Εργασία στο TinyOS

Βασική σελίδα με περιγραφή του TinyOS: http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials

ΒΗΜΑ 1: ΕΓΚΑΤΑΣΤΑΣΗ

Στο https://drive.google.com/file/d/1LTqqCUllwXvIZ59FbVKaMY3GFFxJ6qla/view?usp=sharing θα βρείτε ένα virtual machine (για Ubuntu 64bit λειτουργικά) όπου είναι εγκατεστημένο το TinyOS και ο βοηθητικός κώδικας. Εγώ το τρέχω από VirtualBox.

Ο χρήστης μου λέγεται tinyos και ο κωδικός είναι ο tinyosRULES.

ΠΡΟΣΟΧΗ: Σε καμία περίπτωση μην κάνετε ενημερώσεις λογισμικού στο virtual machine, καθώς αν κατεβάσετε νέες εκδόσεις του μεταγλωττιστή θα χρειαστείτε πιθανώς νέα εγκατάσταση (που δε θα σας κάνω ατομικά εγώ) του TinyOS. Εγώ έχω απενεργοποιήσει τις αυτόματες ενημερώσεις στο vdi που σας δίνω – μην τις ενεργοποιήσετε.

Μπορεί από τα settings να αλλάξετε (αν θέλετε) τις ρυθμίσεις της ανάλυσης της οθόνης (Display), σύμφωνα με αυτές του υπολογιστή σας.

ΒΗΜΑ 2: ΒΟΗΘΗΤΙΚΟΣ ΚΩΔΙΚΑΣ

Ο βοηθητικός κώδικας βρίσκεται **στο φάκελο /home/tinyos/local/src/tinyos-2x/apps/tinyOS.**

Ο φάκελος έχει μέσα ένα αρχείο Makefile και ένα αρχείο README. Δοκιμάστε να κάνετε compile τον κώδικα με την εντολή:
make micaz sim
Τρέχετε το simulation με την εντολή:
python ./mySimulation.py

ΒΗΜΑ 3: ΚΑΤΑΝΟΗΣΗ ΒΟΗΘΗΤΙΚΟΥ ΚΩΔΙΚΑ

1. ΤΟΠΟΛΟΓΙΑ

Τα αρχεία topology.txt και topology2.txt περιέχουν πληροφορίες για τη συνδεσιμότητα 2 κόμβων σε simulation mode. Εξετάζοντας το αρχείο topology.txt, γειτονικά ζευγάρια κόμβων είναι τα:
0 με 1
1 με 4
1 με 7
1 με 2
4 με 5
Όσα ζευγάρια δεν αναφέρονται στο topology.txt δε θα επικοινωνούν μεταξύ τους.

2. Αρχείο του Simulation

Τα αρχικά στάδια της εργασίας σας θα γίνουν σε simulation mode. Συνεπώς, είναι σημαντικό να κατανοήσετε (σε γενικές γραμμές) το αρχείο
mySimulation.py
το οποίο τρέχει την προσομοίωση.

Οι γραμμές:
```
from TOSSIM import *
import sys ,os
import random
t=Tossim([])
```
δημιουργούν ένα αντικείμενο του Tossim.

Οι γραμμές:
```
for i in range(0,10):
        m=t.getNode(i)
        m.bootAtTime(10*t.ticksPerSecond() + i)
```
εκκινεί τους κόμβους (από 0..9) σε λίγο διαφορετικές στιγμές.

Οι εντολές
```
f=sys.stdout
t.addChannel("Boot",f)
t.addChannel("RoutingMsg",f)
t.addChannel("NotifyParentMsg",f)
t.addChannel("Radio",f)
#t.addChannel("Serial",f)
t.addChannel("SRTreeC",f)
#t.addChannel("PacketQueueC",f)
```
Καθορίζουν ποια dbg μηνύματα θα εκτυπώνονται στην οθόνη. Αν πχ το πρώτο όρισμα ενός dbg μηνύματος είναι Boot, το αντίστοιχο μήνυμα θα εκτυπωθεί, αφού έχει προστεθεί το κανάλι Boot. Το σύμβολο '#' είναι το σύμβολο για σχόλιο γραμμής.

3. Ουρά πακέτων

Τα αρχεία PacketQueue.nc και PacketQueueC.nc παρέχουν ένα interface και ένα module (αντίστοιχα) για μία ουρά πακέτων. Αυτές οι ουρές χρησιμοποιούνται συχνά στο βοηθητικό κώδικα που σας παρέχεται.

4. Κύριο module: SRTreeC.nc και configuration: SRTreeAppC.nc

Το κεντρικό αρχείο είναι το SRTreeAppC.nc, όπου γίνεται η καλωδίωση (wiring) όλων των components. Θα πρέπει να ξεχωρίσετε:

a) το component MainC (για την εκκίνηση – θα υλοποιήσουμε το event Booted στο SRTreeC).
b) το component LedsC για το χειρισμό των Leds του αισθητήρα
c) για αποστολή/λήψη μηνυμάτων στον ασύρματο, τα components ActiveMessageC, AMSenderC (2: ένα για routing μηνύματα, και ένα για μηνύματα προς τον πατέρα), και AMReceiverC (2: ένα για routing μηνύματα, και ένα για μηνύματα προς τον πατέρα).
d) για αποστολή/λήψη μηνυμάτων στο serial port, τα components SerialActiveMessageC, SerialAMSenderC, και SerialAMReceiverC.
e) Components για τις ουρές μηνυμάτων σχετικά με την αποστολή/λήψη των

αντίστοιχων 2 τύπων μηνυμάτων.

f) Διάφορους μετρητές (Timers) με ακρίβεια ms.

Αναζητήστε πως ο κόμβος 0 ξεκινάει ένα μετρητή για να στείλει το πρώτο routing μήνυμα. Παρατηρούμε ότι όταν χτυπάει ο μετρητής, κάνει στο τέλος post ένα task για την πραγματική αποστολή του μηνύματος.

Αντίστοιχα, στο αντίστοιχο task (receiveRoutingTask) λήψης του μηνύματος, ο κάθε κόμβος που δεν είχε πατέρα θέτει το επίπεδό του και τον πατέρα του, και στη συνέχεια στέλνει ένα μήνυμα NotifyParent στον πατέρα του, προτού προωθήσει το μήνυμα. Το συγκεκριμένο task είναι αρκετά σύνθετο, καθώς περιέχει και επιλογές για την περίπτωση που ενημερωθούμε για έναν καλύτερο πατέρα από αυτόν που έχουμε αρχικά επιλέξει.

ΕΡΓΑΣΙΑ

Η εργασία αποτελείται από κομμάτια, τα οποία βασίζονται το ένα στο άλλο:

- Ένα βοηθητικό πρόγραμμα που θα δημιουργεί τις τοπολογίες στις οποίες θα ελέγξετε το πρόγραμμά σας.
- Το πρώτο κομμάτι θα σας μάθει πώς να χρησιμοποιείτε μετρητή (ρολόι) και να στέλνετε/λαμβάνετε μηνύματα, υλοποιώντας μία συναθροιστική συνάρτηση **σύμφωνα με το TAG**.
- Στο δεύτερο κομμάτι θα πρέπει να υλοποιήσετε τη λειτουργία του LEACH. Θα χρειαστείτε το πρώτο κομμάτι για την επικοινωνία των clusterheads (ηγετών) με το σταθμό βάσης.
- Υπάρχει και ένα επιπλέον κομμάτι για τους μεταπτυχιακούς φοιτητές μόνο.

Πιο συγκεκριμένα, καλείστε να κάνετε τα ακόλουθα.

ΒΟΗΘΗΤΙΚΟ ΠΡΟΓΡΑΜΜΑ (Προθεσμία: 01/12/2021)

Για να μπορέσετε να ελέγξετε τη λειτουργία των προγραμμάτων σας, απαιτείται η δημιουργία αρχείων τοπολογίας, που θα περιλαμβάνουν πολλούς αισθητήρες, με «αυτόματο» τρόπο. Στο βοηθητικό πρόγραμμα αυτό καλείστε να δημιουργήσετε (σε όποια γλώσσα προγραμματισμού θέλετε) ένα πρόγραμμα που να:

- Παίρνει ως παραμέτρους 1 ακέραιο (θα αναφερόμαστε σε αυτόν ως διάμετρος D) και 1 αριθμό κινητής υποδιαστολής (θα αναφερόμαστε σε αυτόν ως εμβέλεια).
- Θα δημιουργεί D*D κόμβους, με αναγνωριστικά από 0 έως D^2-1, τοποθετημένους σε ένα grid μεγέθους D*D. Ο κόμβος j θα ανήκει στη γραμμή j/D και στη στήλη j%D.
- Θεωρώντας ότι οι οριζόντιες και κάθετες αποστάσεις των κόμβων στο grid είναι ίσες με 1, είναι εύκολο για έναν οποιοδήποτε κόμβο να βρείτε όλους τους κόμβους που βρίσκονται σε απόσταση μικρότερη ή ίση με την εμβέλειά του (δεύτερη παράμετρος του προγράμματος). Πχ, αν η εμβέλεια είναι 1.5, τότε ένας κεντρικός κόμβος έχει 8 γείτονες (σε σχηματισμό αστεριού γύρω από αυτόν, δηλαδή πάνω/κάτω/αριστερά,δεξιά και στις διαγωνίους.
- Αν για κάθε κόμβο βρείτε τους γείτονές του, τότε μπορείτε αυτή την πληροφορία να τη χρησιμοποιήσετε για να δημιουργήσετε ένα αρχείο τοπολογίας, με παρόμοια μορφή με αυτή που σας δίνεται στο αρχείο topology.txt.

ΠΡΟΓΡΑΜΜΑ 1 (Προθεσμία: 01/12/2021)

Κάθε αισθητήρας θα πρέπει κάθε 30 δευτερόλεπτα να παράγει μία τυχαία τιμή ως μέτρησή του και να προωθεί στον πατέρα του κατάλληλη πληροφορία ώστε να μπορεί να υπολογιστεί τελικά ο μέσος όρος και η διασπορά των μετρήσεων σε όλο το δίκτυο.

Η λειτουργία του κώδικά σας ΠΡΕΠΕΙ να γίνεται **ΣΥΜΦΩΝΑ ΜΕ ΤΟ TAG**, διορθώνοντας πράγματα που υπάρχουν στο βοηθητικό κώδικα που σας δίνω, όταν κάτι δε συμβαδίζει με τη λογική του TAG.

Συνεπώς, υλοποιούμε τις συναρτήσεις AVG και VARIANCE, **οι οποίες υπολογίζονται ταυτόχρονα**. Η λειτουργία να τερματίζεται μετά από 300 δευτερόλεπτα. Ο σταθμός βάσης (κόμβος 0) θα πρέπει να τυπώνει το τελικό αποτέλεσμα σε κάθε εποχή.

Η τιμή κάθε κόμβου με αναγνωριστικό Κ σε κάθε εποχή θα είναι ένας τυχαίος ακέραιος αριθμός στο διάστημα [K..K+40].

Ουσιαστικά, ο κάθε κόμβος είναι καλό να γνωρίζει σε κάποια μεταβλητή τις αντίστοιχες **τελευταίες τιμές που έχει λάβει από τα παιδιά του** και στη συνέχεια να υπολογίζει την πληροφορία που πρέπει να μεταδώσει για το υποδέντρο του.

Το συγκεκριμένο πρόγραμμα ΔΕΝ απαιτεί συγχρονισμό των κόμβων με βάση το TAG και δεν απαιτεί να ανοιγοκλείνετε τον ασύρματο. Θα πρέπει να προσπαθήσετε (μπορεί να μην τη καταφέρετε τέλεια, αλλά έστω μερικώς θα πρέπει να γίνει) να πετύχετε όμως τη λογική του TAG (πρώτα μεταδίδουν τα παιδιά, μετά οι γονείς) χρησιμοποιώντας το επίπεδο του κάθε κόμβου στο δέντρο. Σκεφτείτε ότι η λειτουργία που θα κάνει ο κάθε κόμβος θα είναι επαναλαμβανόμενη και θα εξαρτάται από ένα μετρητή. Αν το σκεφτείτε καλά, πρέπει να ρυθμίσετε πότε θα χτυπήσει πρώτη φορά ο μετρητής αυτός.

Το πρόγραμμά σας θα πρέπει να δουλεύει **για οποιοδήποτε αρχείο** topology.txt. Η μοναδική υπόθεση που μπορείτε να κάνετε (αν χρειαστεί) είναι για το μέγιστο αριθμό παιδιών κάθε κόμβου (πχ, 20, ανάλογα με το πώς δημιουργήσατε την τοπολογία σας).

ΠΡΟΓΡΑΜΜΑ 2 (Προθεσμία: 20/12/2021)

Σας ζητείται να υλοποιήσετε τη λειτουργία του LEACH (διαβάστε τη δημοσίευση και τις διαφάνειες για τις λεπτομέρειες). Πρέπει να φροντίσετε ο κώδικάς σας να έχει την εξής λειτουργικότητα:

- Το πρόγραμμα να τρέχει για 3000 sec.
- Ανά 150 sec να γίνεται εκλογή ηγετών (διάρκεια γύρου)
- Σε κάθε εκλογή, το 25% των κόμβων θα εκλέγεται ηγέτης, σύμφωνα με το LEACH. Συνεπώς σε διάστημα 600 sec θα έχουν εκλεγεί όλοι οι κόμβοι από 1 φορά ως ηγέτες.
- Σε αντίθεση με τη δημοσίευση, ο σταθμός βάσης (κόμβος 0) θα είναι ΠΑΝΤΑ ηγέτης, αφού δεν έχει περιορισμούς ενέργειας.
- Ένας κόμβος ο οποίος ΔΕΝ εκλεγεί ηγέτης σε μία διαδικασία εκλογής και ΔΕ λάβει κάποια διαφήμιση από άλλο ηγέτη (πχ, δεν είναι στην εμβέλεια κανενός), γίνεται ηγέτης του εαυτού του μόνο (δε στέλνει διαφήμιση). Μη μετρήσετε, σε αυτή την περίπτωση, ότι ο κόμβος εκλέχθηκε ηγέτης (συνεπώς, αν δεν είχε εκλεγεί ως ηγέτης πριν στο γύρο αυτό, θα πρέπει να είναι υποψήφιος στην επόμενη εκλογή).
- Ένας κόμβος που ΔΕΝ είναι ηγέτης και λάβει παραπάνω από 1 διαφημίσεις κάνει τυχαία επιλογή του ηγέτη του (ανάμεσα σε αυτούς που γνωρίζει από τις διαφημίσεις).
- Ανά 30 sec:
    - Στην πρώτη φάση οι κόμβοι θα στέλνουν τη μέτρησή τους στον ηγέτη τους. Αυτός θα υπολογίζει τη διασπορά τους και το μέσο όρο τους.
    - Στη δεύτερη φάση οι ηγέτες προωθούν **την κατάλληλη πληροφορία** προς το σταθμό βάσης. Για αυτή την προώθηση θα χρειαστείτε να έχετε φτιάξει ένα δέντρο, σύμφωνα με το Πρόγραμμα 1. Συνεπώς, πρώτα δημιουργείτε ένα δέντρο προώθησης αποτελεσμάτων (το οποίο ΔΕΝ αλλάζει σε διαφορετικές εκλογές), και το χρησιμοποιείτε μόνο για την προώθηση των αποτελεσμάτων από τους εκάστοτε ηγέτες προς το σταθμό βάσης.
    - Ο σταθμός βάσης τυπώνει τον τελικό μέσο όρο και την τελική διασπορά.
- Συνεπώς υπάρχουν 5 (=150/30) τέτοιες εποχές ανάμεσα σε κάθε εκλογή ηγέτη.
- ΔΕ σας ζητείται να υλοποιήσετε TDMA σε κάθε cluster. Συνεπώς μπορούν οι κόμβοι που στέλνουν δεδομένα σε ένα ηγέτη να το κάνουν στο ίδιο παράθυρο.

<u>ΤΙ ΘΑ ΠΑΡΑΔΩΣΕΤΕ</u> (σε 2 φάσεις παράδοσης)

1. Κώδικα **<u>με σχόλια</u>**
2. Μία αναφορά **<u>(1 σε κάθε φάση παράδοσης)</u>** που θα περιγράφει
    a. Τι διορθώσατε στο αρχείο που σας δόθηκε και γιατί. Βρήκατε κομμάτια που δεν εκτελούνται ποτέ και τα αφαιρέσατε; Αν ναι, ποια; Βρήκατε κομμάτια που δε λειτουργούσαν σύμφωνα με τις αρχές του TAG; Αν ναι, ποια;
    b. Σημαντικά κομμάτια κώδικα για κάθε ερώτημα και επεξήγηση.
    c. Εξήγηση για τα περιεχόμενα κάθε μηνύματος που στέλνετε – σε τι σας χρησιμεύει το κάθε πεδίο;
    d. <u>Παραδείγματα με πίνακες που θα έχουν συγκεκριμένο δέντρο, δεδομένα</u> κόμβων και το τι υπολόγισε στη ρίζα του δέντρου το πρόγραμμά σας.
    e. Ποιος φοιτητής υλοποίησε ποιο κομμάτι. Η δήλωση αυτή πρέπει να είναι απολύτως ακριβής. Σε περίπτωση αναληθών δηλώσεων θα υπάρχει βαθμολογική ποινή σε όλα τα μέλη της ομάδας.

<u>Επισημαίνεται ότι στη βαθμολογία σας θα μετρήσει ΣΗΜΑΝΤΙΚΑ και το αν έχετε ελαχιστοποιήσει τη μεταδιδόμενη πληροφορία (αριθμό μηνυμάτων και αριθμό bytes σε κάθε μήνυμα).</u> Αυτό πρέπει να γίνει στον κώδικα σας. Δεν αρκεί να τρέχει το πρόγραμμά σας, αλλά και να τρέχει σωστά και βέλτιστα. Το να πείτε στην προφορική εξέταση του πρότζεκτ σας «Ε, ναι, θα μπορούσα να μην το κάνω αυτό» δε μετράει – θα πρέπει να έχετε κάνει τις βελτιστοποιήσεις σας στον κώδικα.

Υπενθυμίζω ότι η αναφορά (σε κάθε στάδιο παράδοσης) βαθμολογείται και ελλιπείς ή ανύπαρκτες αναφορές θα οδηγήσουν σε απώλεια βαθμών, άσχετα με το πόσο καλά τρέχει ο κώδικας.

Βαθμολογία 1^{ου} παραδοτέου (βοηθητικό πρόγραμμα + Πρόγραμμα 1 + Αναφορά): 40%

Βαθμολογία 2^{ου} παραδοτέου (Πρόγραμμα 2 + Αναφορά): 60%. Προφανώς, σε αυτό το παραδοτέο πρέπει να συμπεριλάβετε και το βοηθητικό πρόγραμμα του 1^{ου} παραδοτέου.