# Εργασία TinyOS

## 2η Φάση

Ημερομηνία Παράδοσης : 8 Δεκεμβρίου 2017

Επεξεργασία και Διαχείριση Δεδομένων σε Δίκτυα Αισθητήρων

Διδάσκων : Α. Δεληγιαννάκης

Σταματάκης Γεώργιος - 2013030154

Σκυβαλάκης Κωνσταντίνος - 2013030034

GitHub exercise repo : `https://github.com/gstamatakis/SensorsTinyOS`

# Πρόγραμμα 2

## Προσθήκες κώδικα

Για τον κώδικα του προγράμματος 2 διατηρήσαμε τον routing tree αλγόριθμο που χρησιμοποιήσαμε στην 1η φάση για το TAG. Στη συνέχεια, προσθέσαμε κώδικα στη βιβλιοθήκη SimpleRoutingTree.h σχετικά με τα μηνύματα των advertisements που στέλνουν οι Cluster Heads (CH) καθώς επίσης και για τα μηνύματα των advertisement response που κάνουν οι κόμβοι που λαμβάνουν τις διαφημίσεις και αποφασίζουν να ορίσουν τον εκάστοτε CH ως ηγέτη τους για να του στέλνουν τις μετρήσεις τους. Α-κόμη, προσθέσαμε κάποιους timers σχετικούς με την υλοποίηση του LEACH (π.χ. χρονική διάρκεια του γύρου). Επιπλέον, όπως και στην 1η φάση προσθέσαμε και στο αρχείο SRTreeAppC.nc τις απαραίτητες αλλαγές για να μπορούμε να χρησιμοποιήσουμε τις καινούριες ουρές μηνυμάτων που θα φτιάξουμε, τους timers καθώς επίσης και τα ίδια τα μηνύματα.

Οι σημαντικότερες αλλαγές και προσθήκες κώδικα που κάναμε είναι αυτές που έγιναν στο αρχείο SRTreeC.nc και είναι οι εξής :

- Συνάρτηση εκλογής CH
- Συγχρονισμός timers
- Αποστολή Ad μηνυμάτων και responses αυτών

Ο σχεδιασμός της υλοποίησης μας βασίστηκε πάνω στα 3 παρακάτω κομμάτια της κάθε εποχής :

1. Οι CHs στέλνουν τα advertise μηνύματα.
2. Οι Non-CH κόμβοι κάνουν join τον CH που επιλέγουν.
3. Υπολογισμός Aggregate Queries.

Αναλυτικότερα σχετικά με τη συνάρτηση εκλογής των CH. Η συνάρτηση καλείται μόνο μια φορά σε κάθε γύρο μιας και οι CHs που εκλέγονται παραμένουν σταθεροί μέχρι το τέλος του γύρου. Αυτό που επιστρέφει είναι μια boolean μεταβλητή η οποία δηλώνει αν ο κόμβος ο οποίος την κάλεσε είναι CH. Η διαδικασία εκλογής τωνCHs βασίζεται εξόλοκλήρου σε αυτά που περιγράφει το paper του LEACH. Πιο συγκεκριμένα για να ελέγξουμε αν ο κάθε κόμβος είναι CH σε αυτόν τον γύρο, ελέγχουμε αν ήταν CH τους τελευταίους $1/P$ γύρους και αν ο τυχαίος αριθμός που 'τράβηξε' ο κόμβος στο διάστημα $[0, 1]$ είναι μικρότερος από το threshold που αναγράφεται στο paper.

Στην πρώτη φάση της εποχής, για την αποστολή των ad μηνυμάτων δημιουργήσαμε τις απαραίτητες ουρές, timers και events. Αν ένας κόμβος είναι CH τότε κάνει broadcast τα ads του και όσοι τον ακούνε αποφασίζουν αν θα τον επιλέξουν για ηγέτη. Στη δεύτερη φάση, οι υπόλοιποι κόμβοι που δεν είναι ηγέτες, επιλέγουν τον ηγέτη τους τον οποίο και διατηρούν μέχρι τις επόμενες εκλογές και του προωθούν τα δεδομένα τους. Στην τρίτη και τελευταία φάση, αν ο κόμβος που ελέγχεται είναι CH τότε προσθέτει και τα δικά του δεδομένα στο πακέτο και απλά προωθεί τα δεδομένα που έχει συλλέξει προς τα πάνω στο δέντρο με τελικό προορισμό τη ρίζα. Αν ο κόμβος που ελέγχεται δεν είναι CH, τότε προσθέτει και αυτός τα δικά του δεδομένα στο πακέτο και το προωθεί προς τον CH του είτε απευθείας είτε μέσω ενδιάμεσων κόμβων.

Σύμφωνα με τη δομή της εποχής που ορίσαμε παραπάνω έτσι και στον συγχρονισμό των timers, ελέγξαμε ο timer των ads να χτυπάει μόνο μια φορά σε κάθε γύρο ενώ οι απαντήσεις των κόμβων σε ένα ad μπορεί να συμβούν πολλές φορές μέσα σε ένα γύρο,μιας και έχει να κάνει σχέση με την επιλογή του ηγέτη και την προώθηση των δεδομένων των κόμβων, εξού και ο timer που χρησιμοποιείται. Τέλος, ο MsgTimer έχει να κάνει σχέση με την προώθηση των δεδομένων (ανεξαρτήτως κόμβου) σε στιγμές που δε χρειάζεται να γίνει επιλογή ηγέτη από τους κόμβους.

### Αφαίρεση κώδικα

Γενικά, αφαιρέσαμε πολύ μικρά σε έκταση κομμάτια κώδικα, μόνο ίσως κάτι μικρές λεπτομέρειες που θα μπορούσαμε να είχαμε αφαιρέσει από την πρώτη φάση χωρίς να επηρεάζεται το αποτέλεσμα, όπως π.χ. κάποια else if που δεν εκτελούνταν ποτέ.

## Παραπομπές σε Κώδικα

Λόγω της μεγάλης έκτασης του σημαντικού κατά εμάς κώδικα, δε θα παραθέσουμε screenshots, αλλά αντί αυτού παραπομπές σε σημεία της υλοποίησης μας τα οποία θεωρούμε σημαντικά. Τα παρακάτω κομμάτια κώδικα παρέχονται με αρκετά σχόλια και είναι ουσιαστικά ενσωματωμένα στην ανάλυση παραπάνω.

- bool IamClusterhead() , γραμμή : 871
- task void receiveAdResponseTask() , γραμμή : 834
- event void AdResponseTimer.fired() , γραμμή : 524
- event void AdResponseTimer.fired() , γραμμή : 463
- event void AdTimer.fired() , γραμμή : 404

Να προσθέσουμε ότι στα ad responses που συλλέγει ο CH διατηρούμε όπως και στην πρώτη φάση του πρότζεκτ μια μικρή cache με τις τελευταίες τιμές που έχουμε συλλέξει, έτσι ώστε να δίνουμε όσο το δυνατό σωστότερα αποτελέσματα σε περιπτώσεις αποτυχίας αποστολής ή λήψης κάποιων μηνυμάτων.

# Αποτελέσματα

Παραθέτουμε μαζί με τα αρχεία της εργασίας ένα αρχείο που βρίσκεται στον φάκελο Logs με όνομα logfile_25nodes_LEACH.txt το οποίο περιλαμβάνει τα αποτελέσματα της εκτέλεσης για μια εποχή τα οποία και σχολιάζουμε παρακάτω. Να σημειωθεί ότι το topology έχει διατηρηθεί ίδιο με την πρώτη φάση της εργασίας.

```
0:0:10.000000020 DEBUG (0): AdTimer.fired()
0:0:10.000000020 DEBUG (0): CH decision with LeachRound:0 EPOCH:0 RandResult:0.246021
0:0:10.000000020 DEBUG (0): CH at depth: 0 at round 0 | ROOT
0:0:10.000000020 DEBUG (0): I am a CH!
0:0:10.000000020 DEBUG (0): AdMsg sending...!!!!
0:0:10.000000020 DEBUG (0): sendAdTask() posted!!
0:0:10.000000020 DEBUG (0): AdMsg enqueued successfully in AdSendQueue!!!
0:0:10.000000030 DEBUG (0): sendAdTask(): AdAMSend.send success!
```

Σχήμα 1: Root as a CH sends the first ad message.

```
0:0:10.007736190 DEBUG (6): AdReceive.receive from 0
0:0:10.007736190 DEBUG (5): AdReceive.receive from 0
0:0:10.007736190 DEBUG (1): AdReceive.receive from 0
0:0:10.007736200 DEBUG (6): Accepted Ad from 0
0:0:10.007736200 DEBUG (5): Accepted Ad from 0
0:0:10.007736200 DEBUG (1): Accepted Ad from 0
0:0:10.007904036 DEBUG (0): AdAMSend.sendDone: True
0:0:10.197845484 DEBUG (6): curdepth= 1 , parentID= 0
0:0:10.197845484 DEBUG (5): curdepth= 1 , parentID= 0
0:0:10.197845484 DEBUG (1): curdepth= 1 , parentID= 0
```

Σχήμα 2: Advertisement receive from root.

```
0:0:10.197845494 DEBUG (1): CH decision with LeachRound:0 EPOCH:0 RandResult:0.089995
0:0:10.197845494 DEBUG (1): CH at depth: 1 at round 0| LeachRound == 0
0:0:10.197845494 DEBUG (1): I am a CH!
0:0:10.197845494 DEBUG (1): AdMsg sending...!!!!
0:0:10.197845494 DEBUG (1): sendAdTask() posted!!
0:0:10.197845494 DEBUG (1): AdMsg enqueued successfully in AdSendQueue!!!
0:0:10.197845504 DEBUG (1): sendAdTask(): AdAMSend.send success!
```

Σχήμα 3: Node 1 CH election.

Στο σχήμα 1 η ρίζα (0) στέλνει το πρώτο ad μήνυμα σας γνωστός CH, ενώ στο σχήμα 2 οι κόμβοι 6,5 και 1 λαμβάνουν το ad μήνυμα της ρίζας την οποία και επιλέγουν ως ηγέτη τους. Στο σχήμα 3, ο κόμβος 1 εκλέγεται ηγέτης μιας και δεν είχε ξαναεκλεγεί τους τελευταίους $1/P$ γύρους και πληροί την προϋπόθεση του threshold. Υπάρχει περίπτωση κάποιος κόμβος να απορρίψει κάποιον άλλο κόμβο $j$ για ηγέτη του αν ο ίδιος είναι ηγέτης με το εξής μήνυμα : Discarding received Ad from $j$ because I am a CH!. Όπως επίσης μπορεί και κάποιος κόμβος να μην εκλεγεί ηγέτης, όπου τότε ενημερώνει με το αντίστοιχο μήνυμα : I am NOT a CH!. Η παραπάνω διαδικασία συνεχίζεται μέχρις ώτου έχουν εκλεγεί όλοι οι CHs και τότε αρχίζουν τα ad response μηνύματα με τις μετρήσεις του κάθε κόμβου.

```
0:0:28.125000012 DEBUG (24): Responding to CH 23: LeachRound: 1 Epoch:0 sum:44 sum_squared:1936 curdepth:4
0:0:28.125000012 DEBUG (24): AdResponseTimer.fired(): Task posted.
0:0:28.125000022 DEBUG (24): sendAdResponse(): AdResponseAMSend.send success!
0:0:28.127868658 DEBUG (23): AdResponseReceive.receive from 24
0:0:28.127868668 DEBUG (23): receiveAdResponseTask() CH:YES
0:0:28.127868668 DEBUG (23): receiveAdResponseTask() Caching at 0: ID:24 SUM:44 COUNT:1
0:0:28.128036503 DEBUG (24): AdResponseAMSend.sendDone: True
```

Σχήμα 4: Node 24 ad response to node 23.

Στο σχήμα 4, ο κόμβος 24 προωθεί τα δεδομένα του στον κόμβο 23 και έτσι ο κόμβος 23 ξέρει ότι ο 24 τον έχει επιλέξει ως ηγέτη.

```
0:1:8.593750010 DEBUG (0): ROOT readings: 518 13748 22
0:1:8.593750010 DEBUG (0): ######## ROUND 1 ########

0:1:8.593750010 DEBUG (0): LEACH ROUND:1 EPOCH:1 -> sum: 518 | count: 22 | sum_squared: 13748 | avg: 23.55 | var: 70.52
```

Σχήμα 5: Root prints the results of epoch 1.

```
0:0:48.769531260 DEBUG (5): MsgTimer fired!
0:0:48.769531260 DEBUG (5): Cache val[0]: 38 730 2 from 10
0:0:48.769531260 DEBUG (5): STANDARD_NODE: ParentID:0 Epoch:1 sum:38 count:2 sum_sq:730 curdepth:1
0:0:48.769531260 DEBUG (5): MsgTimer.fired(): Task posted.
0:0:48.769531270 DEBUG (5): sendQueryTask(): QueryAMSend.send success!
0:0:48.775894148 DEBUG (0): QueryReceive.receive from 5
0:0:48.775894158 DEBUG (0): QueryReceived from 5 with values: 38 730 2
0:0:48.776061994 DEBUG (5): QueryAMSend.sendDone: True
```

Σχήμα 6: Node 5 forwards his data.

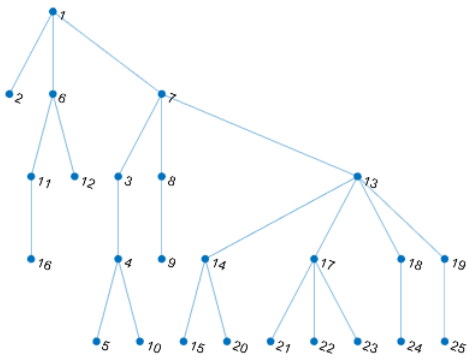
Στο σχήμα 6 ο κόμβος 5 προωθεί τα δεδομένα τους στον CH του που είναι ο 0, η ρίζα δηλαδή. Επιπλέον, σε αυτό το κομμάτι βλέπουμε την 3η φάση που περιγράψαμε παραπάνω.

```
0:0:48.183593761 DEBUG (15): MsgTimer fired!
0:0:48.183593761 DEBUG (15): Cache val[0]: 21 441 1 from 20
0:0:48.183593761 DEBUG (15): CH readings: 38 730 2
0:0:48.183593761 DEBUG (15): CH Aggregate: ParentID:10 Epoch:1 sum:38 count:2 sum_sq:730 curdepth:3
0:0:48.183593761 DEBUG (15): MsgTimer.fired(): Task posted.
0:0:48.183593771 DEBUG (15): sendQueryTask(): QueryAMSend.send success!
0:0:48.186660770 DEBUG (10): QueryReceive.receive from 15
0:0:48.186660780 DEBUG (10): QueryReceived from 15 with values: 38 730 2
0:0:48.186828615 DEBUG (15): QueryAMSend.sendDone: True
```

Σχήμα 7: CH node 15 forwards his data.

Στο σχήμα 7 ο κόμβος 15 που είναι και CH προωθεί με τη σειρά του τα δεδομένα του μαζί και των υποδέντρων του προς τα πάνω στο δέντρο μέσω του κόμβου 10 ο οποίος δεν είναι CH (θα μπορούσε και να ήταν) και τελικό προορισμό τη ρίζα.

Για λόγους ευκολίας και καλύτερης κατανόησης των μηνυμάτων στο logfile παραθέτουμε το δέντρο που έχει προκύψει μετά το routing. Υπενθυμίζουμε ότι όπως και στην πρώτη φάση τα ID των κόμβων που φαίνονται στο δέντρο είναι αυτά που αναγράφονται μείον 1 λόγω MATLAB.




Σχήμα 8: Topology of sensors after routing.

# Καταμερισμός Εργασιών

Τα παραπάνω κομμάτια της δεύτερης φάσης της εργασίας ολοκληρώθηκαν με δίκαιο καταμερισμό και συνεργασία σε κάθε στάδιο. Όλοι οι κώδικες συγγράφηκαν από κοινού ενώ και οι δυο μας προτείναμε διορθώσεις και διαφορετικές λύσεις ο ένας στον άλλο σε κάποιες περιπτώσεις.

# Βιβλιογραφία

[1] Energy-Efficient Communication Protocol for Wireless Microsensor Networks
https://www.gta.ufrj.br/wsns/Routing/leach.pdf