System.out.println("network initialized");

    }
//closing files
    public void finalizeit(){
      try{
        out_very.close();
        out_corr.close();
        out_prob.close();
```

```
        out_perc.close();
      }catch (Exception e){//Catch exception if any
         System.err.println("Error: " + e.getMessage());
      }
    }


    public SuperNode getRoot(){
        return root;
    }

    public int[]    getBrothers(int id){

return((SuperNode)total_nodes[total_nodes[id].getFather()]).getChilds
();
    }
    //returns max found by the algorithm
    public data getMax(){
        if (stats.getTrust()< trust){verify=verify/2;   if
(verify==0){verify=2;} normalize(); }
        if (stats.getTrust()> trust && verify <
100){verify=2*verify;}
        // System.out.println("max trust is "+stats.getTrust());
        return root.getMax();

    }
    //returns true max
    public data getSimpleMax(){
        return root.getSimpleMax();

    }

//checking if stored node gives the actual max
    public void verify(){
        data temp;
        data cur_temp;
        //   stats.print_stats();
        cur_temp    =   getMax();
        temp    =   getSimpleMax();

        if (cur_temp.getData()  ==
temp.getData()){stats.setTrust(true);}
        else{stats.setTrust(false);}




    }
//updates the node stored for true max
```

```
    public void normalize(){
        data temp;

         temp    =   getSimpleMax();

((SuperNode)total_nodes[((SensorNode)total_nodes[temp.getId()]).getFa
ther()]).setTrusted(2, temp.getId());
        root.setTrusted(2,
((SensorNode)total_nodes[temp.getId()]).getFather());

    }

    public void printData(data[] current_data){
        for (int i = 0; i < current_data.length; i++) {
            System.out.println(" id "+current_data[i].getId()+" value
"+current_data[i].getData()+" time "+current_data[i].getTimestamp());

        }

    }
//collecting data
     public void runCircle(){

           if (cur_circle%verify==0){

               verify();
            try{
                int temp    =   stats.getTrust();
              out_very.write("\n" + Integer.toString(verify));
              out_corr.write("\n" + Integer.toString(stats.counter));
              out_prob.write("\n" +
Integer.toString(stats.correct_counter));
              out_perc.write("\n" + Integer.toString(temp));


              }catch (Exception e){//Catch exception if any
              System.err.println("Error: " + e.getMessage());
              }

           }
           for(int
i=num_of_super_nodes;i<num_of_nodes+num_of_super_nodes;i++){
            ((SensorNode)total_nodes[i]).collectData(cur_circle);
        }
           cur_circle++;
      //    System.out.println("cicle "+cur_circle+ " run
successfully");
    }
```

```
    public void printChildren(int id){
        if (id  ==  -99){
            System.out.println(root.numOfChildren());
            System.out.println(id);
            for(int i=0;i<(root.getChilds()).length;i++){

System.out.println("\t"+total_nodes[root.getChildId(i)].getId());
                printChildren(root.getChildId(i));
            }
        }
        else if (total_nodes[id] instanceof SuperNode){
            for(int
i=0;i<((SuperNode)total_nodes[id]).numOfChildren();i++){

System.out.println("\t"+"\t"+total_nodes[((SuperNode)total_nodes[id])
.getChildId(i)].getId());
            }

        }

    }

    public void setTrust(int trust){
    this.trust  =   trust;
    }

}
```

QueryStats.java

```
package sense;
import java.util.*;
/**
 *
 * @author Loukas Tzelis
 */
public class QueryStats {
    private boolean max_trust[];
    int pos;
    int counter;
    int correct_counter;


    public QueryStats(){
         max_trust   =   new boolean[10];
        Arrays.fill(max_trust, Boolean.TRUE);
        pos=0;
        counter =   0;
        correct_counter =   0;
```

```
    }

    public int getTrust(){
       int max_counter =   0;

       for (int i = 0; i < max_trust.length; i++) {
            if (max_trust[i]==true){ max_counter++;}
        }
        return max_counter;

    }

    public void setTrust(boolean value){
        max_trust[pos]  =   value;
        pos++;
        pos =   pos%10;
        counter++;
        if(value    ==  true){        correct_counter++;}

    }

    public void print_stats(){
        System.out.println("from "+counter+" correct answers were
"+correct_counter+" percentace
"+(float)(((float)correct_counter)/((float)counter))+" for trust
"+getTrust());
    }


}
```

Main.java

```
package sense;
import java.io.*;

/**
 *
 * @author Loukas Tzelis
 */
public class Main {

    public  static     FileWriter fstream;// writing percentance of
succes to out.txt
    public  static    BufferedWriter out;

    /**
     * @param args the command line arguments
```

```
     */
    public static void main(String[] args) {



    try{

          fstream = new FileWriter("out.txt");
          out = new BufferedWriter(fstream);
        }
        catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
        }

       Network complex_network =   new Network(100,10,9);//our
network

       for(int i=1;i<500;i++){
       complex_network.runCircle();}

       complex_network.verify();
       int counter =   1;
       int corr_counter    =   1;
       data max;

       data simple_max;

       for (int i = 0; i < 20000; i++) {//runing for 20000 circles,
quering max every 10 circles
           complex_network.runCircle();
           if (i%10 ==  0){
               counter++;
               max  =   complex_network.getMax();
               simple_max   =   complex_network.getSimpleMax();
               if (max.getData()    ==
simple_max.getData()){corr_counter++;}

               try{

                    out.write("\n" +
Float.toString((float)(((float)corr_counter)/((float)counter))));
                 }
              catch (Exception e){
                 System.err.println("Error: " + e.getMessage());
              }

         }

       }
```

```
        try{
           out.close();
         }catch (Exception e){
             System.err.println("Error: " + e.getMessage());
         }

         complex_network.finalizeit();
   }


}
```

Βιβλιογραφία

[1] W. Heinzelman, J. Kulik, and H. Balakrishnan, <Adaptive Protocols for Information Dissemination in Wireless SensorNetworks>

[2] C. Intanagonwiwat, R. Govindan, and D. Estrin, <Directed diffusion: a scalable and robust communication paradigm for sensor networks>

[3] D. Braginsky and D. Estrin, <Rumor Routing Algorithm for Sensor Networks>

[4] F. Ye, A. Chen, S. Liu, L. Zhang, <A scalable solution to minimum cost forwarding in large sensor networks>

[5] C. Schurgers and M.B. Srivastava, <Energy efficient routing in wireless sensor networks>

[6] M. Chu, H. Haussecker, and F. Zhao, <Scalable Information-Driven Sensor Querying and Routing for ad hoc Heterogeneous Sensor Networks>

[7] N. Sadagopan et al., <The ACQUIRE mechanism for efficient querying in sensor networks>

[8] R. C. Shah and J. Rabaey, <Energy Aware Routing for Low Energy Ad Hoc Sensor Networksβ>

[9] S. Servetto and G. Barrenechea, <Constrained Random Walks on Random Graphs: Routing Algorithms for Large Scale Wireless Sensor Networks>

[10] M. J. Handy, M. Haase, D. Timmermann, <Low Energy Adaptive Clustering Hierarchy with Deterministic Cluster-Head Selection>

[11] S. Lindsey, C. Raghavendra, <PEGASIS: Power-Efficient Gathering in Sensor Information Systems>

[12] A. Manjeshwar and D. P. Agarwal, <TEEN: a routing protocol for enhanced efficiency in wireless sensor networks>

[13] A. Manjeshwar and D. P. Agarwal, <APTEEN: A hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks>

[14] L. Subramanian and R. H. Katz, <An Architecture for Building Self Configurable Systems>

[15] Y. Xu, J. Heidemann, D. Estrin,<Geography-informed Energy Conservation for Ad-hoc Routing>

[16] Y. Yu, D. Estrin, and R. Govindan, <Geographical and Energy-Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks>

[17] I. Stojmenovic and X. Lin. <GEDIR: Loop-Free Location Based Routing in Wireless Networks>

[18] F. Kuhn, R. Wattenhofer, A. Zollinger,<Worst-Case optimal and average-case efficient geometric ad-hoc routing>

[19] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris, <SPAN: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks>

[20] Tian He, John A Stankovic, Chenyang Lu, Tarek Abdelzaher<SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks>

[21] Mehdi Kalantari and Mark Shayman, <Energy Efficient Routing in Wireless Sensor Networks>

[22] Samuel Madden, Michael J. Franklin, and Joseph M. Hellerstein, <TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks>

[23] SAMUEL R. MADDEN, TinyDB: An Acquisitional Query Processing System for Sensor Networks

[24] Samuel Madden, Michael J. Franklin, <Fjording the Stream: An Architecture for Queries over Streaming Sensor Data>

[25] Y. Yao and J. Gehrke, <The cougar approach to in-network query processing in sensor networks>

[26] Yanlei Diao, Deepak Ganesan, Gaurav Mathur, and Prashant Shenoy, <Rethinking Data Management for Storagecentric Sensor Networks>

[27] H. Yang, F. Ye and B. Sikdar, <A Dynamic Query-tree Energy Balancing Protocol for Sensor Networks>

[28] Amol Deshpande, Carlos Guestrin, <Model-Driven Data Acquisition in Sensor Networks>

[29] Ming Li, Deepak Ganesan, and Prashant Shenoy, <PRESTO: Feedback-driven Data Management in Sensor Networks>

[30] Q. Li and J. Aslam and D. Rus, <Hierarchical Power-aware Routing in Sensor Networks>

[31] F. Ye, H. Luo, J. Cheng, S. Lu, L. Zhang, ,A Two-tier data dissemination model for large-scale wireless sensor networks.

[32] Reynold Cheng, Dmitri V. Kalashnikov, Sunil Prabhakar, <Evaluating Probabilistic Queries over Imprecise Data>

[33] http://en.wikipedia.org

[34] Ian F. Akyildiz ,Wireless Sensor Networks