ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Επεξεργασία Δεδομένων σε Δίκτυα Αισθητήρων

ΠΛΗ 511

Εργασία Εξαμήνου – Μέρος 1ο

«Προσομοίωση Δικτύου Αισθητήρων με αρχή λειτουργίας σύμφωνα με το TiNA»

Πέτρου Δημήτριος – 2018030070

Καθηγήτης: Α. Δεληγιαννάκης

Χανιά, Δεκέμβριος 2022

1. Εισαγωγή

Σκοπός της εργασίας ήταν η ανάπτυξη σε κώδικα μιας προσομοίωσης δικτύου αισθητήρων οι οποίοι μεταδίδουν μεταξύ τους εικονικές μετρήσεις. Η προσομοίωση είχε την επιλογή να εκτελέσει τυχαία μια εκ των δύο ή και τις δύο συναθροιστικές συναρτήσεις MAX και COUNT επί του συνόλου των κόμβων και των μετρήσεων τους. Η δομή του δικτύου ήταν δενδρική με φύλλα και μεταδόσεις μετρήσεων που πραγματοποιούνταν από τα παιδιά προς τους γονείς, ξεκινώντας από τα φύλλα του δέντρου. Η μετάδοση δεδομένων δεν γινόταν καθολικά για όλους τους κόμβους αλλά διαδοχικά για κάθε βάθος του δέντρου και κάθε κόμβο στο εκάστοτε βάθος. Όλες οι μεταδόσεις λάμβαναν χώρα μόνο αν τα δεδομένα ξεπερνούσαν ένα φράγμα (TCT) όπως ορίζει η σχεδιαστική αρχή TiNA. Το πρόγραμμα διαβάζει ενα αρχείο τοπολογίας κόμβων που παράγεται από ενα εξωτερικό εργαλείο όπως αυτό περιγράφεται παρακάτω. Εν συνεχεία προσομοιώνει την τοπολογία, παράγει τυχαίες μετρήσεις και υλοποιεί μεταδόσεις.

2. Πρόγραμμα παραγωγής αρχείου topology – Πρόγραμμα 1

Η σύνταξη του προγράμματος έγινε σε γλώσσα Python. Για την παραγωγή του αρχείο αρκεί στο terminal να εκτελεστεί η παρακάτω εντολη:

```
python genTopology.py D R
```

Όπου D είναι ο ακεραίος αριθμός που αντιστοιχεί στο πλευρικό μέγεθος του τετράγωνου πλέγματος που θα δημιουργηθεί. R είναι η εμβέλεια του κόμβου βάση της οποίας υπολογίζονται οι γείτονες του κόμβου. Ο κάθε κόμβος j βρίσκεται στην γραμμή j/D και στην στήλη j%D. Οι κόμβοι που παράγονται έχουν ID από $0 \rightarrow D^2 - 1$.

Ένας κόμβος j που βρίσκεται στην γραμμή r = j/D και στην στήλη c = j%D απέχει από έναν άλλον κόμβο (x,y) ευκλείδια απόσταση

$$\text{dist} = \sqrt{(x-r)^2+(y-c)^2}$$

Όσοι κόμβοι έχουν από τον εκάστοτε κόμβο **ευκλείδια απόσταση <= R** θεωρούνται γείτονες.

Το πρόγραμμα αποτελείται από δύο βασικές συναρτήσεις (εκτός της main):

`genTopologyFile(D,R)`: Δημιουργεί το πλέγμα των κόμβων και ύστερα το διατρέχει για κάθε έναν από αυτούς προκειμένου να βρει τους γειτονικούς κόμβους του καλώντας την συνάρτηση εύρεσης γειτόνων. Είναι επίσης υπεύθυνη για την σύνταξη του αρχείου τοπολογίας σύμφωνα με την λίστα των γειτόνων που θα λάβει για τον εκάστοτε κόμβο.

`findNeighbours(D,R,grid,j)`: Βρίσκει τους κόμβους από το grid που ικανοποιούν την συνθήκη της ευκλείδιας απόστασης από τον κόμβο j και τους θέτει ως γειτονικούς κόμβους. Επιστρέφει την λίστα των γειτόνων του κόμβου j προκειμένου να εγγραφούν στο αρχείο.

3. Πρόγραμμα προσομοίωσης – Πρόγραμμα 2

α. Αφαίρεση Κώδικα – Βελτιστοποίηση Μηνυμάτων Κονσόλας

Ο κώδικας που δόθηκε για την άσκηση περιείχε κομμάτια τα οποία δεν συνάδουν με τα πρότυπα του TAG. Αρχικά αφαιρέθηκαν όλα τα τμήματα τα αφορούσαν στην επικοινωνία μέσω Serial Port και τα κομμάτια που αφορούσαν στα LEDs. Τα κομμάτια αυτά δεν περιορίζονται από τα πρότυπα του TAG ωστόσο στην παρούσα άσκηση δεν χρησίμευαν.

Στην συνέχεια κρίθηκε απαραίτητο να αφαιρεθούν όλα τα tasks, οι συναρτήσεις και μεταβλητές που αφορούσαν στην διαχείριση περιπτώσεων κατά τις οποίες υπήρχε απώλεια μετάδοσης μηνυμάτων ή χαμένων μηνυμάτων. Αφαιρέθηκαν έτσι όλα τα τμήματα που αφορούσαν σε Lost Tasks και Busy flags.

Γενικότερα αφαιρέθηκαν όλοι οι ορισμοί IFDEF – ENDIF καθώς η εκτύπωση μηνυμάτων debug γίνεται μέσω της συνάρτησης dbg(…) σε νέα συγκεκριμένα κανάλια για τα οποία περιγράφεται στον παρακάτω πίνακα τι εξυπηρετεί το καθένα:

TinaSetup	Εκτύπωση επιλεγμένου aggregate και TCT
TinaMeasurements	Εκτύπωση μετρήσεων TiNA
RoutingRes	Εκτύπωση δέντρου μετά το routing
Routing	Εκτύπωση μηνυμάτων διαδικασίας routing
Aggregation	Εκτύπωση βημάτων συνάθροισης

Όσον αφορά στην υλοποιήση του Routing ακολουθώντας το πρότυπο TAG τα παιδιά δεν χρειάζεται να ειδοποιούν τον πατέρα μετά από επιτυχή σύνδεση μαζί του. Για τον λόγο αυτό αφαιρέθηκαν όλα τα τμήματα που αφορούσαν σε λειτουργίες Notify. Επίσης καθώς η προσομοίωση διατρέχει το αρχείο τοπολογίας σύνδεει έναν πατέρα με ένα παιδί σύμφωνα με το πρώτο occurrence αυτών. Δεν υπάρχει ανάγκη για εύρεση καλύτερου πατέρα, έτσι από το task **`ReceiveRoutingTask()`** αφαιρέθηκε ο κώδικας που αποσκοπεί στην εύρεση καλύτερου πατέρα. Στην υλοποίηση αυτή όταν ένας κόμβος λαμβάνει RoutingMsg από κάποιον κόμβο τότε θεωρεί πως αυτός είναι ο μοναδικός πατέρας του για όλο το simulation.

β. Δομικές Αλλαγές - Προσθήκες

Στο μοντέλο της προσομοίωσης έγιναν δομικές αλλαγές στα υπάρχοντα **`nx_structs`** όσον αφορά στα πεδία του εκάστοτε μηνύματος, αλλά και προσθήκες νέων:

RoutingMsg:

- Αφαιρέθηκε το πεδίο **senderID** καθώς η συγκεκριμένη πληροφορία εγκιβωτίζεται μέσα στο μήνυμα προτού σταλθεί και κατά την λήψη του η πρόσβαση είναι δύνατη μέσω:

```
AMPacket.source(&tmpMsg)
```

- Ενσωματώθηκε το πεδίο **executionParameters** το οποίο περιλαμβάνει πληροφορία για την επιλογή του TCT και της aggregate συνάρτησης που θα ισχύσουν για το εκάστοτε simulation. Παρακάτω στην αντίστοιχη ενότητα αναλύεται η κωδικοποίηση της πληροφορίας στο πεδίο.

BufferMsg:

Χρησιμοποιείται για εσωτερική (εντός του κόμβου) μετάδοση πληροφορίας τυχαίας μέτρησης για την εκάστοτε εποχή. Εισάγεται στην τοπική ουρά αποστολής αφού υπολογιστεί η μέτρηση κατά το fire του DataDistrTimer() και εξάγεται από το sendDistributionTask() για να ληφθεί η μέτρηση που παράχθηκε από τον ίδιο τον κόμβο για την εκάστοτε εποχή. Δεν μεταδίδεται εκτός του κόμβου

DistributionMsgSingle:

Χρησιμοποιείται για την μετάδοση πληροφόριας κατά την εκτέλεση μιας και μόνο aggregate συνάρτησης. Περιέχει ένα πεδίο το **`data:nx_uint8_t`** στο οποίο αποθηκεύεται το αποτέλεσμα της συναθροιστικής συνάρτησης προς μτεάδοση.

DistributionMsgSemi:

Χρησιμοποιείται για την μετάδοση πληροφόριας κατά την εκτέλεση δύο aggregate συνάρτησεων αλλά με αποκοπή του ενός εκ των δύο αποτελεσμάτων λόγω των περιορισμών του TiNA. Περιέχει δύο πεδία:

`flag : nx_uint8_t`	Λαμβάνει τιμές MAX(0) ή COUNT(1) και χρησιμεύει για την ανίχνευση του είδους της συνάρτησης που μεταδίδεται κατα την αποκοπή της αποστολής ενός εκ των δυό
`data : nx_uint8_t`	Φέρει την τιμή της συναθροιστικής συνάρτησης

DistributionMsgFull:

Χρησιμοποιείται για την μετάδοση πληροφορίας κατά την εκτέλεση δύο aggregate συναρτήσεων όταν δεν υπάρχει αποκοπή λόγω περιορισμών του TiNA. Περιέχει δύο πεδία:

`max : nx_uint16_t`	Φέρει την τιμή της MAX(). 16 bit για να διαχωρίζεται από το DistributionMsgSemi
`count : nx_uint8_t`	Φέρει την τιμή της COUNT()

Σχετικά με τις δομές οι οποίες υλοποιούν τις μεταδόσεις αλλά και για σκοπούς που τις εξυπηρετούσαν υλοποιήθηκαν τα εξής με γνώμονα την διαδικασία αποστολής μηνυμάτων του Routing. Για τα components που το απαιτούσαν έγινε η κατάλληλη «σύνδεση» μέσω του **SRTreeAppC.nc.**

Timers

RoutingTimer	Χρονομετρά την διαδικασία του Routing και την παύει κατά την στιγμή που θα γίνει fired(). Κατά το fired() θα εκκινήσει για κάθε κόμβο τον Distribution με κατάλληλη περιοδικότητα
RoundTimer	Χρονομετρά περιοδικά για περίοδο TIMER_PERIOD_MILLI (=30720, epoch time) ώστε να αλλάζει ο γύρος κατάλληλα.
DataDistrTimer	Χρονομέτρα περιοδικά για κάθε κόμβο ώστε να υπολογίζει και να μεταδίδει πληροφορία στην κατάλληλη στιγμή

**Για τους Timers αναφέρεται αναλυτικά ο τρόπος χρονισμού τους στην επόμενη παράγραφο*

Senders / Receivers / Queues

DistributionPacket	Χρησιμοποιείται για κατασκευή μηνυμάτων
DistributionAMPacket	Χρησιμοποιείται για κατασκευή μηνυμάτων
DistributionAMSend	Χρησιμοποιείται για αποστολή μηνυμάτων
DistributionReceive	Χρησιμοποιείται για λήψη μηνυμάτων
DistributionSendQueue	Χρησιμοποιείται για σειριακή αποθήκευση εξερχόμενων μηνυμάτων
DistributionReceiveQueue	Χρησιμοποιείται για σειριακή αποθήκευση εισερχόμενων μηνυμάτων

Για την παραγωγή τυχαίων μετρήσεων σε κάθε εποχή χρησιμοποιήθηκε η βιβλιοθήκη /dev/urandom με seed το ID του εκάστοτε κόμβου.

Δομές αποθήκευσης πληροφορίας

childNode : struct	Δομή που χρησιμοποιείται για την σύνθεση λίστα πληροφορίων παιδιών εντός ενός κόμβου
children[] : childNode	Χρησιμοποιείται για την αποθήκευση μετρήσεων των παιδιών ενός κόμβου σε περίπτωση απώλειας μηνυμάτων. Εξυπηρετεί η ύπαρξη πληροφορίας αν σε επόμενη εποχή υπάρξει απώλεια.

γ. Χρονισμός – Αλλάγες στους υπάρχοντες timers

Ο χρονισμός και η σωστή αρχικοποίηση μέτρησης των timers ήταν από τα σημαντικότερα τμήματα του simulation. Όπως αναφέρθηκε και παραπάνω υλοποιήθηκαν νέοι timers των οποίων η ύπαρξη διευκολύνει εξαιρετικά τον χρονισμό του timer της αποστολής μετρήσεων (DataDistrTimer).

RoutingTimer:

Διαρκεί 5 δευτερόλεπτα (6400ms/8bit) και επιτρέπει στο δίκτυο να ολοκληρώσει το Routing σε αυτό το χρόνο. Εκκινείται με startOneShot (δεν είναι περιοδικός) μέσα από την αρχικοποίηση του RadioControl.

RoundTimer:

Εκτελείται περιοδικά για περίοδο TIMER_PERIOD_MILLI και κατά το fired() αλλάζει την εποχή και εκτυπώνει το κατάλληλο μήνυμα στην κονσόλα.

Η εκκίνηση των δύο παραπάνω timers γίνεται μέσα από το event RadioControl.startDone():

```
236     event void RadioControl.startDone(error_t err)
237     {
238         if (err == SUCCESS)
239         {
240             int i;
241             for(i=0; i<64;i++){
242                 children[i].nodeID = 0;
243                 children[i].maxVal = 0;
244                 children[i].countVal = 0;
245             }
246 
247             // Allow 5 seconds for routing to be completed
248             call RoutingTimer.startOneShot(6400);
249             call RoundTimer.startPeriodicAt(0,TIMER_PERIOD_MILLI);
```

DataDistrTimer:

Ο χρονισμός του συγκεκριμένου timer ήταν ίσως ο πιο σύνθετος μεταξύ άλλων. Ο κάθε κόμβος θα πρέπει οπωσδήποτε να ανανεώσει τη μέτρηση του και ενδεχομένως να την στείλει σε κάθε εποχή. Για να γίνει αυτό χωρίς να υπάρχουν συγκρούσεις & απώλειες μηνυμάτων θα πρέπει να ληφθούν υπόψη δύο παράγοντες για την εκκίνηση του timer για κάθε κόμβο: Το βάθος στο οποίο βρίσκεται και το ID του.

Ο διαχωρισμός γίνεται σύμφωνα με τους δύο παράγοντες που αναφέρθηκαν παραπάνω:

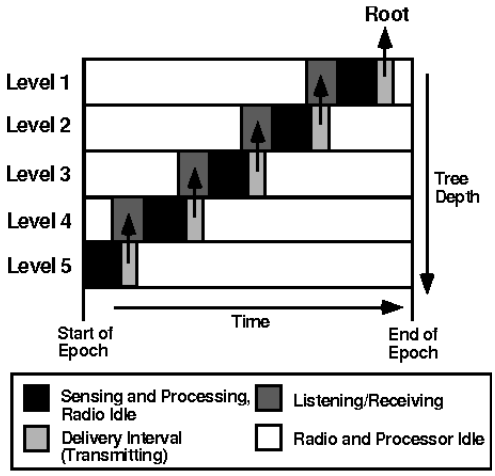
- **curDepth:**
  Σύμφωνα με την παρακάτω ανάθεση όλοι οι κόμβοι που ανήκουν στο ίδιο βάθος θα μεταδώσουν την ίδια χρονική στιγμή για παράθυρο 128ms (TIMER_FAST_PERIOD). Κάτι τέτοιο ωστόσο θα επέφερε συγκρούσεις και απώλειες καθώς κόμβοι του ίδιου βάθους θα προσπαθούσαν να μεταδώσουν την ίδια χρονική στιγμή.

```
call DataDistrTimer.startPeriodicAt( -((curdepth+1)*TIMER_FAST_PERIOD), TIMER_PERIOD_MILLI);
```

- **TOS_NODE_ID:**
  Για να μειωθεί η πιθανότητα σύγκρουσης είναι εφικτό να προστεθεί ένα offset στον χρόνο εκκίνησης του timer κατά μερικά ms σύμφωνα με το TOS_NODE_ID:

```
call DataDistrTimer.startPeriodicAt( -((curdepth+1)*TIMER_FAST_PERIOD)+(TOS_NODE_ID*4), TIMER_PERIOD_MILLI);
```

Ο πολλαπλασιασμός x4 προέκυψε, μετά από δοκιμές, ως το ελάχιστο κάτω φράγμα προτού συμβούν συγκρούσεις που θα οδηγήσουν σε απώλειες μηνυμάτων. Στο διάγραμμα που ακολουθεί φαίνεται η κατανομή του χρόνου της εποχής ανά επιπεδο σύμφωνα με το TAG:



Σημειώνεται πως κρίθηκε απαραίτητο η προσομοιώση να τρέξει για 1200 + 10 δευτερόλεπτα συνολικά προκειμένου να αντισταθμιστεί ο χρόνος που χρειάζεται για το boot. Στο mySimulation.py τροποιήθηκε το εξής:

```
9    SIM_END_TIME= 1210 * t.ticksPerSecond()
```

Σε διαφορετική περίπτωση ο χρόνος δεν επαρκεί για την επεξεργασία και αποστολή μετρήσεων στην τελευταία εποχή (40^n). Μια άλλη λυση στο συγκεκριμένο ζήτημα θα ήταν να εκκινηθούν οι timers 10 δευτερόλεπτα νωρίτερα.

δ. Μετρήσεις - Κωδικοποίηση πληροφορίας μηνυμάτων

Στην αρχή κάθε εποχής ο κάθε κόμβος πρέπει να παράγει μια τυχαία μέτρηση προκείμενου να διαφοροποιείται από την προηγούμενη και να παράγεται μια αντιπροσωπευτική προσομοίωση του δικτύου. Αυτό επιτύγχανεται, όταν γίνεται περιοδικά fired() ο timer DataDistrTimer, με τον τρόπο που αποτυπώνεται στον τμήμα κώδικα παρακάτω:

```
375         //Initially the last measurement is set to 0 so we generate the entry point
376         //for random measurement generation. When the first measurement is generated
377         //then every next measurement will be randomly selected withing a range of +-10%.
378
379         if(lastMeasurement==0){
380             currMeasurement = call Random.rand16()%80+1;
381         }else{
382             uint8_t upper = (int)lastMeasurement + 0.1*lastMeasurement;
383             uint8_t lower = (int)lastMeasurement - 0.1*lastMeasurement;
384
385             currMeasurement = call Random.rand16()%(upper-lower + 1) + lower;
386         }
387
388         //Update lastMeasurement var used to calculate the new value upon +-10% deviation rule.
389         lastMeasurement = currMeasurement;
```

Αρχικά επιλέγεται μια τιμή τυχαία μετάξυ του 1 και του 80 και στη συνέχεια κάθε επόμενη μέτρηση απέχει ±10% από την προηγούμενη. Η απόκλιση των μετρήσεων είναι τυχαία προκειμένου να υπάρχει διαφοροποίηση και κατα περίπτωση υπέρβαση του TCT ώστε να εξακριβωθεί η ορθή λειτουργία του TiNA.

Κατά την διαδικασία του Boot η ρίζα (κόμβος 0) επιλέγει τυχαία να εκτέλεσει μια aggregate συνάρτηση και καθορίζει επίσης τυχαία του ορίου ανοχής (TCT). Αυτό γίνεται πριν το Routing προκειμένου να μην καταναλωθεί πολύτιμος χρόνος. Κατα το Routing ωστόσο γίνεται η αποστολή της επιλογής της ρίζας ως προς τις δύο αυτές παραμέτρους προς τους υπόλοιπους κόμβους. Η πληροφορία ενσωματώνεται στο RoutingMsg χωρίς ωστόσο να καταναλώνονται πολύτιμοι πόροι. Αμφότερες οι επιλογές της ρίζας εισάγονται κωδικοποιήμένα σε ένα πεδίο **`executionParemeters: nx_uint8_t`** στο RoutingMsg.

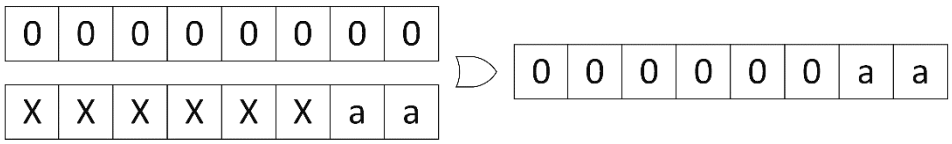
Encoding

Η κωδικοποίηση της πληροφορίας γίνεται από την συνάρτηση encodeParameters():

```
137     /*
138         Encodes execution parameters in order to minimize the transmitting message
139         and reduce power consumption
140     */
141     uint8_t encodeParameters(uint8_t aggr, uint8_t tctArg){
142
143         uint8_t encoded = 0;
144         encoded = encoded | aggr;
145         encoded = encoded<<2;
146         encoded = encoded | tctArg;
147         return encoded;
148
149     }
```

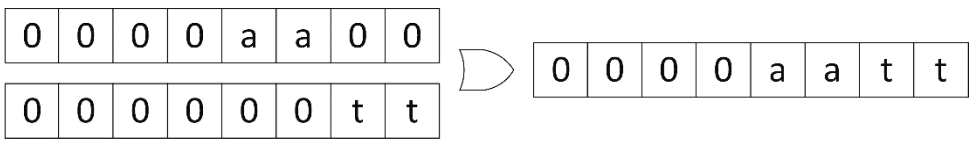
Η λογική της συνάρτησης αποτυπώνεται εν συντομία στα παρακάτω βήματα:

1. Ξεκινώντας με έναν «κενό» αριθμό 8bit “0000-0000” εκτελείται πράξη OR μεταξύ αυτού και της παραμέτρου επιλογής συναθροιστικής aggr. Ουσιαστικά εκτελείται η πράξη:



2. Στη συνέχεια προκειμένου να αποθηκευτεί στον ίδιο αριθμό και το flag για την τιμή του TCT το οποίο παίρνει μέγιστη τιμή 3 (άρα χρειάζονται max 2 bit για την αναπαράσταση του) όλος ο αριθμός γίνεται shift left κατά 2 bit με zero fill.

3. Στη συνέχεια με τον πλέον shifted αριθμό εκτελείται πράξη OR μεταξύ αυτού και της παραμέτρου επιλογής του TCT. Πλέον σε έναν αριθμό είναι ενσωματωμένες και οι δυο παράμετροι λειτουργίας του simulation και το μήνυμα στο οποίο θα συμμετάσχει θα είναι μικρότερο σε μήκος.



Για την αποκωδικοποίηση χρησιμοποιείται αντίστοιχη διαδικασία κατά την λήψη του μηνύματος από τον δέκτη.

ε. Υλοποίηση προτύπου TiNA

Το πρότυπο TiNA το οποίο αποσκοπεί στη βελτιστοποίηση του δικτύου και την εξοικονόμηση ενέργειας μέσω του περιορισμού των μηνυμάτων που αποστέλλονται από έναν κόμβο προς τον γονέα του. Το TiNA ορίζει πως μόνο αν μια νέα μέτρηση απέχει από την προηγούμενη κατά TCT% είναι άξια για να μεταδοθεί στο δίκτυο.

- Εκτέλεση μιας συναθροιστικής συνάρτησης

  Η λογική του κώδικα αποτυπώνεται στο παρακάτω snippet ψευδοκώδικα:

```
if(epoch==1 | newMeasurement > tct*lastMeasurement + lastMeasurement){
      send Message;
}else{
      do-nothing;
}
```

- Εκτέλεση δύο συναθροιστικών συναρτήσεων

  Η λογική του κώδικα αποτυπώνεται στο παρακάτω snippet ψευδοκώδικα:

```
if(epoch==1 | newMeasMAX > tct*lastMeasMAX + lastMeasMAX
           && newMeasCOUNT > tct*lastMeasCOUNT + lastMeasCOUNT){
      send Message for both;

}else if( newMeasMAX > tct*lastMeasMAX + lastMeasMAX
           && newMeasCOUNT < tct*lastMeasCOUNT + lastMeasCOUNT
      send Message for MAX;
}else if( newMeasMAX < tct*lastMeasMAX + lastMeasMAX
           && newMeasCOUNT > tct*lastMeasCOUNT + lastMeasCOUNT
      send Message for COUNT;
}else{
      do-nothing;
}
```

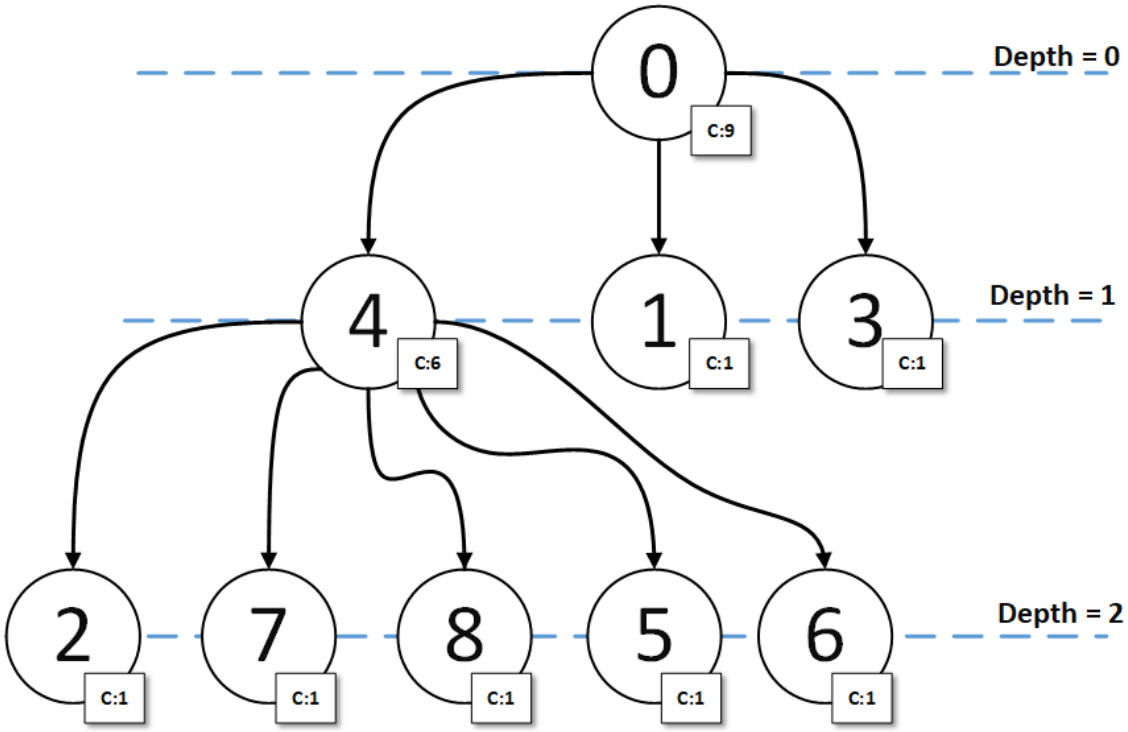
στ. Παραδείγματα εκτέλεσης

- Ενδεικτικό παράδειγμα εκτέλεσης COUNT() σε grid 9x9 με range 1.5

```
0:0:10.125000010 DEBUG (0): ##############################################
0:0:10.125000010 DEBUG (0):                    ROUND   1
0:0:10.125000010 DEBUG (0): ##############################################
0:0:10.135009737 DEBUG (4): New parent for NodeID= 4 : curdepth= 1 , parentID= 0
0:0:10.135009737 DEBUG (1): New parent for NodeID= 1 : curdepth= 1 , parentID= 0
0:0:10.135009737 DEBUG (3): New parent for NodeID= 3 : curdepth= 1 , parentID= 0
0:0:10.267288194 DEBUG (2): New parent for NodeID= 2 : curdepth= 2 , parentID= 4
0:0:10.267288194 DEBUG (7): New parent for NodeID= 7 : curdepth= 2 , parentID= 4
0:0:10.267288194 DEBUG (8): New parent for NodeID= 8 : curdepth= 2 , parentID= 4
0:0:10.267288194 DEBUG (5): New parent for NodeID= 5 : curdepth= 2 , parentID= 4
0:0:10.267288194 DEBUG (6): New parent for NodeID= 6 : curdepth= 2 , parentID= 4
0:0:39.632812510 DEBUG (2): Node [2] at depth -2- measuring now: 55
0:0:39.632812520 DEBUG (2): Node [2] ⟶ Over-TCT measurements: Last= 0| New= 1 ⟹Transmitting ...
0:0:39.636657722 DEBUG (4): ID:[4] received from child [2] COUNT data = 1
0:0:39.644531260 DEBUG (5): Node [5] at depth -2- measuring now: 3
0:0:39.644531270 DEBUG (5): Node [5] ⟶ Over-TCT measurements: Last= 0| New= 1 ⟹Transmitting ...
0:0:39.648437510 DEBUG (6): Node [6] at depth -2- measuring now: 38
0:0:39.648437520 DEBUG (6): Node [6] ⟶ Over-TCT measurements: Last= 0| New= 1 ⟹Transmitting ...
0:0:39.652343760 DEBUG (7): Node [7] at depth -2- measuring now: 51
0:0:39.652343770 DEBUG (7): Node [7] ⟶ Over-TCT measurements: Last= 0| New= 1 ⟹Transmitting ...
0:0:39.653381326 DEBUG (4): ID:[4] received from child [5] COUNT data = 1
0:0:39.655380239 DEBUG (4): ID:[4] received from child [6] COUNT data = 1
0:0:39.656250010 DEBUG (8): Node [8] at depth -2- measuring now: 58
0:0:39.656250020 DEBUG (8): Node [8] ⟶ Over-TCT measurements: Last= 0| New= 1 ⟹Transmitting ...
0:0:39.659576404 DEBUG (4): ID:[4] received from child [7] COUNT data = 1
0:0:39.666137668 DEBUG (4): ID:[4] received from child [8] COUNT data = 1
0:0:39.753906260 DEBUG (1): Node [1] at depth -1- measuring now: 56
0:0:39.753906270 DEBUG (1): Node [1] ⟶ Over-TCT measurements: Last= 0| New= 1 ⟹Transmitting ...
0:0:39.761718760 DEBUG (3): Node [3] at depth -1- measuring now: 55
0:0:39.761718770 DEBUG (3): Node [3] ⟶ Over-TCT measurements: Last= 0| New= 1 ⟹Transmitting ...
0:0:39.764007539 DEBUG (0): ID:[0] received from child [1] COUNT data = 1
0:0:39.765625010 DEBUG (4): Node [4] at depth -1- measuring now: 3
0:0:39.765625020 DEBUG (4): Node [4] ⟶ Over-TCT measurements: Last= 0| New= 6 ⟹Transmitting ...
0:0:39.771057104 DEBUG (0): ID:[0] received from child [3] COUNT data = 1
0:0:39.773605330 DEBUG (0): ID:[0] received from child [4] COUNT data = 6
0:0:39.875000010 DEBUG (0): Node [0] at depth -0- measuring now: 49
0:0:39.875000020 DEBUG (0): Node [0] ⟶ For this round COUNT()= 9
0:0:40.000000010 DEBUG (0): ##############################################
0:0:40.000000010 DEBUG (0):                    ROUND   2
0:0:40.000000010 DEBUG (0): ##############################################
0:1:9.632812510 DEBUG (2): Node [2] at depth -2- measuring now: 56
0:1:9.632812520 DEBUG (2): Node [2] ⟶ Under-TCT measurements: Last= 1| New= 1
0:1:9.644531260 DEBUG (5): Node [5] at depth -2- measuring now: 2
0:1:9.644531270 DEBUG (5): Node [5] ⟶ Under-TCT measurements: Last= 1| New= 1
0:1:9.648437510 DEBUG (6): Node [6] at depth -2- measuring now: 35
0:1:9.648437520 DEBUG (6): Node [6] ⟶ Under-TCT measurements: Last= 1| New= 1
0:1:9.652343760 DEBUG (7): Node [7] at depth -2- measuring now: 45
0:1:9.652343770 DEBUG (7): Node [7] ⟶ Under-TCT measurements: Last= 1| New= 1
0:1:9.656250010 DEBUG (8): Node [8] at depth -2- measuring now: 63
0:1:9.656250020 DEBUG (8): Node [8] ⟶ Under-TCT measurements: Last= 1| New= 1
0:1:9.753906260 DEBUG (1): Node [1] at depth -1- measuring now: 51
0:1:9.753906270 DEBUG (1): Node [1] ⟶ Under-TCT measurements: Last= 1| New= 1
```

Εφόσον το COUNT δεν αλλάζει, παρατηρείται ότι στην δεύτερη επόχη δεν υπάρχει μετάδοση άρα το δέντρο λειτουργεί σύμφωνα με το TiNA.

Δέντρο για την παραπάνω προσομοίωση:

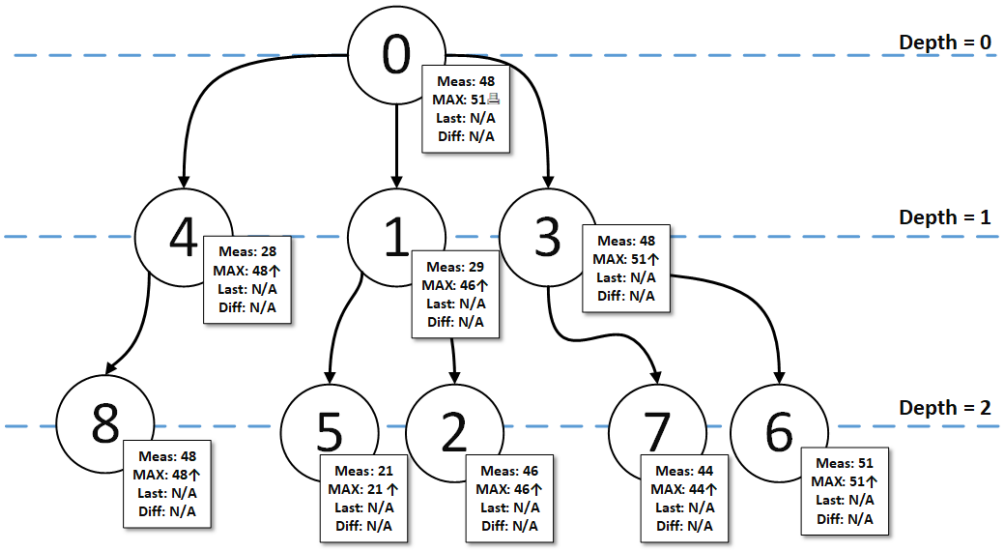


- Ενδεικτικό παράδειγμα εκτέλεσης MAX() σε grid 9x9 με range 1.5 και TCT = 10%

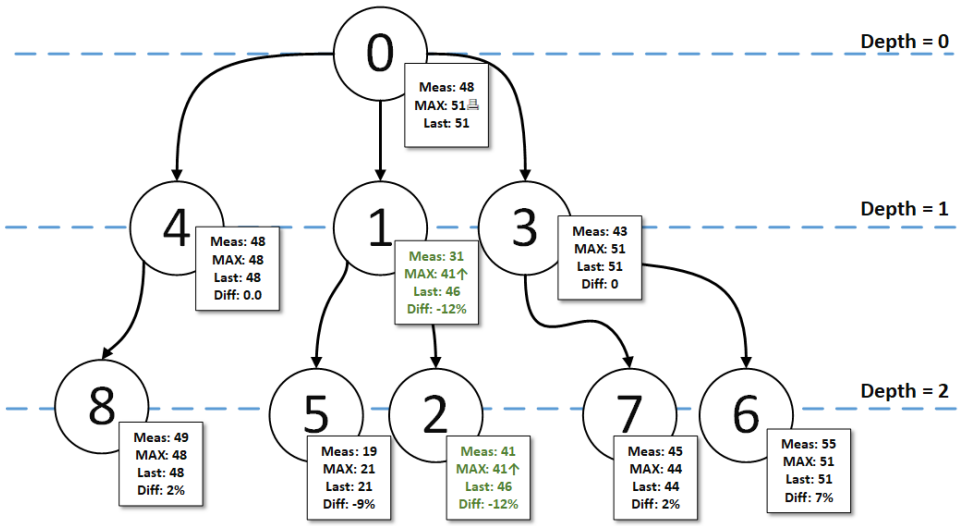
```
0:0:10.125000010 DEBUG (0): ##########################################
0:0:10.125000010 DEBUG (0):                    ROUND   1
0:0:10.125000010 DEBUG (0): ##########################################
0:0:10.135192841 DEBUG (4): New parent for NodeID= 4 : curdepth= 1 , parentID= 0
0:0:10.135192841 DEBUG (1): New parent for NodeID= 1 : curdepth= 1 , parentID= 0
0:0:10.135192841 DEBUG (3): New parent for NodeID= 3 : curdepth= 1 , parentID= 0
0:0:10.267456039 DEBUG (5): New parent for NodeID= 5 : curdepth= 2 , parentID= 1
0:0:10.267456039 DEBUG (2): New parent for NodeID= 2 : curdepth= 2 , parentID= 1
0:0:10.269348118 DEBUG (7): New parent for NodeID= 7 : curdepth= 2 , parentID= 3
0:0:10.269348118 DEBUG (6): New parent for NodeID= 6 : curdepth= 2 , parentID= 3
0:0:10.270385710 DEBUG (8): New parent for NodeID= 8 : curdepth= 2 , parentID= 4
0:0:39.632812510 DEBUG (2): Node [2] at depth -2- measuring now: 46
0:0:39.632812520 DEBUG (2): Node [2] ⟶ Over-TCT measurements: Last= 0| New= 46 ⟹Transmitting ...
0:0:39.637115483 DEBUG (1): ID:[1] received from child [2] MAX data = 46
0:0:39.644531260 DEBUG (5): Node [5] at depth -2- measuring now: 21
0:0:39.644531270 DEBUG (5): Node [5] ⟶ Over-TCT measurements: Last= 0| New= 21 ⟹Transmitting ...
0:0:39.648437510 DEBUG (6): Node [6] at depth -2- measuring now: 51
0:0:39.648437520 DEBUG (6): Node [6] ⟶ Over-TCT measurements: Last= 0| New= 51 ⟹Transmitting ...
0:0:39.651367178 DEBUG (1): ID:[1] received from child [5] MAX data = 21
0:0:39.652343760 DEBUG (7): Node [7] at depth -2- measuring now: 44
0:0:39.652343770 DEBUG (7): Node [7] ⟶ Over-TCT measurements: Last= 0| New= 44 ⟹Transmitting ...
0:0:39.652984623 DEBUG (3): ID:[3] received from child [6] MAX data = 51
0:0:39.656250010 DEBUG (8): Node [8] at depth -2- measuring now: 48
0:0:39.656250020 DEBUG (8): Node [8] ⟶ Over-TCT measurements: Last= 0| New= 48 ⟹Transmitting ...
0:0:39.659179701 DEBUG (4): ID:[4] received from child [8] MAX data = 48
0:0:39.661819433 DEBUG (3): ID:[3] received from child [7] MAX data = 44
0:0:39.753906260 DEBUG (1): Node [1] at depth -1- measuring now: 29
0:0:39.753906270 DEBUG (1): Node [1] ⟶ Over-TCT measurements: Last= 0| New= 46 ⟹Transmitting ...
0:0:39.761718760 DEBUG (3): Node [3] at depth -1- measuring now: 48
0:0:39.761718770 DEBUG (3): Node [3] ⟶ Over-TCT measurements: Last= 0| New= 51 ⟹Transmitting ...
0:0:39.762435893 DEBUG (0): ID:[0] received from child [1] MAX data = 46
0:0:39.765625010 DEBUG (4): Node [4] at depth -1- measuring now: 28
0:0:39.765625020 DEBUG (4): Node [4] ⟶ Over-TCT measurements: Last= 0| New= 48 ⟹Transmitting ...
0:0:39.771392819 DEBUG (0): ID:[0] received from child [4] MAX data = 48
0:0:39.773879964 DEBUG (0): ID:[0] received from child [3] MAX data = 51
0:0:39.875000010 DEBUG (0): Node [0] at depth -0- measuring now: 48
0:0:39.875000020 DEBUG (0): Node [0] ⟶ For this round MAX()= 51
0:0:40.000000010 DEBUG (0): ##########################################
0:0:40.000000010 DEBUG (0):                    ROUND   2
0:0:40.000000010 DEBUG (0): ##########################################
0:1:9.632812510 DEBUG (2): Node [2] at depth -2- measuring now: 41
0:1:9.632812520 DEBUG (2): Node [2] ⟶ Over-TCT measurements: Last= 46| New= 41 ⟹Transmitting ...
0:1:9.641372660 DEBUG (1): ID:[1] received from child [2] MAX data = 41
0:1:9.644531260 DEBUG (5): Node [5] at depth -2- measuring now: 19
0:1:9.644531270 DEBUG (5): Node [5] ⟶ Under-TCT measurements: Last= 21| New= 19
0:1:9.648437510 DEBUG (6): Node [6] at depth -2- measuring now: 55
0:1:9.648437520 DEBUG (6): Node [6] ⟶ Under-TCT measurements: Last= 51| New= 55
0:1:9.652343760 DEBUG (7): Node [7] at depth -2- measuring now: 45
0:1:9.652343770 DEBUG (7): Node [7] ⟶ Under-TCT measurements: Last= 44| New= 45
0:1:9.656250010 DEBUG (8): Node [8] at depth -2- measuring now: 49
0:1:9.656250020 DEBUG (8): Node [8] ⟶ Under-TCT measurements: Last= 48| New= 49
0:1:9.753906260 DEBUG (1): Node [1] at depth -1- measuring now: 31
0:1:9.753906270 DEBUG (1): Node [1] ⟶ Over-TCT measurements: Last= 46| New= 41 ⟹Transmitting ...
0:1:9.757064831 DEBUG (0): ID:[0] received from child [1] MAX data = 41
0:1:9.761718760 DEBUG (3): Node [3] at depth -1- measuring now: 43
0:1:9.761718770 DEBUG (3): Node [3] ⟶ Under-TCT measurements: Last= 51| New= 51
0:1:9.765625010 DEBUG (4): Node [4] at depth -1- measuring now: 25
0:1:9.765625020 DEBUG (4): Node [4] ⟶ Under-TCT measurements: Last= 48| New= 48
0:1:9.875000010 DEBUG (0): Node [0] at depth -0- measuring now: 48
0:1:9.875000020 DEBUG (0): Node [0] ⟶ For this round MAX()= 51
```

Παρατηρούμε ότι στην παραπάνω προσομοιώση γίνονται μεταδόσεις στην πρώτη εποχή ενώ στην δεύτερη μεταδίδονται μόνο οι μετρήσεις που υπερβαίνουν το TCT σε σχέση με τις προηγούμενες

Round 1:



Round 2:



Επισημαίνεται πως το πρόγραμμα 2 δεν δίνει σωστή μέτρηση COUNT όταν εκτελούνται και οι δύο συναθροιστικές συναρτήσεις ταυτόχρονα. Θα επιλυθεί το ζήτημα στο δεύτερο μέρος καθώς δεν υπήρχε αρκετός χρόνος για αποσφαλμάτωση του συγκεκριμένου