**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 ΜΑΧ 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 🡪 D2 – 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

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. Tα 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. Aφαιρέθηκαν 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** | Eκτύπωση 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:**

* Aφαιρέθηκε 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).

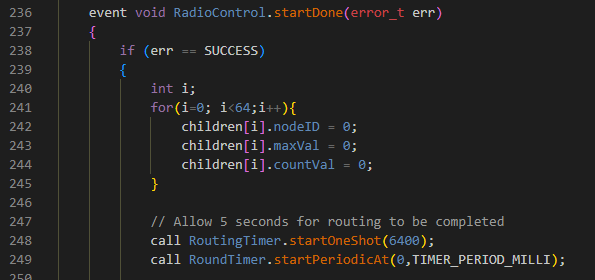
**RoutingTimer:**

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:**

Eκτελείται 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():



**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.

O 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.

* **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:

O 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:

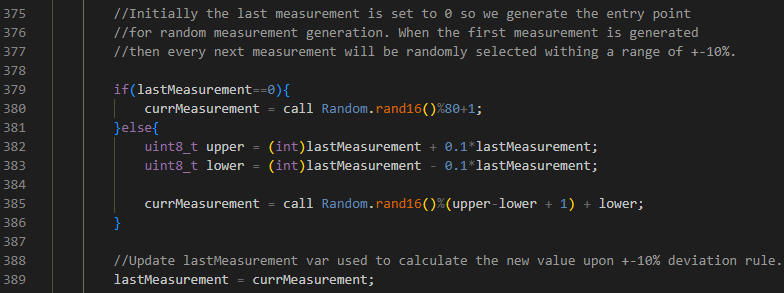
Diagram

Description automatically generated

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:

Σε διαφορετική περίπτωση ο χρόνος δεν επαρκεί για την επεξεργασία και αποστολή μετρήσεων στην τελευταία εποχή (40η). 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:

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():

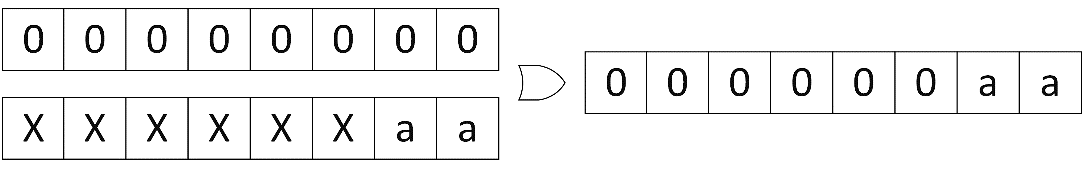
Text

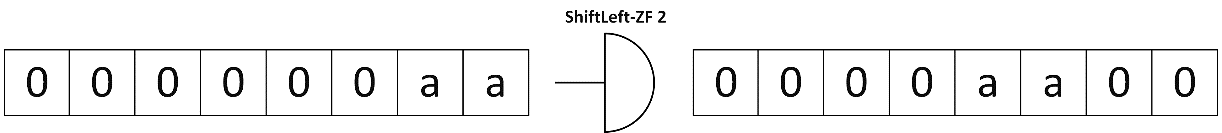
Description automatically generated

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:



1. 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.
2. A picture containing text, clock

   Description automatically generatedΣτη 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**

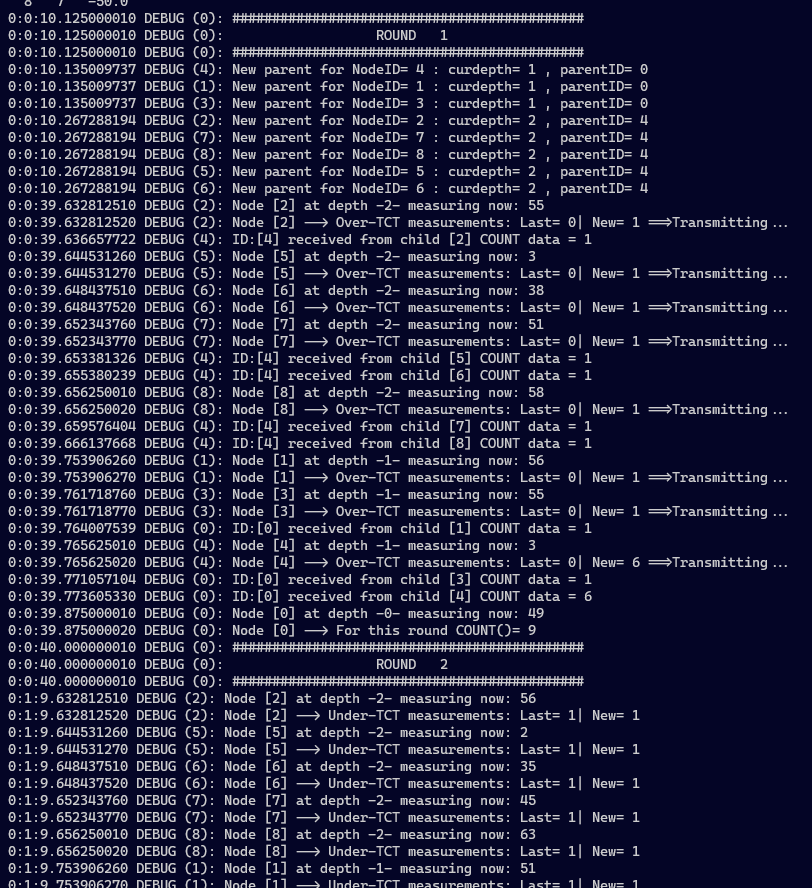
**send Message for COUNT;**

**}else{**

**do-nothing;**

**}**

στ. Παραδείγματα εκτέλεσης

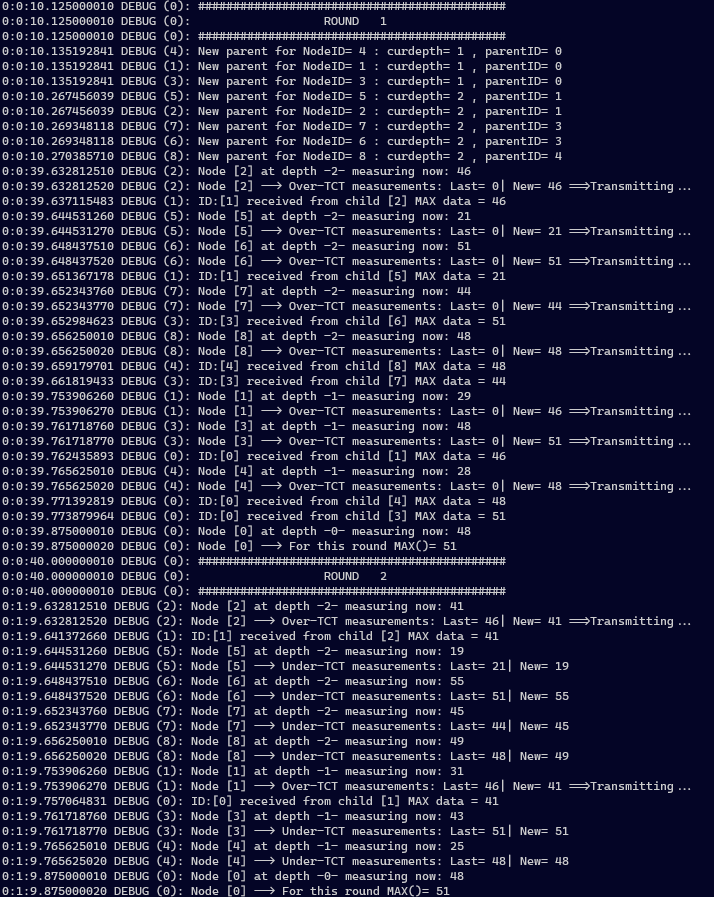
* Ενδεικτικό παράδειγμα εκτέλεσης COUNT() σε grid 9x9 με range 1.5

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:

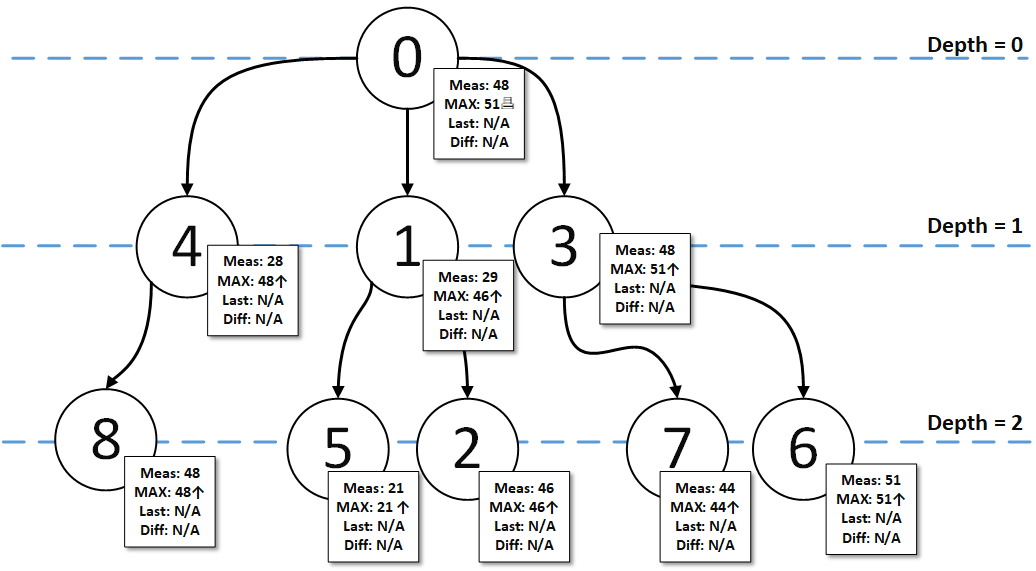
Diagram

Description automatically generated

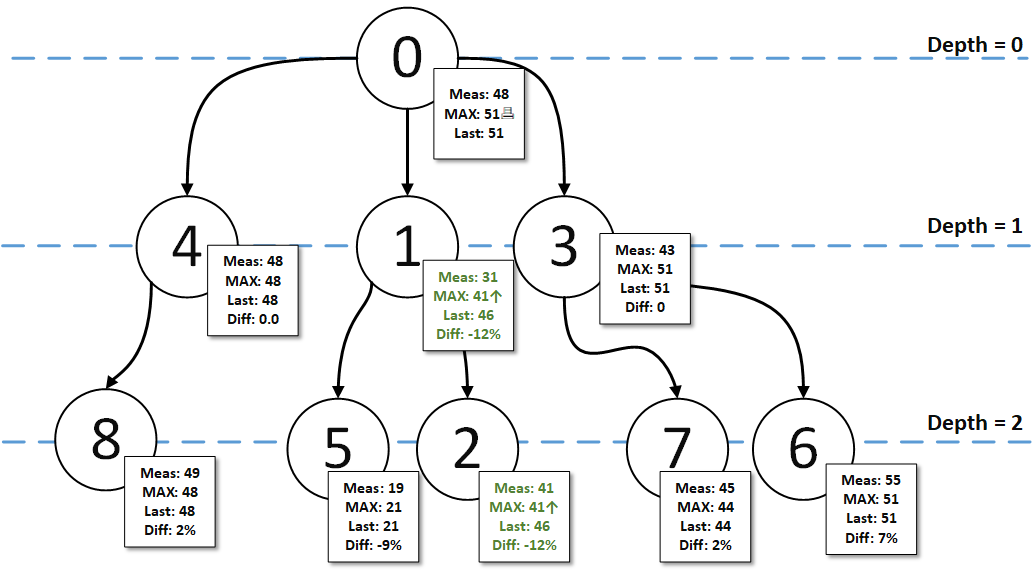
* Ενδεικτικό παράδειγμα εκτέλεσης ΜΑΧ() σε grid 9x9 με range 1.5 και TCT = 10%

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