# ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

# Text Description automatically generated with medium confidenceΣχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Η/Υ

### 

### 

ΠΛΗ 511 - Επεξεργασία και Διαχείριση Δεδομένων σε

Δίκτυα Αισθητήρων

Μέρος 2ο

### Φοιτητές : Διδάσκων :

Σκλάβος Παναγιώτης 2018030170 Δεληγιαννάκης Αντώνιος

ΧρυσήςΕπαμεινώνδας 2018030167

## Α. Εισαγωγή

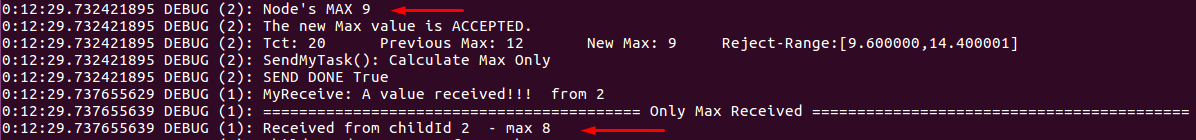
Στην δεύτερη φάση της εργασίας του μαθήματος η προσοχή εστιάζεται στην υλοποίηση μιας λειτουργικότητας εναλλαγής του συνόλου των ερωτημάτων που αποστέλλονται από το δίκτυο. Η εναλλαγή αυτή δεν επηρεάζει τον τρόπο λειτουργίας του δικτύου που υλοποιήθηκε κατά την προηγούμενη φάση, καθώς η λειτουργικότητα συνεχίζει να εκτελείται με εφαρμογή της βελτιωμένης παραλλαγής ΤiNa που περιγράφηκε στο μάθημα. Το σημείο στο οποίο η λειτουργικότητα παρεκκλίνει, είναι στην επιλογή εκτέλεσης ενός διαφορετικού συνόλου συναθροιστικών συναρτήσεων με πιθανότητα 10% από τον σταθμό βάσης, στην αρχή κάθε νέας εποχής. Σε περίπτωση που επιλέγεται να εκτελεσθεί νέα επιλογή συναρτήσεων, η λειτουργία του δικτύου θα πρέπει να ενημερώνεται άμεσα και να μεταβάλλεται κατά την ίδια εποχή που λαμβάνεται η απόφαση ανανέωσης της λειτουργίας.

## Β. Επιδιορθώσεις και Προσαρμογές Κώδικα Φάσης Α

### Β.1 Μάσκες

Όπως παρουσιάσαμε και παραπάνω ο τρόπος λειτουργίας του δικτύου της προηγούμενης φάσης δεν θα επηρεαστεί, ωστόσο, προκειμένου, να μπορέσει να παρουσιαστεί με μεγαλύτερη ακρίβεια η νέα λειτουργικότητα που εντάχθηκε στο δίκτυό μας, οφείλει να γίνει μια αναφορά στις επιδιορθώσεις και σύντομες τροποποιήσεις που πραγματοποιήθηκαν στον παραδοτέο κώδικα του προηγούμενου ερωτήματος.

Η σπουδαιότερη αλλαγή που πραγματοποιήθηκε αποτελεί η προσαρμογή των μασκών που είχαν χρησιμοποιηθεί κατά το μέρος Α για την εξοικονόμηση χώρου στα πακέτα αποστολής όταν το operation ζητούσε να αποσταλούν και οι δύο συναθροισικές αλλά μόνο μία ξεπερνούσε το φράγμα δομούμενο από το tct. Συγκεκριμένα, είχε γίνει λανθασμένη κατανόηση της δομής του 8-bit integer που αποστέλλαμε, με αποτέλεσμα η μάσκα να εφαρμόζεται στο τέλος, όπου βρίσκεται το λιγότερο σημαντικό bit και όχι στην αρχή. Η κατανόηση του σφάλματος αυτού δεν έγινε άμεσα εμφανής, με αποτέλεσμα πολλές φορές να αποστέλλεται εσφαλμένη πληροφορία. Καλό θα είναι τα παραπάνω να επεξηγηθούν με ένα παράδειγμα. Το ακόλουθο αποτελεί ένα στιγμιότυπο από το μέρος Α όπου παρουσιάζεται το σφάλμα αυτό.



Βλέπουμε λοιπόν ότι ο κόμβος 2 αποστέλλει πληροφορία με τιμή 9(1001). Επειδή, όμως η μάσκα εφαρμόζεται στο τελευταίο bit η πληροφορία που διαβάζεται, αφαιρώντας την μονάδα στο τελευταίο bit είναι, όπως φαίνεται από το δεύτερο βέλος: 8(1000). Το φαινόμενο αυτό δημιουργούσε ορισμένα προβλήματα, τα οποία πολλές φορές πέρασαν απαρατήρητα.

Η εσφαλμένη υλοποίηση που είδαμε παραπάνω, επιδιορθώθηκε, αντικαθιστώντας τις μάσκες με τις εύστοχες υλοποιήσεις τους. Αναλυτικότερα, η μάσκα για αποστολή του count, μεταβλήθηκε σε 0x7F = 0111 1111 και η πληροφορία που αποστέλλεται είναι η: (count & 0x7F). Έτσι στην πρώτη θέση λαμβάνουμε με βεβαιότητα 0 (φιλτράρισμα) και στις υπόλοιπες όλη την πληροφορία του count. Αντίστοιχα η μάσκα για αποστολή του max, μεταβλήθηκε σε 0x80= 1000 0000 και η πληροφορία που αποστέλλεται είναι η: (max | 0x80). Έτσι στην πρώτη θέση λαμβάνουμε με βεβαιότητα 1 (προσθήκη) και στις υπόλοιπες όλη την πληροφορία του max.

Τώρα για την αντίστροφη διαδικασία αφαίρεση της μάσκα έχουμε τα εξής:­ Ελέγχεται η πρώτη θέση προκειμένου να αποφανθούμε για το είδος της αποστολής. Αυτό γίνεται ώς εξής: (0x80 & measurementReceived) . Με τον τρόπο αυτό απομονώνεται το πρώτο ψηφίο το οποίο αν είναι 0 διαβάζουμε count ενώ αν είναι 1 διαβάζουμε max. Έπειτα η αποκωδικοποίηση της πληροφορίας είναι ιδιαίτερα απλή. Για το count εφόσον έχουμε 0xxx xxxx η πληροφορία που έχουμε λάβει δεν χρειάζεται καμία επεξεργασία. Από την άλλη για το max η πληροφορία που λαμβάνουμε είναι της μορφής 1xxx xxxx και για να μπορέσουμε να λάβουμε την τιμή του max φιλτράρουμε ως εξής (0x7F & measurementReceived). Έτσι καταλήγουμε με ολόκληρη την πληροφορία του max με ένα μηδενικό στο πρώτο μέλος.

Η υλοποίηση αυτή ωστόσο θέτει και έναν περιορισμό. Πραγματοποιούμε την υπόθεση ότι για την αναπαράσταση του count και του max ποτέ δεν θα χρησιμοποιείται το πρώτο ψηφίο, η οποία είναι αρκετά ακριβής με βάση τον τρόπο που υλοποιείται το πρόγραμμά μας (tct και random\_step\_size). Ωστόσο για αποφευχθεί μεγάλο σφάλμα σε περίπτωση που ο αριθμός που αποστέλλουμε είναι μεγαλύτερος του 127(0111 1111- μεγαλύτερος 7-bit αριθμός), τοποθετήθηκε μια δικλείδα ασφαλείας ώστε να θεωρείται ότι έχουμε 127. Για παράδειγμα αν είχαμε max 131(1000 0011) και εκείνο ξεπερνούσε το tct τότε θα αποστέλλαμε με χρήση μάσκας τον αριθμό 131, άλλα κατά την αποκωδικοποίηση θα λαμβάναμε υπ’ όψιν μόνο τα δεξιά 7-bit. Οπότε ο αριθμός που θα λαμβάναμε θα ήταν το 3. Οπότε μιας και το μεγαλύερο που μπορούμε να αποκωδικοποιήσουμε είναι το 127, τότε θα πραγματοποιήσουμε την λογική για έλεγχο tct και αποστολή στο 127.

### Β.3 Λοιπές τροποποιήσεις.

Δύο ακόμη μικρές τροποποιήσεις πραγματοποιήθηκαν στον παραδοτέο κώδικα του Α Μέρους οι οποίες αφορούν την λήψη τυχαίας τιμής από κάθε κόμβο. Πιο Συγκεκριμένα, τροποποιήθηκε η οριακή τιμή κατά την λήψη της πρώτης τιμής από έναν κόμβο καθώς, υπήρχε μια μικρή αστοχία κατά τα ζητούμενα και ο κόμβος λάμβανε μια τυχαία τιμή στο εύρος [1,79] και όχι στο ζητούμενο [1,80]. Επίσης κατά την τυχαία αλλαγή της αρχικής τιμής κατά +-random step size, δεν είχε ληφθεί ο περιορισμός να μην γίνεται αποδεκτή η τιμή 0. To oποίο μεταβλήθηκε ώστε η τιμή να είναι ελάχιστα 1. Εφόσον δεν είχε ειπωθεί κάτι για την μέγιστη μέτρηση μετά την πρώτη περίοδο δεν έχει υλοποιηθεί κάποιος περιορισμός ως προς το πάνω όριο του εύρους(80) παρά μόνο ο περιορισμός που αναλύθηκε στην προηγούμενη ενότητα όπου περιγράφηκε η αντιμετώπιση τέτοιων τιμών κατά την αποστολή. Εν συντομία, αν παραμένει 7-bit αριθμός αποστέλλεται ως έχει, ενώ αν γίνεται μεγαλύτερος από 7-bit, τότε τοποθετείται η τιμή του στο όριο 127 του μεγαλύτερου 7- bit αριθμού.

Στο βοηθητικό αρχείο έπειτα έγινε μικρή μεταβολή ώστε να μην τοποθετούνται στο αρχείο οι συνδέσεις του κόμβου με τον εαυτό του, καθώς δεν είναι κάτι που επηρεάζει την λειτουργία και κατά τα άλλα απλά τοποθετεί ακόμη περισσότερο φόρτο στο διάβασμα του αρχείου.

Στο αρχείο mySimulation.py, έχει γίνει τοποθέτηση μεταβλητής D την οποίο θα πρέπει να μεταβάλλει ο χρήστης ανάλογα με την διάμετρο του δικτύου ώστε να μπορεί γίνονται οι απαραίτητες αρχικοποιήσεις και το πείραμα να τρέχει για όσους κόμβους απαιτείται. Μικρή μεταβολή καθώς στο προηγούμενο παραδοτέο απλά βάζαμε άμεσα τον μέγιστο αριθμό κόμβων. Επίσης η τιμή του θορύβου έχει τεθεί σε 0 για ευκολότερη αποσφαλμάτωση κατά τους πειραματισμούς. Αλλά μπορεί εύκολα να μεταβληθεί εντός του αρχείου

Τέλος παρατίθεται μια συμπληρωματική περιγραφή για τις σταθερές που βρίσκονται στο header file SimpleRoutingTree.h καθώς δεν έγινε εμφανές στην προηγούμενη αναφορά. Προκειμένου να έχουμε, λοιπόν, ακριβέστερη αντιμετώπιση για κάθε δίκτυο καλό θα είναι να επικεντρώσουμε την προσοχή μας σε δύο σταθερές. Ξεκινώντας από την σημαντικότερη, ο χρήστης ανάλογα με τις διαστάσεις(διάμετρο) του δικτύου καλό θα είναι να παρατηρεί τον χρόνο για τον οποίο αποστέλλονται μηνύματα routing και να τροποποιεί αναλόγως την σταθερά ROUTING\_TIME ώστε να είναι περίπου κατά ένα δευτερόλεπτο τουλάχιστον (προσωπική προτίμηση) περισσότερο από τον χρόνο που απαιτείται. Αυτό για να είμαστε βέβαιοι ότι το Routing ολοκληρώνεται με ασφάλεια. Για τον δικό μας υπολογιστή ισχύουν τα ακόλουθα:

|  |  |
| --- | --- |
| DIAMETER (D) | RECOMMENDED ROUTING\_TIME (ms) |
| 3, 4, 5 |  |
| 6, 7, 8, 9 |  |
| 10 |  |

Εν γένει τοποθετώντας το ROUTING TIME ίσο με 3s, θα αφήσει λίγο παραπάνω από μισό δευτερόλεπτο ακόμα και με διάμετρο 10.

Η άλλη σημαντική σταθερά την οποία θα πρέπει να λαμβάνουμε υπ’ όψιν στα διαφορετικά δίκτυά μας είναι το TIMER\_FAST\_PERIOD. Όσο μεγαλώνουμε την τιμή του τόσο αυξάνεται η ασφάλεια αποφυγής συγκρούσεων καθώς παρέχεται περισσότερος χρόνος σε κάθε επίπεδο(βάθος) να αποστείλει τα δεδομένα του. Επίσης παρέχεται μεγαλύτερη τυχαιότητα από το same level variator ώστε να μην αποστέλλουν ταυτόχρονα κόμβοι του ίδιου επιπέδου. Σε κάθε περίπτωση ωστόσο από τα διάφορα πειράματα που διεξήγαμε αν έχουμε range 1.5 τότε ακόμη και με TIMER\_FAST\_PERIOD = 128 ms, οι απουσίες είναι πολύ μικρές οπότε αποφασίστηκε να επιλεγεί αυτή η τιμή του μειώνοντας τον χρόνο που θα λαμβάνονται τα δεδομένα κατ’ επέκτασιν από τους κόμβους. Ωστόσο αν επιθυμεί κάποιος μεγάλη ακρίβεια, ελαχιστοποιώντας τις συγκρούσεις και απώλειες καλό θα ήταν για δίκτυα μεγάλων διαστάσεων (μεγαλύτερων από διάμετρο 6) η τιμή του να τίθεται στο αρχικό 256 ms, παρόλο που το 128 ms ανταποκρίνεται επαρκώς. Αντίστοιχα σε μικρά δίκτυα ακόμα και 64ms μπορούν να χρησιμοποιηθούν χωρίς τεράστιες απώλειες. Συνεπώς η τελική τιμή υπολογίζεται ανάλογα με τις προτεραιότητες του χρήστη. Στην δική μας περίπτωση αφήνεται ως default τα 128 ms καθώς μειώνουν τον χρόνο που ακούν οι κόμβοι χωρίς πολύ σημαντικές απώλειες στο δίκτυο ακόμη και για μεγάλο αριθμό κόμβων. Προτεινόμενη ρύθμιση για μικρότερo χρόνο παραλαβής πακέτων, με μικρές απώλειες:

|  |  |
| --- | --- |
| DIAMETER (D) | RECOMMENDED TIMER\_FAST\_PERIOD (ms) |
| 3, 4 |  |
| 5, 6 |  |
| 7, 8, 9, 10 |  |

Όλα τα παραπάνω είναι αφορούν topology αρχεία με range of sensors 1.5, αλλά προσαρμόζονται ανάλογα και για οποιοδήποτε άλλο range.

## Γ. Επεξήγηση Κώδικα Φάσης Β

### Γ.1 Προσθήκες σε Υλοποίηση Φάσης Α

Σε αυτό το σημείο πρόκειται να αναλυθούν οι προσθήκες που πραγματοποιήθηκαν στον configuration του δικτύου καθώς και στα declarations των μηνυμάτων ώστε να επιτευχθεί η νέα λειτουργικότητα εναλλαγής ερωτημάτων.

Ξεκινώντας από τις προσθήκες στο configuration, προκειμένου να αποφευχθεί η σύγχυση και η άτακτη αποστολή δεδομένων προστέθηκε για την υλοποίησή μας ένας νέος Timer με ονομασία UpdateOpTimer ο οποίος ως σκοπό έχει την οργάνωση της αποστολής του μηνύματος αλλαγής operation, ανά επίπεδο. Η επιλογή έγινε με το σκεπτικό ότι επιθυμούμε να έχουμε τους κόμβους να αποστέλλουν με συνέπεια το μήνυμα περιοδικά και όχι κατά άτακτο τρόπο, καθώς θα πρέπει να αποστέλλουν την αλλαγή, αφότου την λάβουν οι ίδιοι από τους άλλους κόμβους. Επίσης η αποστολή από κάθε κόμβο για το την αλλαγή συναθροιστικής γίνεται με broadcast, καθώς αφενός θέλουμε να μειώσουμε το πλήθος τον μηνυμάτων που αποστέλλουμε και αφετέρου πάντα μπορεί να υπάρξει απώλεια στα μηνύματα εντός ενός μεγάλου δικτύου. Συνεπώς, με τον τρόπο κάθε κόμβος αποστέλλει μια φορά την πληροφορία - όχι ένα μήνυμα για κάθε παιδί- και ακόμη και αν κάποιο μήνυμα από τον πατέρα χαθεί, ο κόμβος μπορεί να το λάβει από κάποιο γείτονα ή παιδί του, για τον οποίο βρίσκεται στο range επικοινωνίας.

Έχοντας τα άνωθεν στον νου αποφασίστηκε να διατηρείται η δομή του δέντρου του προηγούμενου ερωτήματος. Το πρόβλημα ωστόσο, με αυτήν την προσέγγιση είναι ότι αν χαθεί κάποιο μήνυμα εναλλαγής συνάθροισης και κάποιος κόμβος συνεχίσει να αποστέλλει έτσι για την προηγούμενη επιλογή, με την διατήρηση του δέντρου εγείρονται πολυάριθμα προβλήματα. Την αντιμετώπιση του ζητουμένου αυτού καλούμασταν να δώσουμε στο πρόγραμμα 3 το οποίο τελικά δεν υλοποιήθηκε.

Ωστόσο η υλοποίηση που παραδίδουμε δεν απαιτείται την μεταβολή του χρονισμού των κόμβων οι οποίοι διατηρούν τον χρονισμό του Προγράμματος 1. Αυτό αιτιολογείται από το γεγονός ότι το μήνυμα για αλλαγή της συναθροιστικής προς αποστολή, αποστέλλεται στην αρχή της περιόδου συνεπώς έχουν προλάβει όλοι οι κόμβοι να ενημερωθούν προτού να αρχίσουν την αποστολή των μετρήσεών τους που γίνεται στο τέλος της.

Εκτός από τον Timer στον configuration file προστέθηκαν επίσης και τα interfaces για το νέο είδος μηνύματος (AMSenderC, AMReceiverC, PacketQueueC-για send και receive αντίστοιχα) καθώς και πραγματοποιήθηκε το wiring τους με στο αρχείο SRTreeAppc.nc για να τα χρησιμοποιήσουμε στο module SRTreeC.nc.

Όπως έχουμε αναφέρει πολλάκις προηγουμένως, έγινε και μία προσθήκη νέου είδους μηνύματος, εξού και η χρήση των προαναφερόμενων interfaces, το οποίο δηλώθηκε στο SimpleRoutingTree.h ως OpMsg και η μόνη πληροφορία που περιέχει είναι εκείνη του νέου operation(8-bit integer).

Αυτά είναι όσα μεταβλήθηκαν ως προς την δομή του Project σε σχέση με την φάση Α, και εφόσον, έχουν παρουσιαστεί όλες οι αλλαγές μπορούμε τώρα να επεξηγήσουμε την λειτουργικότητα που πραγματοποιούν.

### Γ.2 Επεξήγηση Λειτουργίας

Αξίζει να σημειωθεί ότι η λειτουργικότητα που προστίθεται για τα μηνύματα OpMsg προσεγγίζει σε μεγάλο βαθμό εκείνη των RoutingMsg. Για τον λόγο αυτό ανάλογες είναι και οι υλοποιήσεις. Όσα πρόκεται να περιγράψουμε παρακάτω αφορούν το main Module SRTreeC.nc και τις προσθήκες που έγιναν για να συμπληρώσουν την λειτουργία του. Η σειρά που πρόκειται να ακολουθήσουμε ακολουθεί την σειρά εκτέλεσης του κώδικα μέσα στο αρχείο ώστε να είναι κατανοητό κάθε βήμα.

Ξεκινώντας λοιπόν η πρώτη εποχή εκτελείται με βάση την λειτουργικότητα που υλοποιήσαμε στην πρώτη φάση. Στις εποχές που ακολουθούν ωστόσο, προστίθεται επιπλέον λειτουργικότητα. Συγκεκριμένα, όπως αναφέραμε στο προηγούμενο κομμάτι της εργασίας, είχε υλοποιηθεί ένας Timer με ονομασία RoundTimer ο οποίος χτυπούσε στο τέλος μιας εποχής σηματοδοτώντας την αρχή της επόμενης. Στην προηγούμενη υλοποίηση όταν γινόταν Fired απλώς τυπωνόταν ο αριθμός του γύρου εκτέλεσης. Στην ανανεωμένη όμως, η λειτουργικότητα του επεκτείνεται καθώς ο σταθμός βάσης(TOS\_NODE\_ID=0) επιλέγει έναν τυχαίο integer από το 0 έως το 9. Αν ο αριθμός αυτός ισούται με 0, δηλαδή με πιθανότητα (1 από 10), ανανεώνεται η global μεταβλητή op\_changed. Σε περίπτωση λοιπόν που επιλεγεί από τον σταθμό βάσης ότι θα γίνει αλλαγή της συναθροιστικής διαλέγουμε τυχαία ένα από τα άλλα 2 διαθέσιμα operations και ανανεώνουμε το παλιό. Έτσι στην φάση αυτή καλείται oneShot ο UpdateOpTimer με όρισμα 0 ώστε να εκκινηθεί άμεσα η διαδικασία ενημέρωσης.

Η σημασία της μεταβλητής οp\_changed είναι εξαiρετικά σημαντική γι αυτό θα αναλύσουμε ευθέως την χρήση της ώστε να είναι ευκολότερα κατανοητή όταν αναφέρεται αργότερα. Η μεταβλητή λοιπόν, στην αρχή κάθε γύρου(RoundTimer.fired()) αρχικοποιείται ως FALSE. Αν κάποια στιγμή κατά την διάρκεια του γύρου η τιμή της αλλάξει σε TRUE τότε γνωρίζουμε ότι έχει πραγματοποιηθεί αλλαγή σε αυτόν τον γύρω. Περισσότερες πληροφορίες θα δοθούν ακολουθώνας την υλοποίηση.

Όταν λοιπόν χτυπήσει ο updateOpTimer του σταθμού βάσης εκκινείται η διαδικασία αποστολής. Με ανάλογο τρόπο λοιπόν με το RoutingMsg, εφόσον ολοκληρωθούν οι κατάλληλοι έλεγχοι, συμπληρώνεται τον OpMsg με το νέο operation και δρομολογείται για αποστολή αφότου τοποθετηθεί στην κατάλληλη ουρά OpSendQueue. Έτσι λοιπόν γίνεται post το task για αποστολή του μηνύματος sendOpTask(), και η λειτουργία μεταφέρεται σε αυτό, όπου και γίνεται η αποστολή με broadcast, αν δεν υπάρχει κάποιο σφάλμα.

Mε την ολοκλήρωση του send οδηγούμαστε στην OpAMSend.sendDone() όπου αν έχει μπεί κάποιο άλλο μήνυμα στην ουρά και περιμένει να αποσταλεί, δρομολογείται για να ξεκινήσει κάνοντας ξανά post.

Έτσι ολοκληρώνεται η διαδικασία ενημέρωσης από τον κόμβο βάσης και μεταφέρουμε το ενδιαφέρον μας στους κόμβους παραλήπτες και την συνάρτηση OpReceive.receive(). Εκεί παραλαμβάνεται το πακέτο και δρομολογείται για επεξεργασία εισάγοντάς το στην ουρά OpReceiveQueue και κάνοντας post το receiveOpTask().

Συνεχίζοντας εντός του receiveOpTask(), το OpMsg γίνεται dequeue από την ουρά και ελέγχεται αν έχει πατέρα. Ο έλεγχος αυτός γίνεται σε περίπτωση που κάποιος κόμβος δεν είχε ακούσει το RoutingMsg και δεν είχε τοποθετηθεί στο δέντρο, άλλα άκουσε από τα νέα broadcast για την αλλαγή operation. Αν ο κόμβος λοιπόν που παρέλαβε το μήνυμα εντάσσεται σε αυτή την κατηγορία, τότε σταματάει εκεί την διαδικασία του receive και δεν εκτελεί κάποια ενέργεια καθώς όντας unrouted θα έχει ελλιπή πληροφόρηση που θα μπορούσε να επηρεάσει την λειτουργικότητα του δικτύου. Συγκεκριμένα, θα μπορούσε να αναθέσει ως πατέρα εκείνον που απέστειλε το μήνυμα καθώς θα μπορούσε να αποσπάσει την πληροφορία μέσω της συνάρτησης OpAMPacket.source, και θα μπορούσε να γνωρίσει το νέο operation προς αποστολή. Ωστόσο το opMsg δεν περιέχει την πληροφορία του βάθους και του tct που εμπεριείχε το RoutingMsg. Βασικότερο εκ των 2 είναι το πρώτο καθώς δεν θα γνωρίζουμε πότε να αποστέλλουμε (σε ποίο timeslice με βάση βάθος) οπότε ο πατέρας θα έπρεπε να λαμβάνει δεδομένα για μεγαλύερο χρόνο καθώς και ο νέος κόμβος δεν θα λειτουργεί με βάση το ζητούμενο πρωτόκολλο.

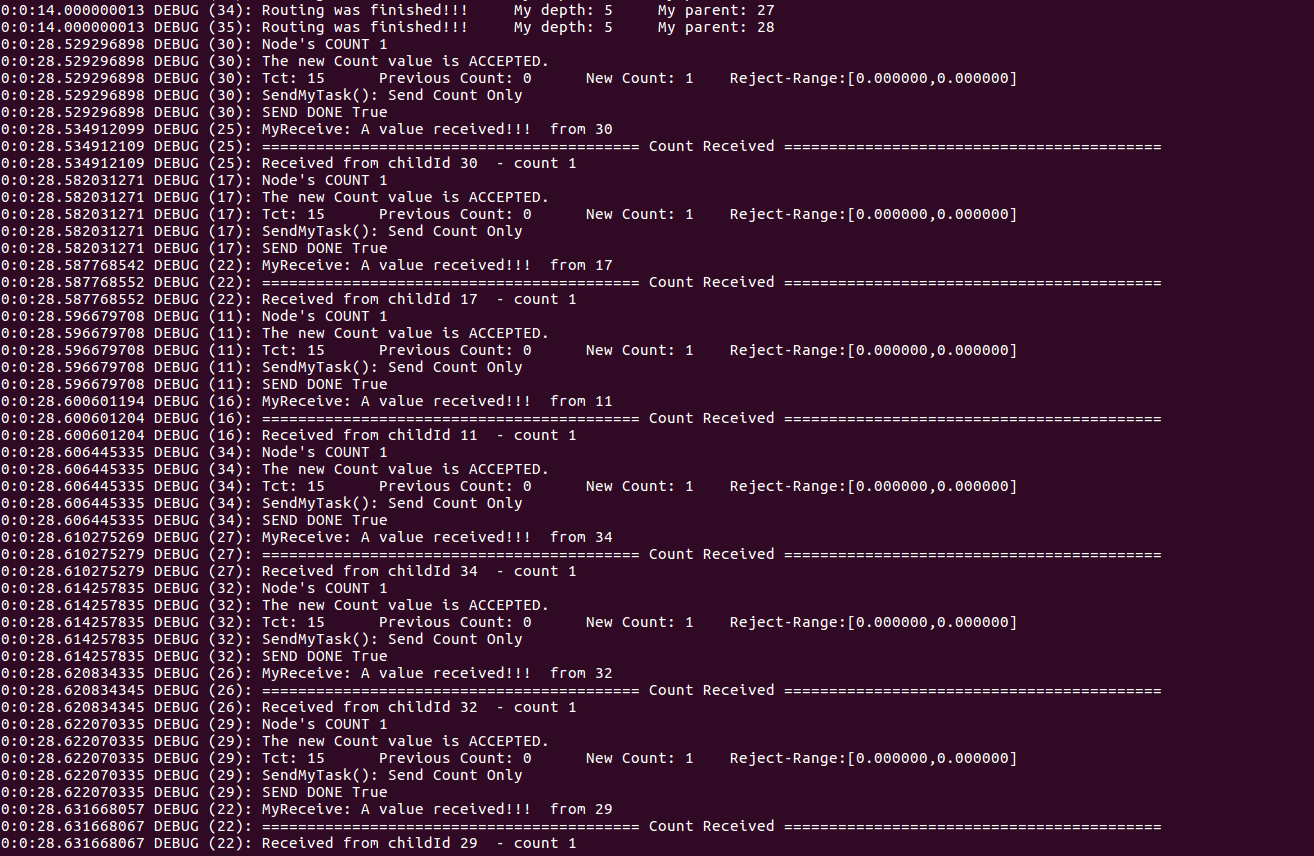
Διαφορετικά, αν ο κόμβος που έλαβε το μήνυμα διαθέτει πατέρα, τότε ελέγχεται αν η μεταβλητή op\_changed που αναφέραμε νωρίτερα είναι FALSE. Ο έλεγχος αυτός πραγματοποιείται καθώς ένας κόμβος συνήθως λαμβάνει πολυάριθμα broadcasts, για ανανέωση operation από τους γείτονές του, οπότε η ακόλουθη διαδικασία θέλουμε να ακολουθείται μια φορά και όχι επαναλαμβανόμενα για κάθε broadcast που λαμβάνει. Συνεπώς αν δεν έχει μεταβληθεί το operation του κόμβου κατά αυτόν τον γύρω, τότε θέτει το operation του στην τιμή που έλαβε από το μήνυμά του και εφόσον δεν μιλάμε για τον κόμβο βάσης που ήταν εκείνος που εκκίνησε την διαδικασία, θέτουμε το οp\_changed σε TRUE και κάνουμε OneShot call στον UpdateOpTimer με όρισμα TIMER\_FAST\_PERIOD. Κατά τον τρόπο αυτό, το μήνυμα θα συνεχίσει να λαμβάνεται και να αποστέλλεται από τους επόμενους κόμβους, με μια κατά depth προσέγγιση, δηλαδή με την σειρά που έλαβαν το broadcast για αλλαγή operation.

Έτσι συνοψίζεται η διαδικασία ενημέρωσης η οποία θα συνεχίσει να μεταδίδεται έως να μην υπάρχει κόμβος βαθύτερα να μεταδώσει. Συνεπώς, όλη η διαδικασία συνοψίζεται σε λίγο χρόνο μετά την αρχή της περιόδου, με αποτέλεσμα όταν οι κόμβοι επιχειρήσουν να αποστείλουν τις τιμές στο τέλος της εποχής, να λειτουργούν όλοι με βάση τη νέα ζητούμενη συναθροιστική. Βέβαια, σε περιπτώσεις όπου οι συνδέσεις μεταξύ των κόμβων δεν είναι πολλές πχ range 1, τότε ελαφρά συνδεδεμένοι κόμβοι κινδυνεύουν να χάσουν το μήνυμα μεταβολής συναθροιστικής, διακινδυνεύοντας έτσι την ακεραιοότητα των απότελεσμάτων. Η υλοποίηση που περιγράψαμε λοιπόν δεν καλύπτει το ενδεχόμενο αυτό το οποίο θα ερχόταν με την υλοποίηση του 3ου Προγράμματος, αλλά περιορίζεται στα ζητούμενα του Προγράμματος 2.

## Δ. Παρουσίαση Λειτουργίας και Επεξήγηση Αποτελεσμάτων

Η παρουσίαση που ακολουθεί γίνεται πάνω σε ένα δίκτυο αισθητήρων με διάμετρο 6 και range 1,5 ενώ ο θόρυβος τίθεται ίσο με μηδέν για να φανούν καθαρά τα αποτελέσματα. Στο δίκτυο αυτό θα παρατηρήσουμε όλες τις καταστάσεις λειτουργίας που περιγράφονται από την εξής υλοποίηση.

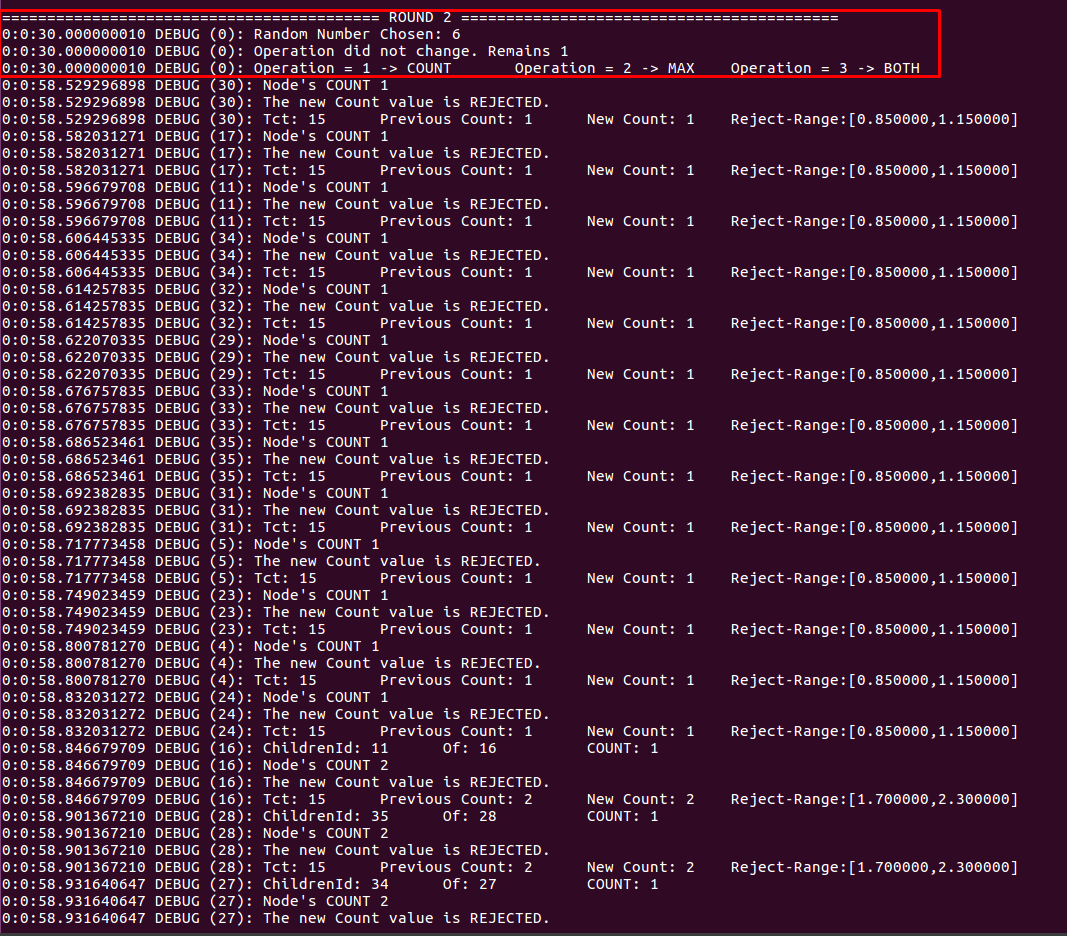
### Δ.1 Πρώτη Εποχή-Λειτουργία όπως μέρος Α



Δ.1. Στιγμιότυπο λειτουργίας μετά το Routing την πρώτη περίοδο

Παρατηρόυμε παραπάνω ότι το αφότου ολοκληρωθεί το Routing αρχίζει να εκτελείται κανονικά η λειτουργία του προγράμματος όπως είχαμε δει στην παράδοση του πρώτου μέρους. Συγκεκριμένα εδώ παρατηρούμε να έχει επιλεγεί operation = Count και οι κόμβοι έχουν αρχίσει την λειτουργία τους χωρίς καμία διαφοροποίηση με παλιά

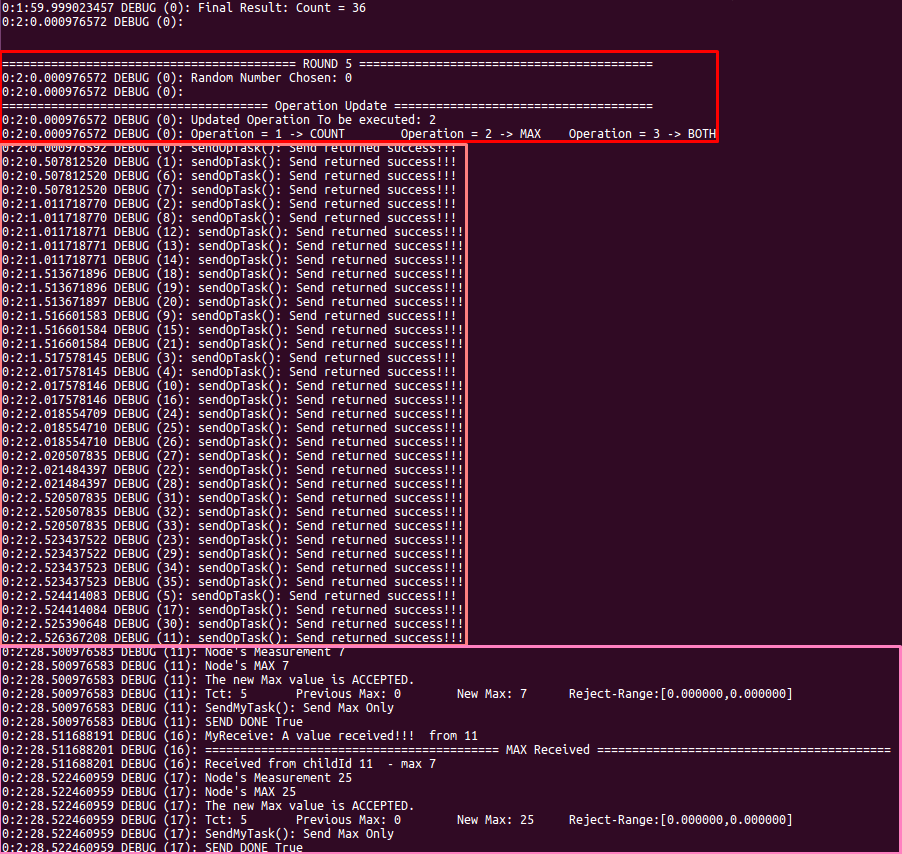
### Δ.2 Δεύτερη Εποχή Λειτουργίας



Δ.2. Στιγμιότυπο λειτουργίας της Δεύτερης Περιόδου

Κατά την δεύτερη περίοδο όπως παρατηρούμε και από την κυκλωμένη επιφάνεια επιλέγεται τυχαία ένας αριθμός κατά την αρχή της (simulation time = 30s). Ο αριθμός αυτός δεν ισούται με μονάδα μηδέν οπότε βλέπουμε στην επόμενη γραμμή ότι το operation παραμένει εκείνο που εκτελούνταν. Έπειτα η λειτουργία συνεχίζει κανονικά πραγματοποιώντας την αποστολή των μετρήσεων λίγο πριν το πέρας της εποχής 58.52 s.

### Δ.3 Περίοδος Αλλαγής Operation

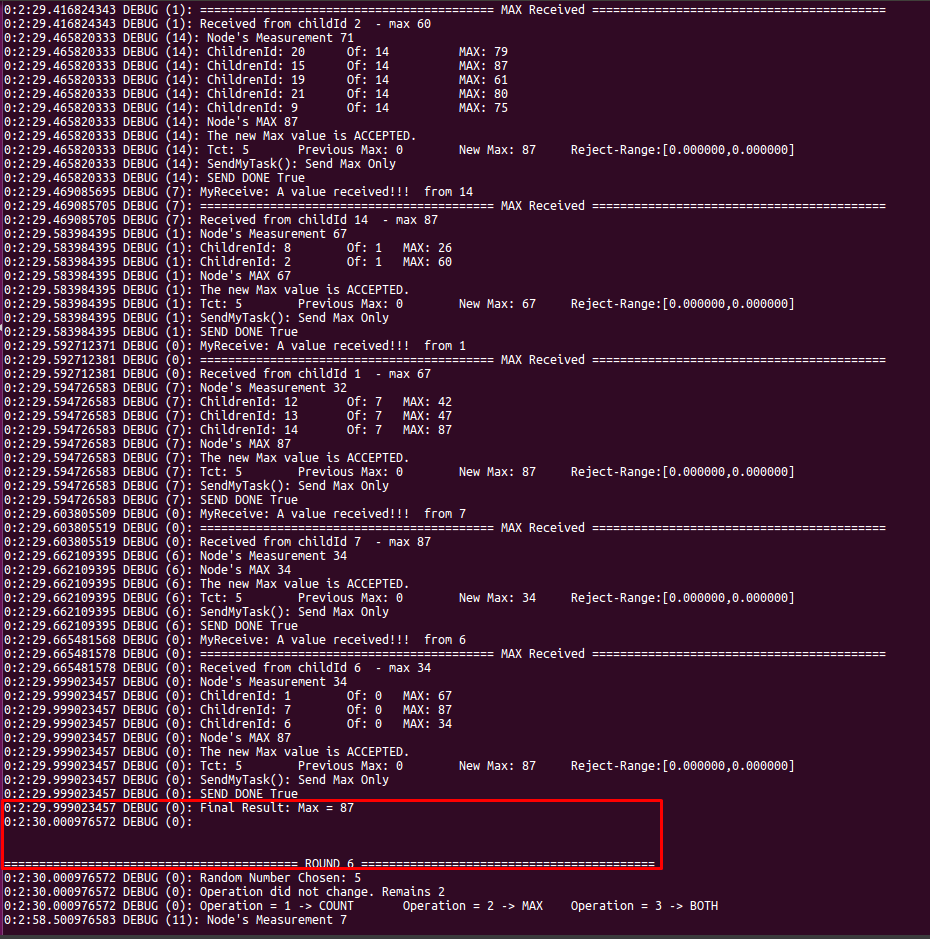


Δ.3.1 Στιγμιότυπο λειτουργίας της Περιόδου αλλαγής συναθροιστικής

Ξεκινώντας βλέπουμε να εκτυπώνεται στην κορυφή το τελευταίο αποτέλεσμα της προηγούμενης εποχής η οποία εκτελούσε count operation. Σε αυτό το σημείο αξίζει να παρατηρήσουμε τρία κυκλωμένα τμήματα εκτυπώσεων που φαίνονται παραπάνω. Αρχικά στο πρώτο τμήμα με κόκκινη border βλέπουμε την αλλαγή του operation. Συγκεκριμένα με το που ξεκινάει η νέα περίοδος επιλέγεται τυχαίος αριθμός στο range 0-9. Εφόσον ο αριθμός είναι το 0 Αυτό σηματοδοτεί ότι θα γίνει Update του operation όπως τυπώνεται ακολούθως. Το νέο operation που επιλέγεται δεν άλλο από το MAX όπως φαίνεται στις ακόλουθες γραμμές.

Το επόμενο τμήμα είναι το επόμενο κυκλωμένο κομμάτι με πορτοκαλί border που περιέχει μια σειρά εκτυπώσεων με το εξής μήνυμα: SendOpTask(): Send Returned Success!!!. To μήνυμα αυτό επαναλαμβάνεται 36 φορές καθώς όλοι οι κόμβοι που λαμβάνουντην ενημέρωση επιχειρούν να την γνωστοποιήσουν με broadcast στους υπόλοιπους. Αξίζει να παρακολουθήσουμε τον χρόνο των εκτυπώσεων οι οποίες πραγματοποιούνται την διάταξη που περιγράψαμε ακολουθώντας ανα depth λογική. Για παράδειγμα Οι πρώτοι που έλαβαν το broadcast μήνυμα ενημέρωσης από τον κόμβο 0 είναι οι κόμβοι 1,6, 7 οι οποίοι στέλνουν στο πρώτο timeslice. Έπειτα οι δεύτεροι που έλαβαν το broadcast μήνυμα ενημέρωσης από αυτούς είναι οι κόμβοι 2, 8, 12, 13, 14 οι οποίοι στέλνουν στο δεύτερο timeslice κ.ο.κ.

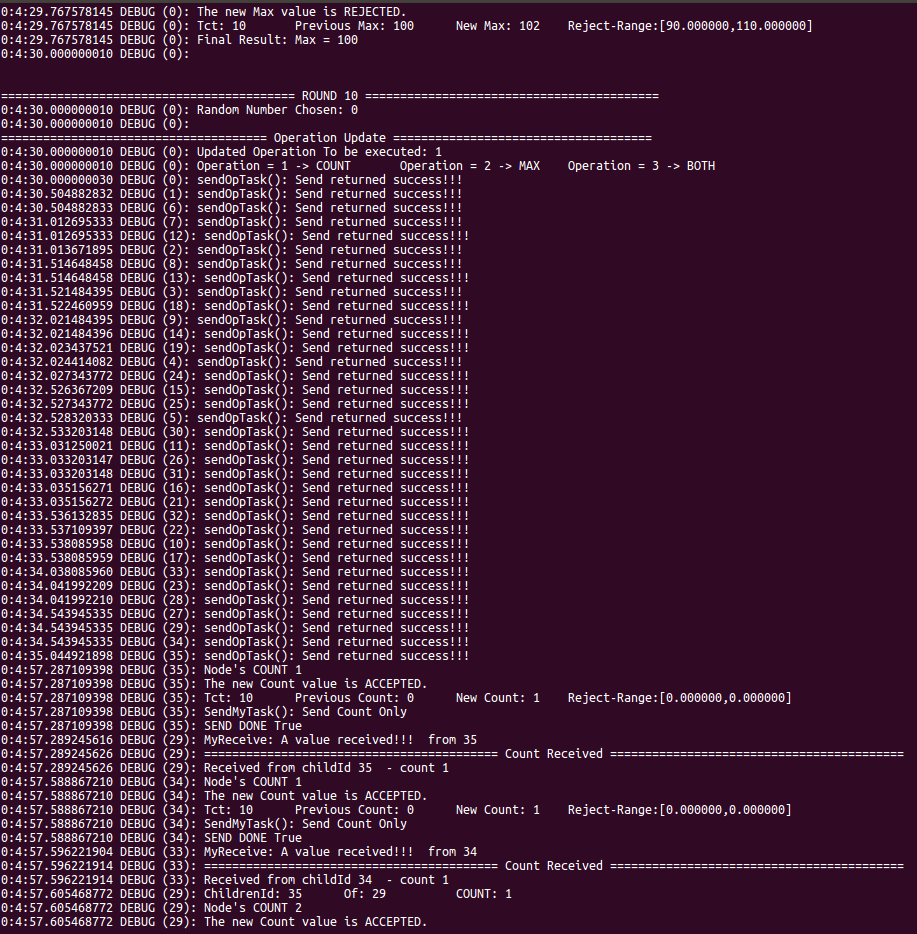
Τέλος στο τρίτο τμήμα με ροζ border βλέπουμε στο τέλος της περιόδου το δίκτυο μας να έχει αρχίσει να λειτουργεί με βάση το νέο του ζητούμενο ήδη από την ίδια εποχή καταλήγοντας στην νέα τιμή MAX για το δίκτυο όπως φαίνεται παρακάτω. Και με τον τρόπο αυτό έπεται η λειτουργία ώσπου να αλλάξει ξανά η συναθροιστική.



Δ.3.2 Στιγμιότυπο Παρουσίασης αποτελέσματος νέας συναθροιστικής

### Δ.4 Παράδειγμα δικτύου ίδιων διαστάσεων με range = 1 και Θόρυβο

Η περίπτωση αυτή μπορεί να παράξει ένα πολύ ενδιαφέρον παράδειγμα, καθώς βλέπουμε το ενδεχόμενο που γεννά την ανάγκη για την υλοποίηση του 3ου Προγράμματος. Συγκεκριμένα με τις συνθήκες αυτές – χαμηλή συνδεσιμότητα και θόρυβο στο περιβάλλον, μπορεί με μεγαλύτερη ευκολία κάποιος κόμβος να χάσει το broadcast αλλαγής operation με αποτέλεσμα να συνεχίσει να λειτουργεί με διαφορετικό operation από το κύκλωμα. Για να γίνει περισσότερο εμφανές το φαινόμενο εντοπίστηκε παράδειγμα όπου η μεταβολή γίνεται από MAX σε COUNT. οπότε το COUNT του δικτύου διαμέτρου 36 κόμβων γίνεται ιδιαίτερα μεγάλο. Αρχικά από το ακόλουθο στιγμιότυπο μπορούμε εύκολα να παρατηρήσουμε ότι ένας κόμβος δεν έλαβε το μήνυμα αλλαγής: 20. Αυτό σημαίνει ότι θα συνεχίσει να λειτουργεί κατά το MAX οperation



Δ.4.1 Στιγμιότυπο Παρουσίασης αποστολής αλλαγής με απουσία του κόμβου 20

Η απουσία του κόμβου αυτού οδηγεί έπειτα στο εξής φαινόμενο: τον κόμβο να στέλνει την τιμή 61 και το κόμβο 19 να το λαμβάνει ως count.

Graphical user interface

Description automatically generated

Δ.4.2 Στιγμιότυπο Παρουσίασης εσφαλμένης αποστολής του κόμβου 20

Έτσι καταλήγουμε τελικά στο προφανώς εσφαλμένο αποτέλεσμα:

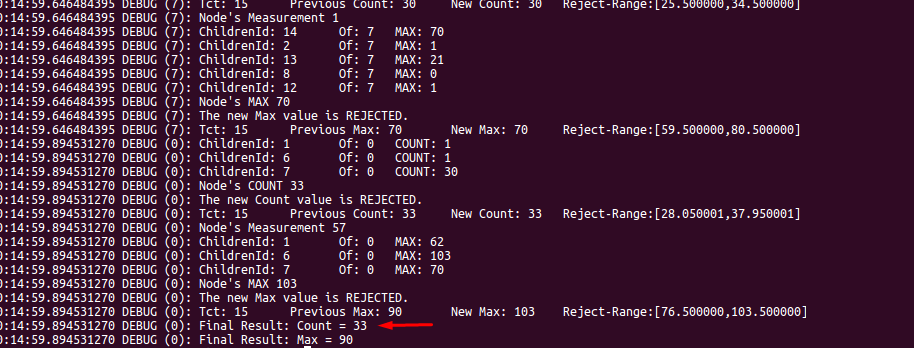
Text

Description automatically generated

Δ.4.2 Στιγμιότυπο Παρουσίασης εσφαλμένης αποτελέσματος για συναθροιστική COUNT

### Δ.5 Παράδειγμα δικτύου με range = 1.5 και Θόρυβο

Όταν ωστόσο υπάρχει θόρυβος, αλλά το δίκτυο έχει πολλούς δεσμούς μεταξύ των κόμβων του παρατηρείται ότι λειτουργεί με μεγαλύτερο ρίσκο. Με άλλα λόγια στα simulations το count που χρησιμοποιείται για ευκολότερη αποσφαλμάτωση, καταλήγει σε τιμές που ποικίλουν, λόγω του ότι χάνονται μηνύματα. Δηλαδή αν το μήνυμα που χαθεί κουβαλάει κάποιο count πολλών προηγούμενων κόμβων τότε το σφάλμα είναι μεγάλο. Αντιθέτως αν χαθούν τα μηνύματα κόμβων σε φύλλα το σφάλμα είναι μικρό. Ένα τυχαίο παράδειγμα είναι το ακόλουθο:

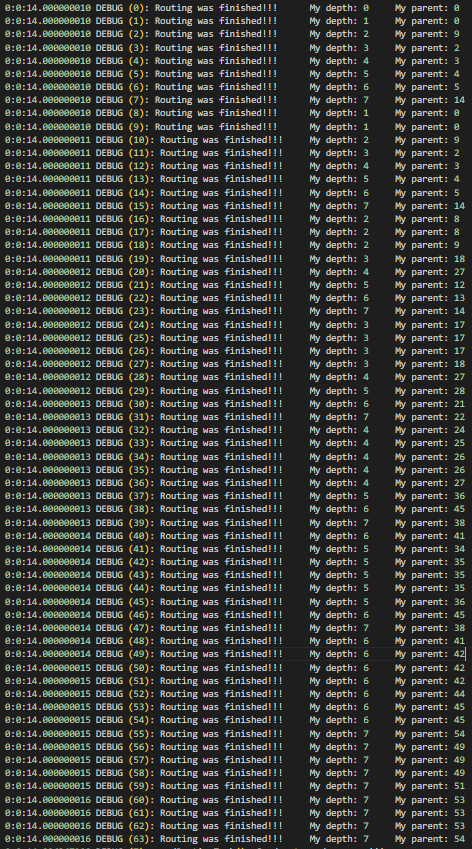


Δ.5 Στιγμιότυπο Παρουσίασης απουσίας μηνυμάτων

Απουσία μηνυμάτων μπορεί ωστόσο να υπάρξει και σε δίκτυα χωρίς θόρυβο ένα οι διαστάσεις τους είναι μεγάλες. Δηλαδή σε κάθε δίκτυο η πιθανότητα απουσίας κάποιου μηνύματος δεν είναι μηδενική.

### Δ.6 Συμπληρωματικό Παράδειγμα: Επίδειξη λειτουργικότητας Φάσης Α μετά από αποσφαλμάτωση σε δίκτυο 8x8 και range 1.5 απουσία θορύβου

Πραγματοποιώντας του Routing Προκύπτει το ακόλουθο αποτέλεσμα:



Δ.6.1 Στιγμιότυπο Παρουσίασης 8x8 grid Routing

Diagram

Description automatically generatedΚαι το δέντρο που κατασκευάζεται ακολουθώντας το παραπάνω είναι το εξής:

Δ.6.1 Δέντρο που προκύπτει ακολουθώντας το Routing

Στο τέλος λοιπόν της πρώτης περιόδου προκύπτει για τον υπολογισμό του count:

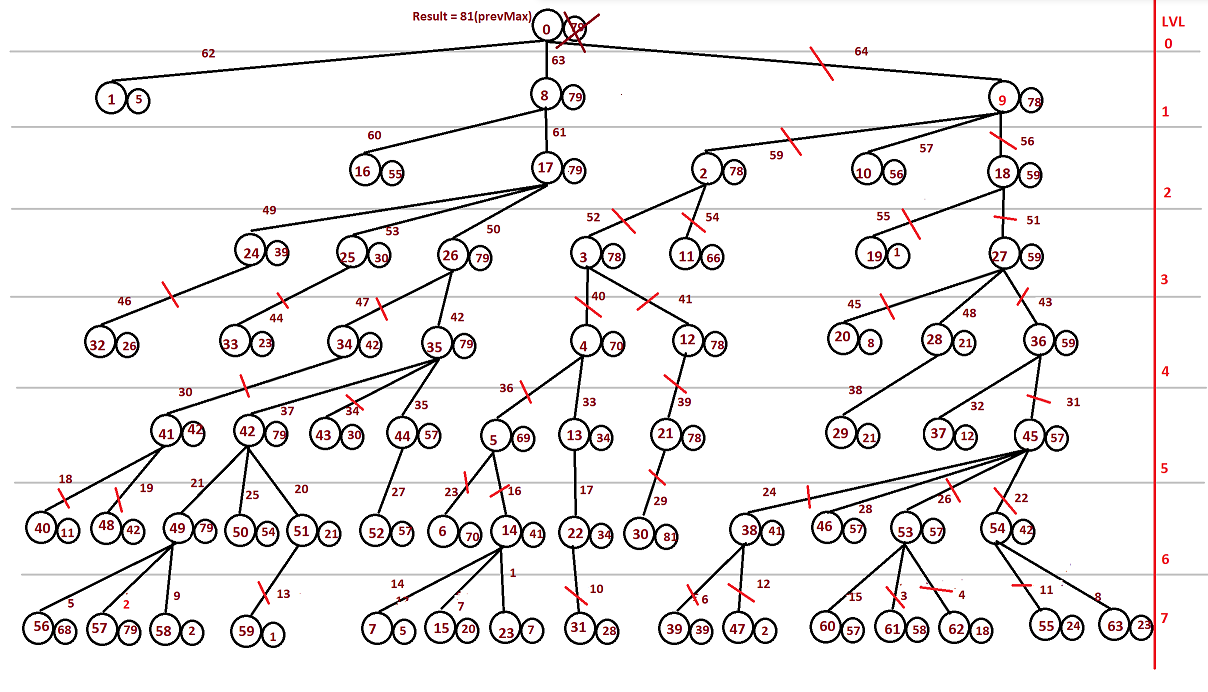
Text

Description automatically generated

Δ.6.2 Υπολογισμός Count και επιβεβαίωση ορθού Routing.

Έτσι λοιπόν βλέπουμε ότι το count προκύπτει ίσο με 64 που ήταν και το επιθυμητό. Η τιμή αυτή θα μπορούσε να είναι και μικρότερη αν κατά την αποστολή χανόταν κάποιο μήνυμα, ωστόσο στο πείραμα που εξετάζουμε κανένα πακέτο μήνυμα count δεν έχει χαθεί λαμβάνοντας την ακρινή τιμή.

Τέλος θα πραγματοποιηθεί μια πλήρης ανασκόπηση της εκτέλεσης που βρίσκεται αποθηκευμένη στο αρχείο output.txt, πραγματοποιώντας την κατά βάθος ανέγερση του max κατά την 7η περίοδο, έως και τον κόμβο βάσης. Οι αριθμοί στις γραμμές μετάβασης συμβολίζουν την σειρά εκτέλεσης ενώ οι αριθμοί στα κελιά συμβολίζουν την νέα τιμή του max που προκύπτει κατά την εποχή αυτή. Σε περίπτωση που η τιμή αυτή δεν απέχει από την προηγουμένη κατά +- tct η περισσότερο τότε απορρίπτεται και αυτό συμβολίζεται με την πλάγια κόκκινη γραμμή στην μετάβαση.



Δ.6.3 Ανέγερση από φύλλα έως ρίζα των υπολογιζόμενων MAX

Σχολιάζοντας τα παραπάνω αποτελέσματα, θα σταθούμε στα 2 σημαντικότερα σημεία του παραπάνω πειράματος: Από την μία λοιπόν, παρατηρούμε ένα πολύ σημαντικό γεγονός. Η μετάδοση της πληροφορίας από τα φύλλα έως την ρίζα επιτυγχάνεται πάντα ανά επίπεδο καθώς παρατηρούμε ότι σε όλο το εύρος του παραδείγματος, ούτε μια φορά ένας κόμβος μικρότερου βάθος δεν αποστέλλει νωρίτερα την πληροφορία του από κάποιο κόμβο μεγαλύτερο. Από την άλλη τώρα, βλέπουμε ότι παρόλο που στο πείραμα έχουμε την μικρότερη δυνατή τιμή tct,(5%), από τις 64 αποστολές που θα γινόντουσαν με TAG με την βελτιωμένη υλοποίηση TiNa που εφαρμόστηκε πραγματοποιούνται μόλις 30. Δηλαδή, αποφεύγεται ποσοστό μεγαλύτερο του 50%.

Σημείωση: Η παραπάνω εικόνα εκτέλεσης καθώς και το αρχείο output.txt θα παραδοθούν με την άσκηση σε περίπτωση που χρειαστεί να μελετηθεί παραπάνω το παράδειγμα.

## Ε. Διαμοιρασμός Εργασιών

Οι εργασίες στο τμήμα αυτό έγιναν κατά κύριο λόγο από κοινού με συχνή ενδοεπικοινωνία. Συνεπώς δεν θα γίνει κάποιος καταμερισμός εργασιών.