ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Η/Υ

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ / **TECHNICAL UNIVERSITY OF CRETE**

ΠΛΗ 511 - Επεξεργασία και Διαχείριση Δεδομένων σε Δίκτυα Αισθητήρων

Μέρος 2ο

Φοιτητές :

Σκλάβος Παναγιώτης 2018030170

Χρυσής Επαμεινώνδας 2018030167

Διδάσκων :

Δεληγιαννάκης Αντώνιος

Α. Εισαγωγή

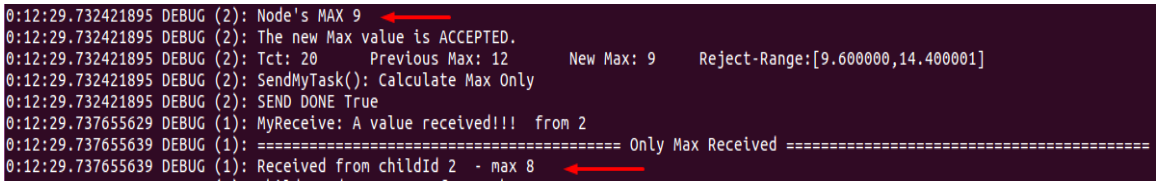
Στην δεύτερη φάση της εργασίας του μαθήματος η προσοχή εστιάζεται στην υλοποίηση μιας λειτουργικότητας εναλλαγής του συνόλου των ερωτημάτων που αποστέλλονται από το δίκτυο. Η εναλλαγή αυτή δεν επηρεάζει τον τρόπο λειτουργίας του δικτύου που υλοποιήθηκε κατά την προηγούμενη φάση, καθώς η λειτουργικότητα συνεχίζει να εκτελείται με εφαρμογή της βελτιωμένης παραλλαγής TiNa 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. Το οποίο 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	$2 \cdot 1024$
6, 7, 8, 9	$3 \cdot 1024$
10	$4 \cdot 1024$

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 μικρότερο χρόνο παραλαβής πακέτων, με μικρές απώλειες:

DIAMETER (D)	RECOMMENDED TIMER_FAST_PERIOD (ms)
3, 4	64
5, 6	128
7, 8, 9, 10	256

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 op_changed είναι εξαιρετικά 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.

Με 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 op_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

```
0:0:14.000000013 DEBUG (34): Routing was finished!!!     My depth: 5     My parent: 27
0:0:14.000000013 DEBUG (35): Routing was finished!!!     My depth: 5     My parent: 28
0:0:28.529296898 DEBUG (30): Node's COUNT 1
0:0:28.529296898 DEBUG (30): The new Count value is ACCEPTED.
0:0:28.529296898 DEBUG (30): Tct: 15      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:28.529296898 DEBUG (30): SendMyTask(): Send Count Only
0:0:28.529296898 DEBUG (30): SEND DONE True
0:0:28.534912099 DEBUG (25): MyReceive: A value received!!!  from 30
0:0:28.534912109 DEBUG (25): ======================================== Count Received ========================================
0:0:28.534912109 DEBUG (25): Received from childId 30  - count 1
0:0:28.582031271 DEBUG (17): Node's COUNT 1
0:0:28.582031271 DEBUG (17): The new Count value is ACCEPTED.
0:0:28.582031271 DEBUG (17): Tct: 15      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:28.582031271 DEBUG (17): SendMyTask(): Send Count Only
0:0:28.582031271 DEBUG (17): SEND DONE True
0:0:28.587768542 DEBUG (22): MyReceive: A value received!!!  from 17
0:0:28.587768552 DEBUG (22): ======================================== Count Received ========================================
0:0:28.587768552 DEBUG (22): Received from childId 17  - count 1
0:0:28.596679708 DEBUG (11): Node's COUNT 1
0:0:28.596679708 DEBUG (11): The new Count value is ACCEPTED.
0:0:28.596679708 DEBUG (11): Tct: 15      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:28.596679708 DEBUG (11): SendMyTask(): Send Count Only
0:0:28.596679708 DEBUG (11): SEND DONE True
0:0:28.600601194 DEBUG (16): MyReceive: A value received!!!  from 11
0:0:28.600601204 DEBUG (16): ======================================== Count Received ========================================
0:0:28.600601204 DEBUG (16): Received from childId 11  - count 1
0:0:28.606445335 DEBUG (34): Node's COUNT 1
0:0:28.606445335 DEBUG (34): The new Count value is ACCEPTED.
0:0:28.606445335 DEBUG (34): Tct: 15      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:28.606445335 DEBUG (34): SendMyTask(): Send Count Only
0:0:28.606445335 DEBUG (34): SEND DONE True
0:0:28.610275269 DEBUG (27): MyReceive: A value received!!!  from 34
0:0:28.610275279 DEBUG (27): ======================================== Count Received ========================================
0:0:28.610275279 DEBUG (27): Received from childId 34  - count 1
0:0:28.614257835 DEBUG (32): Node's COUNT 1
0:0:28.614257835 DEBUG (32): The new Count value is ACCEPTED.
0:0:28.614257835 DEBUG (32): Tct: 15      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:28.614257835 DEBUG (32): SendMyTask(): Send Count Only
0:0:28.614257835 DEBUG (32): SEND DONE True
0:0:28.620834335 DEBUG (26): MyReceive: A value received!!!  from 32
0:0:28.620834345 DEBUG (26): ======================================== Count Received ========================================
0:0:28.620834345 DEBUG (26): Received from childId 32  - count 1
0:0:28.622070335 DEBUG (29): Node's COUNT 1
0:0:28.622070335 DEBUG (29): The new Count value is ACCEPTED.
0:0:28.622070335 DEBUG (29): Tct: 15      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:28.622070335 DEBUG (29): SendMyTask(): Send Count Only
0:0:28.622070335 DEBUG (29): SEND DONE True
0:0:28.631668057 DEBUG (22): MyReceive: A value received!!!  from 29
0:0:28.631668067 DEBUG (22): ======================================== Count Received ========================================
0:0:28.631668067 DEBUG (22): Received from childId 29  - count 1
```

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

Παρατηρούμε 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

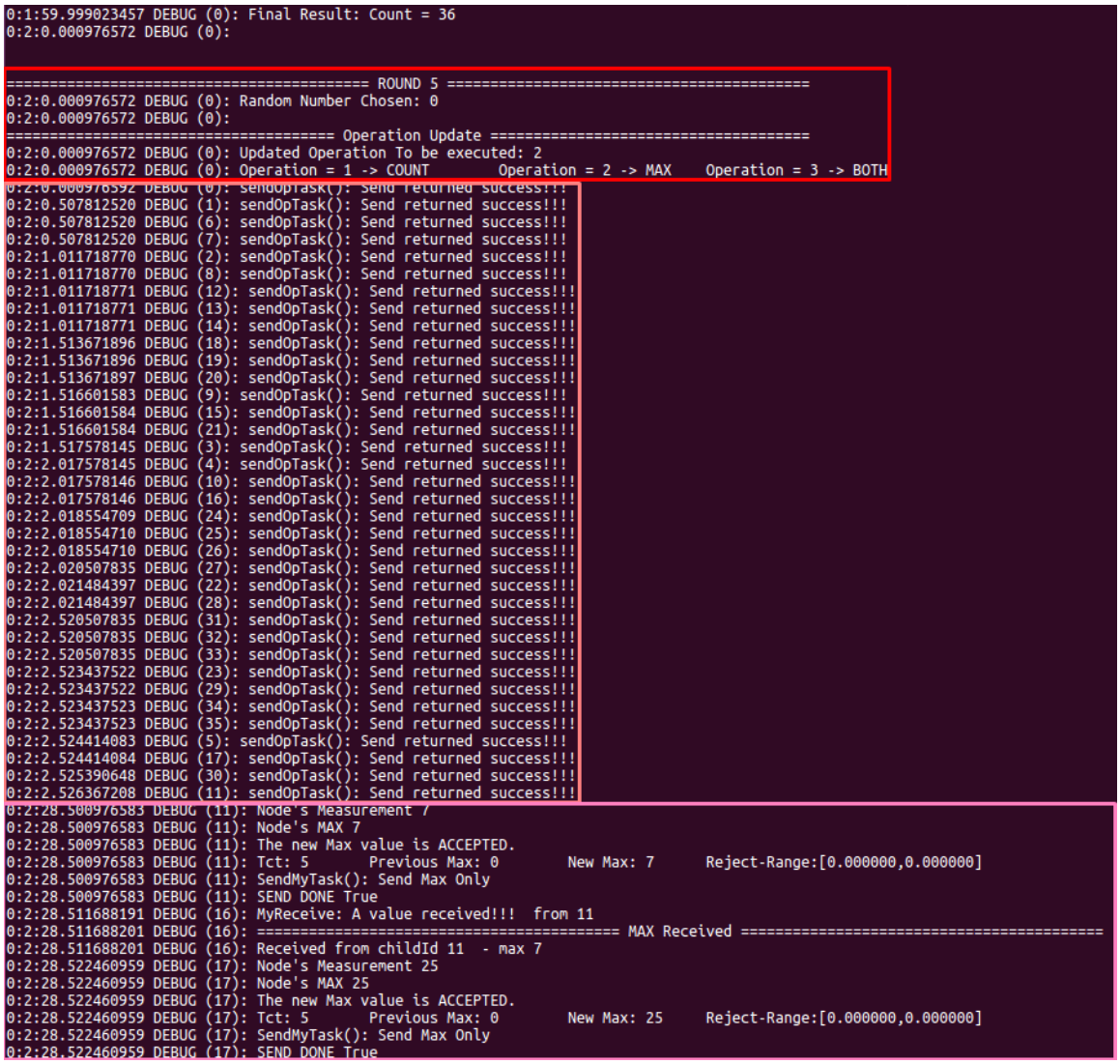
Δ.2 Δεύτερη Εποχή Λειτουργίας

```
======================================== ROUND 2 ========================================
0:0:30.000000010 DEBUG (0): Random Number Chosen: 6
0:0:30.000000010 DEBUG (0): Operation did not change. Remains 1
0:0:30.000000010 DEBUG (0): Operation = 1 -> COUNT      Operation = 2 -> MAX    Operation = 3 -> BOTH
0:0:58.529296898 DEBUG (30): Node's COUNT 1
0:0:58.529296898 DEBUG (30): The new Count value is REJECTED.
0:0:58.529296898 DEBUG (30): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.582031271 DEBUG (17): Node's COUNT 1
0:0:58.582031271 DEBUG (17): The new Count value is REJECTED.
0:0:58.582031271 DEBUG (17): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.596679708 DEBUG (11): Node's COUNT 1
0:0:58.596679708 DEBUG (11): The new Count value is REJECTED.
0:0:58.596679708 DEBUG (11): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.606445335 DEBUG (34): Node's COUNT 1
0:0:58.606445335 DEBUG (34): The new Count value is REJECTED.
0:0:58.606445335 DEBUG (34): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.614257835 DEBUG (32): Node's COUNT 1
0:0:58.614257835 DEBUG (32): The new Count value is REJECTED.
0:0:58.614257835 DEBUG (32): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.622070335 DEBUG (29): Node's COUNT 1
0:0:58.622070335 DEBUG (29): The new Count value is REJECTED.
0:0:58.622070335 DEBUG (29): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.676757835 DEBUG (33): Node's COUNT 1
0:0:58.676757835 DEBUG (33): The new Count value is REJECTED.
0:0:58.676757835 DEBUG (33): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.686523461 DEBUG (35): Node's COUNT 1
0:0:58.686523461 DEBUG (35): The new Count value is REJECTED.
0:0:58.686523461 DEBUG (35): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.692382835 DEBUG (31): Node's COUNT 1
0:0:58.692382835 DEBUG (31): The new Count value is REJECTED.
0:0:58.692382835 DEBUG (31): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.717773458 DEBUG (5): Node's COUNT 1
0:0:58.717773458 DEBUG (5): The new Count value is REJECTED.
0:0:58.717773458 DEBUG (5): Tct: 15       Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.749023459 DEBUG (23): Node's COUNT 1
0:0:58.749023459 DEBUG (23): The new Count value is REJECTED.
0:0:58.749023459 DEBUG (23): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.800781270 DEBUG (4): Node's COUNT 1
0:0:58.800781270 DEBUG (4): The new Count value is REJECTED.
0:0:58.800781270 DEBUG (4): Tct: 15       Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.832031272 DEBUG (24): Node's COUNT 1
0:0:58.832031272 DEBUG (24): The new Count value is REJECTED.
0:0:58.832031272 DEBUG (24): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:0:58.846679709 DEBUG (16): ChildrenId: 11      Of: 16         COUNT: 1
0:0:58.846679709 DEBUG (16): Node's COUNT 2
0:0:58.846679709 DEBUG (16): The new Count value is REJECTED.
0:0:58.846679709 DEBUG (16): Tct: 15      Previous Count: 2      New Count: 2    Reject-Range:[1.700000,2.300000]
0:0:58.901367210 DEBUG (28): ChildrenId: 35      Of: 28         COUNT: 1
0:0:58.901367210 DEBUG (28): Node's COUNT 2
0:0:58.901367210 DEBUG (28): The new Count value is REJECTED.
0:0:58.901367210 DEBUG (28): Tct: 15      Previous Count: 2      New Count: 2    Reject-Range:[1.700000,2.300000]
0:0:58.931640647 DEBUG (27): ChildrenId: 34      Of: 27         COUNT: 1
0:0:58.931640647 DEBUG (27): Node's COUNT 2
0:0:58.931640647 DEBUG (27): The new Count value is REJECTED.
```

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



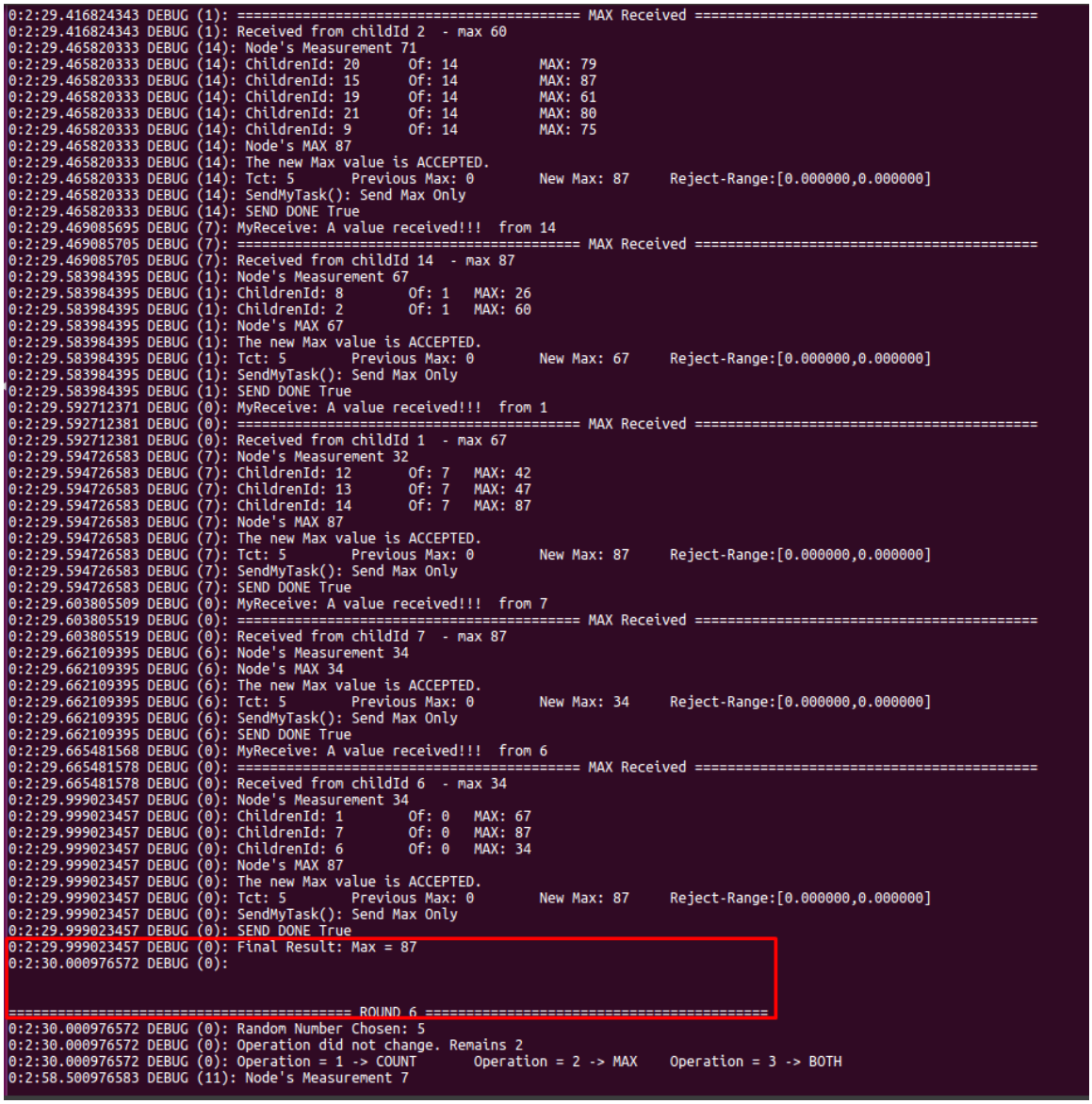
Δ.3.1 Στιγμιότυπο λειτουργίας της Περιόδου αλλαγής συναθροιστικής

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!!!. Το 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 operation

```
0:4:29.767578145 DEBUG (0): The new Max value is REJECTED.
0:4:29.767578145 DEBUG (0): Tct: 10       Previous Max: 100      New Max: 102    Reject-Range:[90.000000,110.000000]
0:4:29.767578145 DEBUG (0): Final Result: Max = 100
0:4:30.000000010 DEBUG (0):


======================================== ROUND 10 ========================================
0:4:30.000000010 DEBUG (0): Random Number Chosen: 0
0:4:30.000000010 DEBUG (0):
================================== Operation Update ==================================
0:4:30.000000010 DEBUG (0): Updated Operation To be executed: 1
0:4:30.000000010 DEBUG (0): Operation = 1 -> COUNT       Operation = 2 -> MAX    Operation = 3 -> BOTH
0:4:30.000000030 DEBUG (0): sendOpTask(): Send returned success!!!
0:4:30.504882832 DEBUG (1): sendOpTask(): Send returned success!!!
0:4:30.504882833 DEBUG (6): sendOpTask(): Send returned success!!!
0:4:31.012695333 DEBUG (7): sendOpTask(): Send returned success!!!
0:4:31.012695333 DEBUG (12): sendOpTask(): Send returned success!!!
0:4:31.013671895 DEBUG (2): sendOpTask(): Send returned success!!!
0:4:31.514648458 DEBUG (8): sendOpTask(): Send returned success!!!
0:4:31.514648458 DEBUG (13): sendOpTask(): Send returned success!!!
0:4:31.521484395 DEBUG (3): sendOpTask(): Send returned success!!!
0:4:31.522460959 DEBUG (18): sendOpTask(): Send returned success!!!
0:4:32.021484395 DEBUG (9): sendOpTask(): Send returned success!!!
0:4:32.021484396 DEBUG (14): sendOpTask(): Send returned success!!!
0:4:32.023437521 DEBUG (19): sendOpTask(): Send returned success!!!
0:4:32.024414082 DEBUG (4): sendOpTask(): Send returned success!!!
0:4:32.027343772 DEBUG (24): sendOpTask(): Send returned success!!!
0:4:32.526367209 DEBUG (15): sendOpTask(): Send returned success!!!
0:4:32.527343772 DEBUG (25): sendOpTask(): Send returned success!!!
0:4:32.528320333 DEBUG (5): sendOpTask(): Send returned success!!!
0:4:32.533203148 DEBUG (30): sendOpTask(): Send returned success!!!
0:4:33.031250021 DEBUG (11): sendOpTask(): Send returned success!!!
0:4:33.033203147 DEBUG (26): sendOpTask(): Send returned success!!!
0:4:33.033203148 DEBUG (31): sendOpTask(): Send returned success!!!
0:4:33.035156271 DEBUG (16): sendOpTask(): Send returned success!!!
0:4:33.035156272 DEBUG (21): sendOpTask(): Send returned success!!!
0:4:33.536132835 DEBUG (32): sendOpTask(): Send returned success!!!
0:4:33.537109397 DEBUG (22): sendOpTask(): Send returned success!!!
0:4:33.538085958 DEBUG (10): sendOpTask(): Send returned success!!!
0:4:33.538085959 DEBUG (17): sendOpTask(): Send returned success!!!
0:4:34.038085960 DEBUG (33): sendOpTask(): Send returned success!!!
0:4:34.041992209 DEBUG (23): sendOpTask(): Send returned success!!!
0:4:34.041992210 DEBUG (28): sendOpTask(): Send returned success!!!
0:4:34.543945335 DEBUG (27): sendOpTask(): Send returned success!!!
0:4:34.543945335 DEBUG (29): sendOpTask(): Send returned success!!!
0:4:34.543945335 DEBUG (34): sendOpTask(): Send returned success!!!
0:4:35.044921898 DEBUG (35): sendOpTask(): Send returned success!!!
0:4:57.287109398 DEBUG (35): Node's COUNT 1
0:4:57.287109398 DEBUG (35): The new Count value is ACCEPTED.
0:4:57.287109398 DEBUG (35): Tct: 10      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:4:57.287109398 DEBUG (35): SendMyTask(): Send Count Only
0:4:57.287109398 DEBUG (35): SEND DONE True
0:4:57.289245616 DEBUG (29): MyReceive: A value received!!!  from 35
0:4:57.289245626 DEBUG (29): ======================================== Count Received ========================================
0:4:57.289245626 DEBUG (29): Received from childId 35  - count 1
0:4:57.588867210 DEBUG (34): Node's COUNT 1
0:4:57.588867210 DEBUG (34): The new Count value is ACCEPTED.
0:4:57.588867210 DEBUG (34): Tct: 10      Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:4:57.588867210 DEBUG (34): SendMyTask(): Send Count Only
0:4:57.588867210 DEBUG (34): SEND DONE True
0:4:57.596221904 DEBUG (33): MyReceive: A value received!!!  from 34
0:4:57.596221914 DEBUG (33): ======================================== Count Received ========================================
0:4:57.596221914 DEBUG (33): Received from childId 34  - count 1
0:4:57.605468772 DEBUG (29): ChildrenId: 35      Of: 29          COUNT: 1
0:4:57.605468772 DEBUG (29): Node's COUNT 2
0:4:57.605468772 DEBUG (29): The new Count value is ACCEPTED.
```

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.

```
0:4:58.532226584 DEBUG (20): Node's Measurement 61
0:4:58.532226584 DEBUG (20): ChildrenId: 21      Of: 20          MAX: 6
0:4:58.532226584 DEBUG (20): Node's MAX 61
0:4:58.532226584 DEBUG (20): The new Max value is ACCEPTED.
0:4:58.532226584 DEBUG (20): Tct: 10      Previous Max: 68       New Max: 61     Reject-Range:[61.199997,74.800003]
0:4:58.532226584 DEBUG (20): SendMyTask(): Send Max Only
0:4:58.532226584 DEBUG (20): SEND DONE True
0:4:58.539993262 DEBUG (19): MyReceive: A value received!!!  from 20
0:4:58.539993272 DEBUG (19): ======================================== Count Received ========================================
0:4:58.539993272 DEBUG (19): Received from childId 20  - count 61
```

Δ.4.2 Στιγμιότυπο Παρουσίασης εσφαλμένης αποστολής του κόμβου 20

Έτσι καταλήγουμε τελικά στο προφανώς εσφαλμένο αποτέλεσμα:

```
0:4:59.666793798 DEBUG (0): ======================================== Count Received ========================================
0:4:59.666793798 DEBUG (0): Received from childId 6  - count 77
0:4:59.767578145 DEBUG (0): ChildrenId: 1        Of: 0   COUNT: 8
0:4:59.767578145 DEBUG (0): ChildrenId: 6        Of: 0   COUNT: 77
0:4:59.767578145 DEBUG (0): Node's COUNT 86
0:4:59.767578145 DEBUG (0): The new Count value is ACCEPTED.
0:4:59.767578145 DEBUG (0): Tct: 10       Previous Count: 0      New Count: 86   Reject-Range:[0.000000,0.000000]
0:4:59.767578145 DEBUG (0): SendMyTask(): Send Count Only
0:4:59.767578145 DEBUG (0): SEND DONE True
0:4:59.767578145 DEBUG (0): Final Result: Count = 86
0:5:0.000000010 DEBUG (0):
```

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:

```
):14:59.646484395 DEBUG (7): Tct: 15      Previous Count: 30     New Count: 30   Reject-Range:[25.500000,34.500000]
):14:59.646484395 DEBUG (7): Node's Measurement 1
):14:59.646484395 DEBUG (7): ChildrenId: 14      Of: 7   MAX: 70
):14:59.646484395 DEBUG (7): ChildrenId: 2       Of: 7   MAX: 1
):14:59.646484395 DEBUG (7): ChildrenId: 13      Of: 7   MAX: 21
):14:59.646484395 DEBUG (7): ChildrenId: 8       Of: 7   MAX: 0
):14:59.646484395 DEBUG (7): ChildrenId: 12      Of: 7   MAX: 1
):14:59.646484395 DEBUG (7): Node's MAX 70
):14:59.646484395 DEBUG (7): The new Max value is REJECTED.
):14:59.646484395 DEBUG (7): Tct: 15      Previous Max: 70       New Max: 70     Reject-Range:[59.500000,80.500000]
):14:59.894531270 DEBUG (0): ChildrenId: 1       Of: 0   COUNT: 1
):14:59.894531270 DEBUG (0): ChildrenId: 6       Of: 0   COUNT: 1
):14:59.894531270 DEBUG (0): ChildrenId: 7       Of: 0   COUNT: 30
):14:59.894531270 DEBUG (0): Node's COUNT 33
):14:59.894531270 DEBUG (0): The new Count value is REJECTED.
):14:59.894531270 DEBUG (0): Tct: 15      Previous Count: 33     New Count: 33   Reject-Range:[28.050001,37.950001]
):14:59.894531270 DEBUG (0): Node's Measurement 57
):14:59.894531270 DEBUG (0): ChildrenId: 1       Of: 0   MAX: 62
):14:59.894531270 DEBUG (0): ChildrenId: 6       Of: 0   MAX: 103
):14:59.894531270 DEBUG (0): ChildrenId: 7       Of: 0   MAX: 70
):14:59.894531270 DEBUG (0): Node's MAX 103
):14:59.894531270 DEBUG (0): The new Max value is REJECTED.
):14:59.894531270 DEBUG (0): Tct: 15      Previous Max: 90       New Max: 103    Reject-Range:[76.500000,103.500000]
):14:59.894531270 DEBUG (0): Final Result: Count = 33
):14:59.894531270 DEBUG (0): Final Result: Max = 90
```

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.

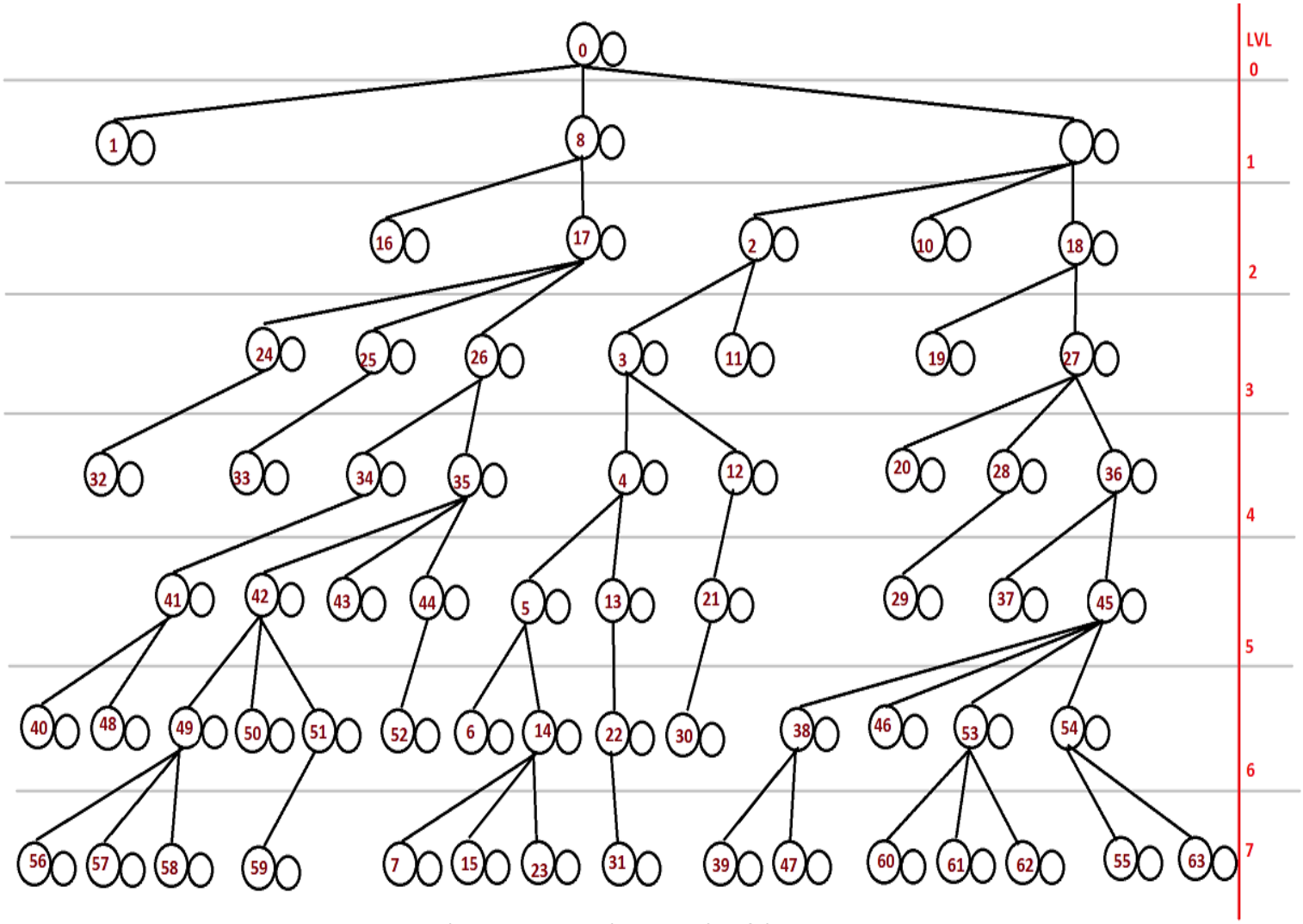
Δ.6 Συμπληρωματικό Παράδειγμα: Επίδειξη λειτουργικότητας Φάσης Α μετά από αποσφαλμάτωση σε δίκτυο 8x8 και range 1.5 απουσία θορύβου

Πραγματοποιώντας του Routing Προκύπτει το ακόλουθο αποτέλεσμα:

```
0:0:14.000000010 DEBUG (0): Routing was finished!!!      My depth: 0     My parent: 0
0:0:14.000000010 DEBUG (1): Routing was finished!!!      My depth: 1     My parent: 0
0:0:14.000000010 DEBUG (2): Routing was finished!!!      My depth: 2     My parent: 9
0:0:14.000000010 DEBUG (3): Routing was finished!!!      My depth: 3     My parent: 2
0:0:14.000000010 DEBUG (4): Routing was finished!!!      My depth: 4     My parent: 3
0:0:14.000000010 DEBUG (5): Routing was finished!!!      My depth: 5     My parent: 4
0:0:14.000000010 DEBUG (6): Routing was finished!!!      My depth: 6     My parent: 5
0:0:14.000000010 DEBUG (7): Routing was finished!!!      My depth: 7     My parent: 14
0:0:14.000000010 DEBUG (8): Routing was finished!!!      My depth: 1     My parent: 0
0:0:14.000000010 DEBUG (9): Routing was finished!!!      My depth: 1     My parent: 0
0:0:14.000000011 DEBUG (10): Routing was finished!!!     My depth: 2     My parent: 9
0:0:14.000000011 DEBUG (11): Routing was finished!!!     My depth: 3     My parent: 2
0:0:14.000000011 DEBUG (12): Routing was finished!!!     My depth: 4     My parent: 3
0:0:14.000000011 DEBUG (13): Routing was finished!!!     My depth: 5     My parent: 4
0:0:14.000000011 DEBUG (14): Routing was finished!!!     My depth: 6     My parent: 5
0:0:14.000000011 DEBUG (15): Routing was finished!!!     My depth: 7     My parent: 14
0:0:14.000000011 DEBUG (16): Routing was finished!!!     My depth: 2     My parent: 8
0:0:14.000000011 DEBUG (17): Routing was finished!!!     My depth: 2     My parent: 8
0:0:14.000000011 DEBUG (18): Routing was finished!!!     My depth: 2     My parent: 9
0:0:14.000000011 DEBUG (19): Routing was finished!!!     My depth: 3     My parent: 18
0:0:14.000000012 DEBUG (20): Routing was finished!!!     My depth: 4     My parent: 27
0:0:14.000000012 DEBUG (21): Routing was finished!!!     My depth: 5     My parent: 12
0:0:14.000000012 DEBUG (22): Routing was finished!!!     My depth: 6     My parent: 13
0:0:14.000000012 DEBUG (23): Routing was finished!!!     My depth: 7     My parent: 14
0:0:14.000000012 DEBUG (24): Routing was finished!!!     My depth: 3     My parent: 17
0:0:14.000000012 DEBUG (25): Routing was finished!!!     My depth: 3     My parent: 17
0:0:14.000000012 DEBUG (26): Routing was finished!!!     My depth: 3     My parent: 17
0:0:14.000000012 DEBUG (27): Routing was finished!!!     My depth: 3     My parent: 18
0:0:14.000000012 DEBUG (28): Routing was finished!!!     My depth: 4     My parent: 27
0:0:14.000000012 DEBUG (29): Routing was finished!!!     My depth: 5     My parent: 28
0:0:14.000000013 DEBUG (30): Routing was finished!!!     My depth: 6     My parent: 21
0:0:14.000000013 DEBUG (31): Routing was finished!!!     My depth: 7     My parent: 22
0:0:14.000000013 DEBUG (32): Routing was finished!!!     My depth: 4     My parent: 24
0:0:14.000000013 DEBUG (33): Routing was finished!!!     My depth: 4     My parent: 25
0:0:14.000000013 DEBUG (34): Routing was finished!!!     My depth: 4     My parent: 26
0:0:14.000000013 DEBUG (35): Routing was finished!!!     My depth: 4     My parent: 26
0:0:14.000000013 DEBUG (36): Routing was finished!!!     My depth: 4     My parent: 27
0:0:14.000000013 DEBUG (37): Routing was finished!!!     My depth: 5     My parent: 36
0:0:14.000000013 DEBUG (38): Routing was finished!!!     My depth: 6     My parent: 45
0:0:14.000000013 DEBUG (39): Routing was finished!!!     My depth: 7     My parent: 38
0:0:14.000000014 DEBUG (40): Routing was finished!!!     My depth: 6     My parent: 41
0:0:14.000000014 DEBUG (41): Routing was finished!!!     My depth: 5     My parent: 34
0:0:14.000000014 DEBUG (42): Routing was finished!!!     My depth: 5     My parent: 35
0:0:14.000000014 DEBUG (43): Routing was finished!!!     My depth: 5     My parent: 35
0:0:14.000000014 DEBUG (44): Routing was finished!!!     My depth: 5     My parent: 35
0:0:14.000000014 DEBUG (45): Routing was finished!!!     My depth: 5     My parent: 36
0:0:14.000000014 DEBUG (46): Routing was finished!!!     My depth: 6     My parent: 45
0:0:14.000000014 DEBUG (47): Routing was finished!!!     My depth: 7     My parent: 38
0:0:14.000000014 DEBUG (48): Routing was finished!!!     My depth: 6     My parent: 41
0:0:14.000000014 DEBUG (49): Routing was finished!!!     My depth: 6     My parent: 42
0:0:14.000000015 DEBUG (50): Routing was finished!!!     My depth: 6     My parent: 42
0:0:14.000000015 DEBUG (51): Routing was finished!!!     My depth: 6     My parent: 42
0:0:14.000000015 DEBUG (52): Routing was finished!!!     My depth: 6     My parent: 44
0:0:14.000000015 DEBUG (53): Routing was finished!!!     My depth: 6     My parent: 45
0:0:14.000000015 DEBUG (54): Routing was finished!!!     My depth: 6     My parent: 45
0:0:14.000000015 DEBUG (55): Routing was finished!!!     My depth: 7     My parent: 54
0:0:14.000000015 DEBUG (56): Routing was finished!!!     My depth: 7     My parent: 49
0:0:14.000000015 DEBUG (57): Routing was finished!!!     My depth: 7     My parent: 49
0:0:14.000000015 DEBUG (58): Routing was finished!!!     My depth: 7     My parent: 49
0:0:14.000000015 DEBUG (59): Routing was finished!!!     My depth: 7     My parent: 51
0:0:14.000000016 DEBUG (60): Routing was finished!!!     My depth: 7     My parent: 53
0:0:14.000000016 DEBUG (61): Routing was finished!!!     My depth: 7     My parent: 53
0:0:14.000000016 DEBUG (62): Routing was finished!!!     My depth: 7     My parent: 53
0:0:14.000000016 DEBUG (63): Routing was finished!!!     My depth: 7     My parent: 54
```

Δ.6.1 Στιγμιότυπο Παρουσίασης 8x8 grid Routing

Και το δέντρο που κατασκευάζεται ακολουθώντας το παραπάνω είναι το εξής:



Δ.6.1 Δέντρο που προκύπτει ακολουθώντας το Routing

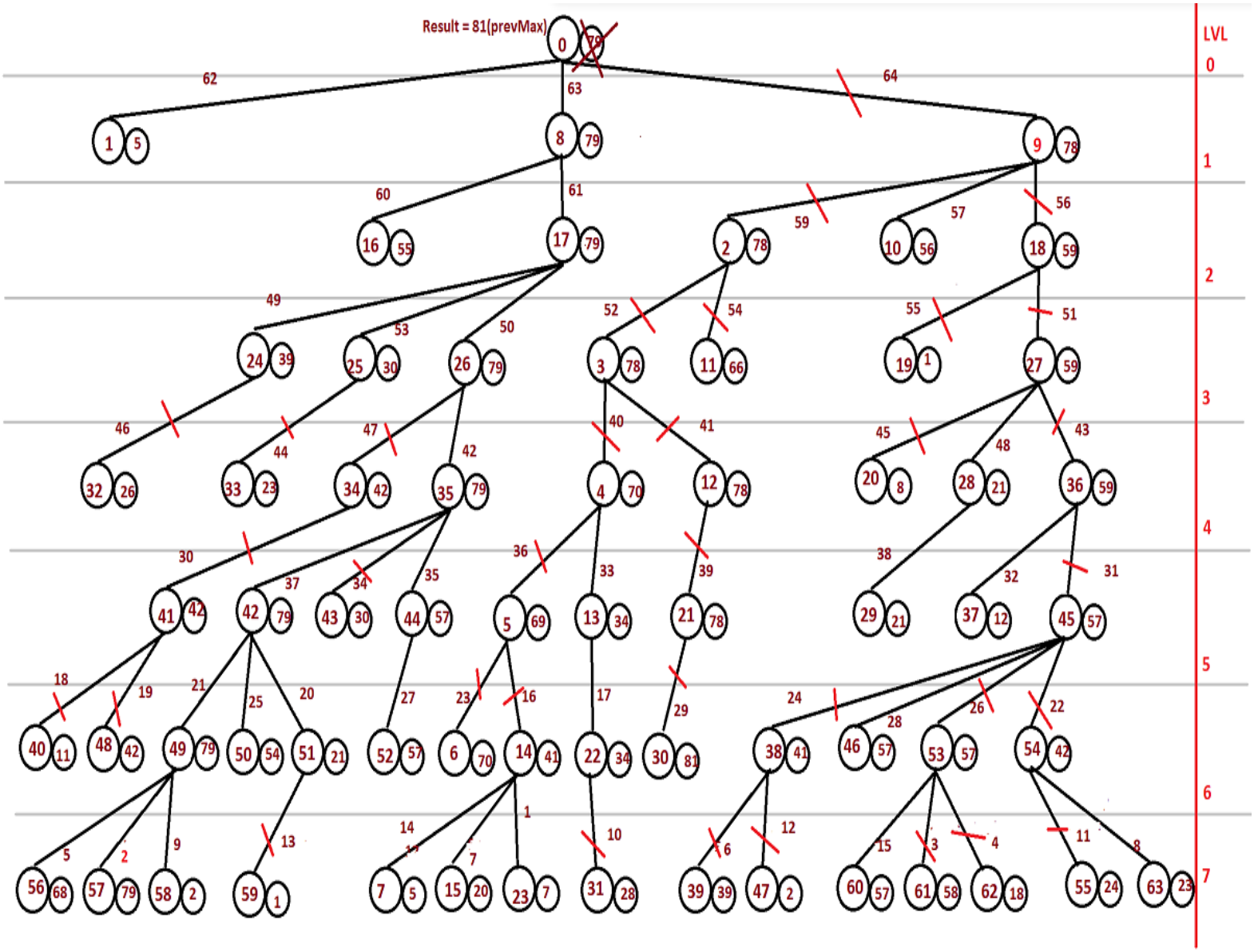
Στο τέλος λοιπόν της πρώτης περιόδου προκύπτει για τον υπολογισμό του count:

```
0:0:29.029296895 DEBUG (1): The new Count value is ACCEPTED.
0:0:29.029296895 DEBUG (1): Tct: 5    Previous Count: 0  New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:29.029296895 DEBUG (1): SendMyTask(): Send Count Only
0:0:29.029296895 DEBUG (1): SEND DONE True
0:0:29.031875615 DEBUG (0): MyReceive: A value received!!!  from 1
0:0:29.031875625 DEBUG (0): ======================================== Count Received ========================================
0:0:29.031875625 DEBUG (0): Received from childId 1  - count 1
0:0:29.074218770 DEBUG (8): ChildrenId: 16   Of: 8   COUNT: 1
0:0:29.074218770 DEBUG (8): ChildrenId: 17   Of: 8   COUNT: 22
0:0:29.074218770 DEBUG (8): Node's COUNT 24
0:0:29.074218770 DEBUG (8): The new Count value is ACCEPTED.
0:0:29.074218770 DEBUG (8): Tct: 5    Previous Count: 0  New Count: 24   Reject-Range:[0.000000,0.000000]
0:0:29.074218770 DEBUG (8): SendMyTask(): Send Count Only
0:0:29.074218770 DEBUG (8): SEND DONE True
0:0:29.084854084 DEBUG (0): MyReceive: A value received!!!  from 8
0:0:29.084854094 DEBUG (0): ======================================== Count Received ========================================
0:0:29.084854094 DEBUG (0): Received from childId 8  - count 24
0:0:29.200195333 DEBUG (9): ChildrenId: 18   Of: 9   COUNT: 20
0:0:29.200195333 DEBUG (9): ChildrenId: 10   Of: 9   COUNT: 1
0:0:29.200195333 DEBUG (9): ChildrenId: 2    Of: 9   COUNT: 16
0:0:29.200195333 DEBUG (9): Node's COUNT 38
0:0:29.200195333 DEBUG (9): The new Count value is ACCEPTED.
0:0:29.200195333 DEBUG (9): Tct: 5    Previous Count: 0  New Count: 38   Reject-Range:[0.000000,0.000000]
0:0:29.200195333 DEBUG (9): SendMyTask(): Send Count Only
0:0:29.200195333 DEBUG (9): SEND DONE True
0:0:29.204208371 DEBUG (0): MyReceive: A value received!!!  from 9
0:0:29.204208381 DEBUG (0): ======================================== Count Received ========================================
0:0:29.204208381 DEBUG (0): Received from childId 9  - count 38
0:0:29.538085957 DEBUG (0): ChildrenId: 1    Of: 0   COUNT: 1
0:0:29.538085957 DEBUG (0): ChildrenId: 8    Of: 0   COUNT: 24
0:0:29.538085957 DEBUG (0): ChildrenId: 9    Of: 0   COUNT: 38
0:0:29.538085957 DEBUG (0): Node's COUNT 64
0:0:29.538085957 DEBUG (0): The new Count value is ACCEPTED.
0:0:29.538085957 DEBUG (0): Tct: 5    Previous Count: 0  New Count: 64   Reject-Range:[0.000000,0.000000]
0:0:29.538085957 DEBUG (0): SendMyTask(): Send Count Only
0:0:29.538085957 DEBUG (0): SEND DONE True
0:0:29.538085957 DEBUG (0): Final Result: Count = 64
0:0:30.000000010 DEBUG (0):
```

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.