ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Η/Υ

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ / TECHNICAL UNIVERSITY OF CRETE

ΠΛΗ 511 - Επεξεργασία και Διαχείριση Δεδομένων σε Δίκτυα Αισθητήρων

Μέρος 1ο

Φοιτητές :

Σκλάβος Παναγιώτης 2018030170

Χρυσής Επαμεινώνδας 2018030167

Διδάσκων :

Δεληγιαννάκης Αντώνιος

Περιεχόμενα

Α. Εισαγωγή

Το πρώτο μέρος της εργασίας του μαθήματος αποτελεί μια γνωριμία με το περιβάλλον του tinyOS καθώς και αποτελεί μια ευκαιρία εξοικείωσης με τον χρονοπρογραμματισμό αισθητήρων μέσα σε ένα δίκτυο. Συγκεκριμένα το έργο μας κατά την εκπόνηση της εργασίας αυτής ήταν τριμελές. Σε πρώτη φάση οφείλαμε να κατανοήσουμε τον βοηθητικό κώδικα που μας δόθηκε εντός του .vdi δίσκου και με βάση την κατανόηση του, να αφαιρέσουμε όσα τμήματα δεν κρίνονται αναγκαία για την υλοποίησή μας. Έπειτα χρήσιμη κρίθηκε η υλοποίηση ενός βοηθητικού προγράμματος το οποίο αυτοματοποιεί την διαδικασία δημιουργίας αρχείων topology ώστε να μπορεί ο κώδικάς μας να ελεγχθεί σε διαφορετικές διαρρυθμίσεις δικτύων. Τέλος, εφόσον είχαν πραγματοποιηθεί αμφότερες οι διαδικασίες, τελικό ζητούμενο ήταν η υλοποίηση δυο συναθροιστηκών συναρτήσεων καθώς και ο συνδυασμός αυτών, με σκοπό την αποτελεσματική αξιοποίηση του δικτύου ώστε να τηρείται συγχρονισμός ανά επίπεδο(με βάσει το TiNA), καθώς και να μεταδίδεται βέλτιστα η ζητούμενη πληροφορία. Για να είμαστε περισσότερο ακριβείς το πρωτόκολλο που υλοποιήθηκε αποτελεί μια βελτιωμένη λογική του TiNA, όπου το tct εφαρμόζεται σε όλους τους κόμβους και όχι μόνο στα φύλλα.

Β. Βοηθητικό πρόγραμμα

Για την δημιουργία του βοηθητικού προγράμματος, που παράγει ένα τυχαίο δίκτυο αισθητήρων, ακολουθήθηκαν τα βήματα της εκφώνησης και υλοποιήθηκε το αρχείο topology_gen.py , το οποίο λαμβάνει ως ορίσματα την διάμετρο(D) και την εμβέλεια (R) των κόμβων. Κατά τον τρόπο αυτό παράγεται ενα grid μεγέθους D*D πάνω στο οποίο εντοπίζονται γειτονικοί αισθητήρες συγκρίνοντας την γεωμετρική τους απόσταση με την εμβέλεια τους, Οι αισθητήρες λοιπόν, των οποίων η εμβέλεια καλύπτει άλλους αισθητήρες εισάγονται στο αρχείο ως γειτονικοί.

Με τον τρόπο αυτό λοιπόν, παράγεται ένα νέο αρχείο topology στο οποίο αποτυπώνονται τα αποτελέσματα του προγράμματος με μορφή αντίστοιχη του δοθέντος:

0 1 -50.

1 0 -50.0

0 2 -50.0

2 0 -50.0

0 3 -50.0

3 0 -50.0

...

Γ. Κατανόηση και προσαρμογή αρχικού κώδικα

Σε πρώτη φάση θα αναλυθεί η συλλογιστική που ακολουθήθηκε για την «εκκαθάριση» του δοθέντος κώδικα από τα τμήματα που δεν εξυπηρετούσαν το πρωτόκολλο TiNa. Η εργασία μας επικεντρώθηκε στο αρχείο *SRTreeC.nc* το οποίο αποτελεί το κύριο module της εργασίας, ενώ για την εφαρμογή νέων wirings συμπληρώθηκε το configuration αρχείο SRTeeApp.nc. Ακόμη τοποθετήθηκαν νέες σταθερές και structs για την μεταφορά μηνυμάτων στο header file: SimpleRoutingTree.h.

Ξεκινώντας λοιπόν, αφαιρέθηκαν τα ψήγματα κώδικα που ασχολούνταν με την λειτουργία των Leds. Σε κανένα κομμάτι της εργασίας δεν μας ζητείται να πραγματοποιήσουμε κάποια λειτουργία με αυτά, συνεπώς αποτελούσε κώδικα χωρίς λειτουργικότητα. Με παρόμοιο σκεπτικό αφαιρέθηκαν τα τμήματα κώδικα που αφορούσαν την σειριακή θύρα, καθώς και ο κώδικας που εμπεριεχόταν μεταξύ των blocks #ifdef SERIAL_EN και #endif.

Συνεχίζοντας, αφαιρέθηκαν κομμάτια κώδικα που διαχειρίζονταν το ενδεχόμενο να να χαθεί ένα task, όταν, πρωτού ολοκληρωθεί η διαδικασία αποστολής του , λαμβάναμε νέο task για αποστολή. Το Tina δεν είναι σχεδιασμένο ώστε να διαχειρίζεται τέτοιες καταστάσεις, συνεπώς, κατά τον τρόπο αυτό έχασαν λόγο ύπαρξης ο μετρητής LostTaskTimer καθώς και οι συναρτήσεις set Lost Task και Send Busy.

Στο κομμάτι που αφορούσε την διαδικασία του Routing των κόμβων και την δημιουργία του δένδρου, παρατηρήθηκε ότι το Routing συνέβαινε σε κάθε εποχή περιοδικά , γεγόνος που δεν έπρεπε να υλοποιείται με αυτόν τον τρόπο αλλά να γίνεται μόνο κατά την πρώτη περίοδο, οπότε αφαιρέθηκε η περιοδική κλήση του μετρητή του κόμβου 0, δηλαδή η εντολή call RoutingMsgTimer.startOneShot(TIMER_PERIOD_MILLI);

Από RoutingMsg, ο ορισμός του οποίου παρατίθεται στο header file, αφαιρέθηκε το πεδίο SenderID καθώς αυτή η πληροφορία υπάρχει στον header του μηνύματος και είναι εύκολα προβιβάσιμη απο την AMPacket.source(&RoutingMsg). Με τον τρόπο αυτό επιτυγχάνεται ταυτόχρονο ο περιορισμός του μεγέθους του RoutingMsg, το οποίο μπορεί να μην έχει ιδιαίτερα μεγάλο κόστος καθώς αποστέλλεται μόνο στην αρχή, ωστόσο ποτέ δεν είναι κακό να περιορίζεται ο όγκος της απεσταλμένης πληροφορίας αν αυτό δεν διακινδυνεύει την λειτουργικότητα.

Επιπλέον αφαιρέθηκε όλη η λογική του NotifyParent (τύπος– δομή μηνύματος, Timer, NotifyReceive, NotifySend) καθώς έγινε αντιληπτό ότι κατά το Routing του αρχικού προγράμματος, οι κόμβοι που λάμβαναν ένα RoutingMsg, επέστρεφαν μια επιβεβαίωση στον πατέρα τους γι’ αυτό, γεγονός που αντιτίθεται στην λογική του TiNA. Αναλυτικότερα, βασιζόμενοι σε ΤιΝΑ λογική, οι κόμβοι , δεν έχουν παρά να θέσουν ως πατέρα τους εκείνον από τον οποίο θα λάβουν το RoutingMsg. Εκείνος με την σειρά του ενημερώνεται για την σύνδεση με τα παιδία του όταν κάποια στιγμή αρχίσει να λαμβάνει απαντήσεις στα queries που τους έχει παραθέσει.

Εφαρμόζοντας λοιπόν τις άνωθεν τροποποιήσεις ο εναπομένων κώδικας είναι σύμφωνος με τα ζητούμενα και η διαδικασία του Routing έχει ρυθμιστεί σύμφωνα με την επιθυμητή λειτουργία. Το ενδιαφέρον στρέφεται τώρα, στην λήψη μετρήσεων και τη δρομολόγηση τους προς τη ρίζα υπολογίζοντας τις συναθροιστικές συναρτήσεις MAX και COUNT.

Δ. Επεξήγηση Κώδικα

Στο τμήμα αυτό θα επεξηγηθούν οι αλλαγές και οι προσθήκες που πραγματοποιήθηκαν ώστε να επιτευχθεί η επιθυμητή Λειτουργία του Tina και η μετάδοση των αποτελεσμάτων των συναθροιστικών συναρτήσων.

Δ1. Custom Timers

Σε πρώτη φάση δημιουργήθηκε ένας μετρητής RoundTimer ο οποίος χτυπάει με περίοδο TIMER_PERIOD_MILLI = 30*1024 ms. Ο μετρητής αυτός σηματοδοτεί την αρχή μιας νέας περιόδου και κάνει increment την global μεταβλητή roundCounter. Επιπλέον στο event RoundTimer.fired() τυπώνεται ένα βοηθητικό μήνυμα το οποία παρουσιάζει τον αριθμό τρέχοντος γύρου.

Δημιουργήθηκε επίσης ένας δεύτερος timer με ονομασία MyTimer ο οποίος ξεκινάει και χτυπάει πρώτη φορά μετά από ROUTING_TIME το οποίο ανατίθεται αρκετά μεγάλο ώστε να προλάβει να ολοκληρωθεί το routing σε όλο το δένδρο ώστε να αναγνωρίσει κάθε κόμβος το βάθος και τον πατέρα του. Ενδεικτικά η τιμή της σταθεράς είναι ορισμένη ως $ROUTING\_TIME = 3 * 1024\, ms$. Το γεγονός αυτό παρουσιάζεται μέσα στον event *RadioControl.startDone* με την κλήση της εντολής έναρξης του μετρητή για μια φορά: $MyTimer.startOneShot(TIMER\_ROUTING\_TIME)$. Έπειτα όταν ενεργοποιηθεί για πρώτη φορά. Ο μετρητής αρχίζει να χτυπάει περιοδικά, κάθε φορά σηματοδοτώντας την νέα λήψη μέτρησης. Σηγκεκριμένα η εντολή αυτή παρουσιάζεται στο event $MyTimer.fired()$ ως ακολούθως:

$$call\ MyTimer.startPeriodicAt((-(BOOT\_TIME) - \big((curdepth + 1) * TIMER\_FAST\_PERIOD\big) + (same\_level\_variator)), TIMER\_PERIOD\_MILLI)$$

Επεξηγώντας τα παραπάνω ορίσματα έχουμε: Το πρώτο όρισμα επιδεικνύει το πότε θα αρχίσει την μέτρηση ο μετρητής και απαιτεί σημαντική ανάλυση για την κατανόηση της λειτουργίας του μετρητή. Για τον λόγο αυτό θα επεξηγηθεί κάθε όρισμα ξεχωριστά.

Στην φάση αυτή αξίζει να επισημανθεί το γεγονός ότι η χρονική αφετηρία του μετρητή είναι η στιγμή που θα γίνει booted ο αισθητήρας. Ο χρόνος αυτός αντιστοιχίζεται στον χρόνο BOOT_TIME ο οποίος παρουσιάζεται και στο αρχείο *mySimulation.py* ως την χρονική στιγμή της προσομοίωσης που οι αισθητήρες θα εκκινήσουν την λειτουργία τους. Ο χρόνος αυτός ισούται με: $BOOT\_TIME = 10 * 1024\, ms$. Εύκολα λοιπόν γίνεται αντιληπτό ότι το timeline που αντιλαμβάνεται ο αισθητήρας ,διαφέρει από εκείνο της προσομοίωσης καθώς ο αισθητήρας δεν μπορεί να αντιληφθεί τα πρώτα 10 περίπου seconds ώσπου να κάνει boot. Συνεπώς αφαιρούμε *BOOT_TIME* ώστε να μετατοπίσουμε τον χρόνο στην αρχή της προσομοίωσης.

Συνεχίζοντας ακολουθεί το κομμάτι –TIMER_FAST_PERIOD*(curdepth+1) το οποίο σηματοδοτεί το διαφορετικό timeslice που έχει αποδοθεί σε κάθε επίπεδο του δέντρου. Συγκεκριμένα με τον τρόπο αυτό διαχωρίζομε ένα timeslice πλάτους TIMER_FAST_PERIOD = 256 ms για κάθε επίπεδο του δέντρου και ο παράγοντας “-(curdepth+1)” είναι για να επιβεβαιώσει ότι κόμβοι με μεγαλύτερο βάθος ξεκινάνε να μεταδίδουν στο time slice τους νωρίτερα από

εκείνους με μικρότερο βάθος. Το γεγονός αυτό όμως τοποθετεί τους μετρητές να χτυπάνε σε χρόνο πριν την αρχή του simulation για κάθε φορά εγείροντας το ερώτημα αν χάνεται η πρώτη περίοδος. Η απάντηση είναι πως όχι, καθώς, λόγω περιοδικότητας ο χρόνος αυτός μετατοπίζεται στο τέλος της πρώτης περιόδου. Οπότε ανακεφαλαιώνοντας, το τελευταίο time slice που θα μεταδώσει (για caurdepth=0) θα τοποθείται να τελειώνει στην αρχή της επόμενης περιόδου καθώς θα εκτελείται πρώτη φορά σε χρόνο TIMER_PERIOD_MILLI-TIMER_FAST_PERIOD.

Το μόνο πρόβλημα που απομένει να επιλυθεί είναι ότι με τον τρόπο αυτό όλοι οι κόμβοι του ίδιου επιπέδου θα αρχίζουν να μεταδίδουν στην αρχή timeslice τους δημιουργώντας έτσι πολλαπλές συγκρούσεις που θα έχουν ως άμεσο αντίκτυπο την απώλεια πληροφορίας. Στην επίλυση το προβλήματος αυτού έρχεται να συνδράμει η μεταβλητή same_level_variator. Δύο πράγματα είναι σημαντικό να σχολιαστούν για να κατανοήσουμε την λειτουργία της μεταβλητής.

Αφενός το πρόσημό της και αφετέρου την τιμή της. Σε πρώτη φάση αν παρατηρήσουμε το πρόσημο αντιλαμβανόμαστε ότι οι αισθητήρες θα αποκτούν μια διακύμανση προς την θετική κατεύθυνση του άξονα. Αυτό συμβαίνει διότι αν εξωθούνταν προς τα αριστερά θα μεταπηδούσαν για αρνητικές τιμές του variator στο προηγούμενο timeslice μιας και η αφετηρία αποστολής είναι η αρχή του timeslice. Με τον τρόπο αυτό θα γινόταν μια μάταιη μετατόπιση σε γειτονικό timeslice. Αντιθέτως ξεκινώντας από την αρχή του δικού τους slot και κινούμενοι προς τα δεξιά επιτυγχάνεται να αποφεύγονται μεταπηδήσεις στα timeslices των άλλων επιπέδων εάν η τιμή του variator είναι μικρότερη ή ίση με TIMER_FAST_PERIOD.

Συνεπώς, ως επέκταση των όσων είπαμε παραπάνω καταλήγουμε ότι το πρόσημο του variator θα πρέπει να είναι θετικό και η τιμή του μικρότερη από το μέγεθος του timeSlice ώστε να μην συγχέονται τα slices μεταξύ τους. Για τον λόγο αυτό ανατίθεται στον variator μια τυχαία τιμή για κάθε κόμβο του ίδιου επιπέδου η οποία φράσσεται από το TIMER_FAST_PERIOD ως εξής: $same\_level\_variator = (call\ Random.rand16())\%(TIMER\_FAST\_PERIOD)$. Έτσι, για να επιτευχθεί κάποια σύγκρουση μεταξύ των κόμβων θα πρέπει τελικά να καταλήξουν στον ίδιο variator πιθανότητα, που είναι πολύ μικρή.

Δ.2 Λήψη Μετρήσεων

Εφόσον ρυθμίστηκαν κατάλληλα οι Timers την εφαρμογής, στην φάση αυτή θα επεξηγηθεί ο τρόπος με τον οποίο έγινε η λήψη των μετρήσεων για τον υπολογισμό των συναθροιστικών συναρτήσεων.

Αρχικά σε πρώτη φάση η συναθροιστική συνάρτηση που θα υπολογίζεται κατά το simulation επιλέγεται από τον σταθμό βάσης όταν χτυπήσει ο Timer για αποστολή RoutingMsg. Η Επιλογή γίνεται με τυχαίο τρόπο μεταξύ 3 αριθμών. Καθένας από τους αριθμούς έχει οριστέι ως enum στο header αρχείο ως ακολούθως:

- $COUNT = 1$
- $MAX = 1$
- $BOTH = 1$

Με τον τρόπο αυτό καταχωρείται σε όσους αισθητήρες λαμβάνον το RoutingMsg και επαναλαμβάνουν την αποστολή της ζητούμενης μέτρησης κάθε περίοδο όταν αυτό επιτρέπεται από το tct του TiNa. Η τιμή του Tct υπολογίζεται με αντίστοιχο τρόπο στον σταθμό βάσης και αποστέλλεται κι αυτή με το RoutingMsg.

Κάθε αισθητήρας λαμβάνει μια τιμή όταν χτυπήσει ο MyTimer για να την κάνει send εντός του event *MyTimer.fired*(). Η τιμή στην πρώτη περίοδο παίρνει μια τυχαία τιμή από 1 έως 80 ενώ τις επόμενες λαμβάνει μια τυχαία τιμή στο εύρος:

$(1 - random\_step\_size) * value$ έως $(1 + random\_step\_size) * value$

όπου η τιμή του $random\_step\_size$ δίνεται από την εκφώνηση.

Για την αποστολή των τιμών των συναθροισιτκών μετρήσεων Count και Max έχει γίνει υλοποίηση 2 structs στο header αρχείο με όνομα Measurement {uint8_t measurement} και TwoMeasurements(uint8_t count, uint8_t max) αντίστοιχα.

Δ.3 Βελτιωμένη λογική TiNa με ελαχιστοποίηση μεταδιδόμενης πληροφορίας

Για την υλοποίηση του Tina δημιουργήθηκαν τα ακόλουθα:

- Event MyAMSend.sendDone
- Event MyReceive.receive
- Ουρά MySendQueue
- Ουρά MyReceiveQueue
- Task sendMyTask
- Task receiveMyTask
- Internal functions: count_calculation, max_calculation, children_init

Η περιγραφή των παραπάνω θα γίνει συγκεντρωτικά καθώς έχουν συμπληρωματική λειτουργία. Για να γίνει μια ολοκληρωμένη περιγραφή οφείλουμε να θέσουμε ώς αφετηρία το event *MyTimer.fired(),* καθώς εκεί λαμβάνεται η μέτρηση του αισθητήρα η οποία γίνεται wrap σε ένα AMPacket και τοποθετείται στην ουρά mySendQueue. Έπειτα γίνεται post το task sendMyTask() το οποίο θα διαχειριστεί την τιμή του αισθητήρα έως και την αποστολή του. Φθάνοντας στο sendMyTask() εάν η ουρά αποστολής δεν είναι άδεια, αφαιρούμε ένα στοιχείο μέτρησης και ελέηγχουμε το operation που πρόκειται να εκτελεστεί το οποίο έχει ληφθεί και αποθηκευτεί από το receiceRoutingMsg(). Έπειτα για αποφυγή επανάληψης κώδικας δημιοργούνται 4 flags: countOp(eration) & countChanged και αντίστοιχα για το max. Αναλόγως με το operation που έχει ληφθεί στο routing η countOp και maxOp μεταβλητές γίνονται set. Έπειτα γίνονται οι υπολογισμοί του count και του max με τις ομώνυμες συναρτήσεις. Αν το αποτέλεσμα τους ξεπερνάει το όριο του tct τότε θέτουμε True την κατάλληλη μεταβλητή changed.

Στην φάση αυτή ελέγχεται ποίες μεταβλητές μεταβλήθηκαν αρκετά ώστε να ξεπεράσουν το tct και αναλόγως διακρίνονται 3 περιπτώσεις σύμφωνα με την αποστολή τους:

- Απαιτείται αποστολή αμφότερων: Τότε συμπληρώνονται τα πεδία του TwoMeasurment και πραγματοποιείται η αποστολή.

- Απαιτείται αποστολή μόνο του max: ελέγχεται λοιπόν αν το operation ήταν BOTH ή MAX. Αν είναι MAX query γίνεται αποστολή του struct Measurement περιέχοντας το MAX. Διαφορετικά, εκμεταλλευόμαστε την εξής παρατήρηση: Οι τιμές που μπορούμε να λάβουμε δεν ξεπερνούν το $2^7 = 128$. Όποτε σε κάθε περίπτωση το least-significant bit θα μένει ανεκμετάλλευτο. Για τον λόγο αυτό το τελευταίο bit θα χρησιμοποιείται ως flag για την αποστολή μίας τιμής όταν το operation είναι BOTH. Συνεπώς αν βρίσκουμε 1 στο τελικό bit και μας έρθει μήνυμα μεγέθους μιας μέτρησης ενώ του query ζητάει και τις 2. Τότε θα αποστέλλουμε το max bitwise or με το (0x01). Με αποτέλεσμα να περνάει όλη η μέτρηση και στο τελευταίο bit να τοποθετείται μονάδα.
- Απαιτείται αποστολή μόνο του count: ελέγχεται ξανά αν το operation είναι BOTH ή COUNT. Αν είναι COUNT τότε απλώς αποστέλλεται η τιμή. Διαφορετικά, παρόλο που δεν χρησιμοποιείται το τελευταίο bit για να είμαστε βέβαιοι γίνεται bitwise and με(0xFE) ώστε να έχουμε ολόκληρη την μέτρηση και βέβαιο μηδενικό στην τελική θέση.

Εφόσον έχουν γίνει όλα τα παραπάνω το μόνο που απομένει είναι να ανανεωθεί η προηγούμενη τιμή της συναθροιστικής συνάρτησης που μεταβλήθηκε και να τυπωθεί το τελικό αποτέλεσμα.

Η αντίστροφη διαδικασία πραγματοποιείται ακολουθώντας από το event *MyReceive.receive(). Συγκεκριμένα παραλαμβάνεται το AMPacket που περιλαμβάνει την τιμή που μας απέστηλαν το οποίο εισάγεται στην MyReceiveQueue() προτού κληθεί το task receiveMyTask().

Μέσα στο ReceiveMyTask ελέγχεται σε πρώτη φάση το operation. Αν πρόκειται για μονής πολλαπλότητας query τότε ανανεώνεται η αντίστοιχη τιμή στο παιδί και αν είναι πρόκειται για πρώτη παραλαβή από τον συγκεκριμένο αποστολέα τότε εισάγεται στα παιδιά του αποδέκτη. Διαφορετικά είναι query πολλαπλότητας 2 τότε ελέγχεται το μέγεθος του παραληφθέντος πακέτου. Αν Είναι για δυο μετρήσεις τότε ανανεώνονται τα 2 παιδία του παιδιού, αλλιώς αν το least significant bit είναι 1 τότε ανανεώνεται μόνο το field max του παδιού. Το αντίστροφο ισχύει για το count.

Με τον τρόπο αυτό επιτυγχάνεται σε μεγάλο βαθμό ένας από τους βασικούς μας στόχους κατά την υλοποίηση του πρωτοκόλλου, που δεν είναι άλλος από την ελαχιστοποίηση της μεταδιδόμενης πληροφορίας από τους κόμβους. Συγκεκριμένα ο κόμβος μας δεν αποστέλλει καθόλου άχρηστη πληροφορία κατά τις μεταδόσεις του και επιτυγχάνει με μόνο 1 byte σε περίπτωση που ανανεώνεται μια τιμή, ή 2 bytes σε περίπτωση που ανανεώνονται 2, να αποστείλει στον πατέρα του όλη την ζητούμενη πληροφορία για την συναθροιστική συνάρτηση της επιλογής μας. Στην πραγματικότητα η εισαγωγή του bit mask μας ελευθερώνει 2 επιπλέον bytes καθώς η εναλλακτική μας, εφόσον δεν θα μπορούσαμε με αποστολή Measurement να αναγνωρίσουμε για ποια μέτρηση πρόκειται, θα ήταν να στέλνουμε ένα TwoMeasurement Msg δηλαδή 2 bytes, επιβαρυμένα από ένα extra flag πεδίο το οποίο θα μετέφερε την πληροφορία για το ποιος κόμβος μεταβάλλεται.

Δ.4 Βελτιωμένη λογική TiNa με ελαχιστοποίηση χρόνου που κόμβος λαμβάνει δεδομένα

Μια άλλη σημαντική βελτίωση που θα μπορούσε να επιτευχθεί είναι να ελαχιστοποιήσουμε τον χρόνο κατά τον οποίο ο δέκτης ενός αισθητήρα θα λαμβάνει πληροφορία. Το παραπάνω ζητούμενο, μπορεί άμεσα να επιτευχθεί εάν μειώσουμε το παράθυρο κατά το οποίο αποστέλλουν τα δεδομένα τους οι κόμβοι κάθε επιπέδου.

Ωστόσο η τροποποίηση αυτή ενέχει και κρυφούς κινδύνους. Αναλυτικότερα αν κάνουμε την θεώρηση ότι σε ένα επίπεδο περιέχεται μεγάλο πλήθος κόμβων, η μείωση του πλάτους του παραθύρου θα έχει ως αποτέλεσμα οι κόμβοι να έχουν μεγαλύτερη πιθανότητα σύγκρουσης παρά την τυχαία εκκίνηση αποστολής τους με βάση τον *same_level_variator.* Βασιζόμενοι σε αυτό, παρόλο που στα περισσότερα topology αρχεία μας ακόμη και με TIMER_FAST_PERIOD = 64ms δεν είχαμε σπουδαίο αριθμό συγκρούσεων, η τιμή του τελικά ανατέθηκε σε 128 ms θέλοντας να αυξήσουμε την ασφάλεια του δικτύου σε απώλειες.

Αξίζει να σημειωθεί ότι σε δίκτυο όπου οι κόμβοι μας έχουν πολλαπλές συνδέσεις μεταξύ τους. Συνεπώς έχουν περισσότερες από μία επιλογές αποστολής σε πατέρα, ο χρόνος της περιόδου θα μπορούσε να τροποποιηθεί με μεγαλύτερη ασφάλεια καθώς η πληροφορία θα μπορούσε να διατηρείται με μεγαλύτερη ασφάλεια εντός του δικτύου ακόμη και αν οι συγκρούσεις ήταν περισσότερες.

Δ.5 Σημείωση Τυχαιότητας

Αξίζει να σημειωθεί ότι για να μπορέσουμε να παράγουμε καλύτερα τυχαίους αριθμούς λαμβάνουμε ένα διαφορετικό Seed για κάθε κόμβο μήκους 16-bit το οποίο με την χρήση της βιβλιοθήκης *dev/urandom* μας παρέχει την δυνατότητα να δημιουργούμε αξιόπιστους random generators.

Ε. Παρουσίαση και επεξήγηση λειτουργίας

Ε.1. Παρουσίαση Routing Σε Αρχείο topology2.txt

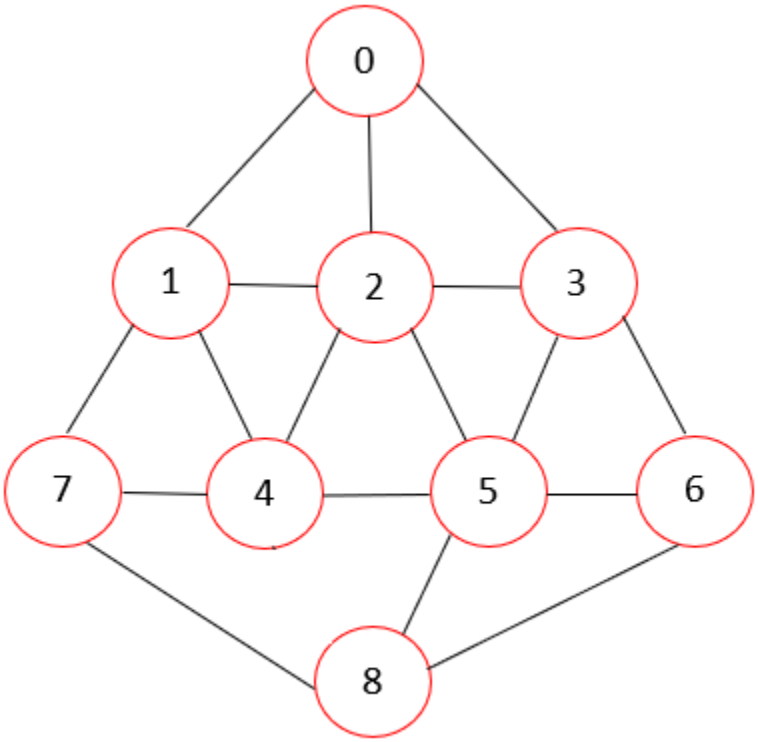
```
======================================== ROUND 1 ========================================
0:0:10.125000020 DEBUG (0): sendRoutingTask(): Send returned success!!!
0:0:10.133300752 DEBUG (3): Routing: Something received!!!  from 0
0:0:10.133300752 DEBUG (2): Routing: Something received!!!  from 0
0:0:10.133300752 DEBUG (1): Routing: Something received!!!  from 0
0:0:10.257812520 DEBUG (1): sendRoutingTask(): Send returned success!!!
0:0:10.257812520 DEBUG (2): sendRoutingTask(): Send returned success!!!
0:0:10.257812520 DEBUG (3): sendRoutingTask(): Send returned success!!!
0:0:10.261383055 DEBUG (2): Routing: Something received!!!  from 1
0:0:10.261383055 DEBUG (7): Routing: Something received!!!  from 1
0:0:10.261383055 DEBUG (4): Routing: Something received!!!  from 1
0:0:10.261383055 DEBUG (0): Routing: Something received!!!  from 1
0:0:10.267593346 DEBUG (3): Routing: Something received!!!  from 2
0:0:10.267593346 DEBUG (5): Routing: Something received!!!  from 2
0:0:10.267593346 DEBUG (1): Routing: Something received!!!  from 2
0:0:10.267593346 DEBUG (0): Routing: Something received!!!  from 2
0:0:10.267593346 DEBUG (4): Routing: Something received!!!  from 2
0:0:10.269424391 DEBUG (6): Routing: Something received!!!  from 3
0:0:10.269424391 DEBUG (5): Routing: Something received!!!  from 3
0:0:10.269424391 DEBUG (2): Routing: Something received!!!  from 3
0:0:10.269424391 DEBUG (0): Routing: Something received!!!  from 3
0:0:10.385742207 DEBUG (4): sendRoutingTask(): Send returned success!!!
0:0:10.385742208 DEBUG (7): sendRoutingTask(): Send returned success!!!
0:0:10.388839724 DEBUG (5): Routing: Something received!!!  from 4
0:0:10.388839724 DEBUG (7): Routing: Something received!!!  from 4
0:0:10.388839724 DEBUG (2): Routing: Something received!!!  from 4
0:0:10.388839724 DEBUG (1): Routing: Something received!!!  from 4
0:0:10.392578145 DEBUG (5): sendRoutingTask(): Send returned success!!!
0:0:10.392837503 DEBUG (8): Routing: Something received!!!  from 7
0:0:10.392837503 DEBUG (4): Routing: Something received!!!  from 7
0:0:10.392837503 DEBUG (1): Routing: Something received!!!  from 7
0:0:10.393554708 DEBUG (6): sendRoutingTask(): Send returned success!!!
0:0:10.397674557 DEBUG (8): Routing: Something received!!!  from 6
0:0:10.397674557 DEBUG (5): Routing: Something received!!!  from 6
0:0:10.397674557 DEBUG (3): Routing: Something received!!!  from 6
0:0:10.399841286 DEBUG (8): Routing: Something received!!!  from 5
0:0:10.399841286 DEBUG (6): Routing: Something received!!!  from 5
0:0:10.399841286 DEBUG (4): Routing: Something received!!!  from 5
0:0:10.399841286 DEBUG (3): Routing: Something received!!!  from 5
0:0:10.399841286 DEBUG (2): Routing: Something received!!!  from 5
0:0:10.517578145 DEBUG (8): sendRoutingTask(): Send returned success!!!
0:0:10.520217900 DEBUG (7): Routing: Something received!!!  from 8
0:0:10.520217900 DEBUG (6): Routing: Something received!!!  from 8
0:0:10.520217900 DEBUG (5): Routing: Something received!!!  from 8
0:0:12.000000010 DEBUG (0): Routing was finished!!!
0:0:12.000000010 DEBUG (1): Routing was finished!!!
0:0:12.000000010 DEBUG (2): Routing was finished!!!
0:0:12.000000010 DEBUG (3): Routing was finished!!!
0:0:12.000000010 DEBUG (4): Routing was finished!!!
0:0:12.000000010 DEBUG (5): Routing was finished!!!
0:0:12.000000010 DEBUG (6): Routing was finished!!!
0:0:12.000000010 DEBUG (7): Routing was finished!!!
0:0:12.000000010 DEBUG (8): Routing was finished!!!
0:0:12.000000010 DEBUG (9): Routing was finished!!!
```

Από το screenshot αυτό μπορούμε να παρακολουθήσουμε την λειτουργία του προγράμματός μας κατά το Routing. Επιλέχθηκε εσκεμμένα το αρχείο topology2.txt καθώς εφόσον αποτελούσε δεδομένο αρχείο η κατανόηση του θα ήταν ταχύτερη για τον εξεταστή. Αξίζει επιπλέον να σημειωθεί ότι για τους σκοπούς του πειράματος αυτό η παράμετρος -50 έχει τεθεί σε 0 για να μην έχουμε απώλειες και να μπορέσουμε να παρατηρήσουμε μια ολοκληρωμένη απεικόνιση της διαδικασίας.

Τα πεδία που παρουσιάζονται στο screenshot είναι τα ακόλουθα:

1. SendRoutingTask(): Send Return Success !!!: Το πεδίο αυτό εκτυπώνεται από το SendRoutingTask όπως και αναγράφεται και παρουσιάζει την επιτυχημένη αποστολή του routingMsg από κάποιον κόμβο.
2. *Routing: Something received!!! From: x :* Το πεδίο αυτό προέρχεται από το *RoutingReceive.receive() και εκτυπώνεται όταν κάποιος κόμβος λάβει κάποιο μήνυμα Routing με source τον κόμβο x.*
3. *Routing was finished!!!:* Το πεδίο αυτό εκτυπώνεται μέσα στο MyTimer.fired() όταν χτυπήσει για 1η φορά κατά την ολοκλήρωση του Routing. Ο χρόνος που δίνεται ώσπου να γίνει πρώτη φορά fired() δίδεται στο header file και όπως βλέπουμε και από τον χρόνο της προσομοίωσης ισούται με 2s ώστε με βεβαιότητα να έχει ολοκληρωθεί το routing όλων των κόμβων. Εϊναι εμφανές ότι ο χρόνος είναι παραπάνω από αρκετός καθώς το τελευταίο μήνυμα Routing ελήφθη περίπου ενάμιση δευτερόλεπτο πρίν.

Το δένδρο που σχηματίζεται μετά τις συνδέσεις του Routing είναι το ακόλουθο:



Το δένδρο λοιπόν δημιουργείται με κατάλληλο τρόπο και βλέπουμε ότι οι κόμβοι αποστέλλουν το μήνυμα του Routing με σειρά βάθους. Δηλαδή κάθε κόμβος που λαμβάνει μήνυμα από μεγαλύτερο επίπεδο στέλνει κι αυτός με την σειρά του.

E.2. Παρουσίαση Λειτουργίας MAX και COUNT χωριστά Σε Αρχείο topology.txt

Η λειτουργία αυτή θα παρουσιαστεί σύντομα καθώς η ανάλυση θα ακολουθήσει στην επόμενη ενότητα που θα εκτυπώνονται και οι 2 μαζί. Σε πρώτη φάση απλά θα αναδειχθεί η ορθή λειτουργία των 2 λειτουργιών χωριστά.

Τα ακόλουθα screenshots προέρχονται από το μικρό topology.txt αρχείο για να γίνει μια σύντομη παρουσίαση των αποτελεσμάτων. Έχουμε λοιπόν:

```
0:0:10.754882833 DEBUG (7): sendRoutingTask(): Send returned success!!!
0:0:10.760360706 DEBUG (5): Routing: Something received!!!  from 4
0:0:10.763732878 DEBUG (1): Routing: Something received!!!  from 2
0:0:11.009765645 DEBUG (5): sendRoutingTask(): Send returned success!!!
0:0:11.014755240 DEBUG (4): Routing: Something received!!!  from 5
0:0:12.000000010 DEBUG (0): Routing was finished!!!
0:0:12.000000010 DEBUG (1): Routing was finished!!!
0:0:12.000000010 DEBUG (2): Routing was finished!!!
0:0:12.000000010 DEBUG (3): Routing was finished!!!
0:0:12.000000010 DEBUG (4): Routing was finished!!!
0:0:12.000000010 DEBUG (5): Routing was finished!!!
0:0:12.000000010 DEBUG (6): Routing was finished!!!
0:0:12.000000010 DEBUG (7): Routing was finished!!!
0:0:12.000000010 DEBUG (8): Routing was finished!!!
0:0:12.000000010 DEBUG (9): Routing was finished!!!
0:0:29.145507833 DEBUG (5): Node's COUNT 1
0:0:29.145507833 DEBUG (5): The new Count value is ACCEPTED.
0:0:29.145507833 DEBUG (5): Tct: 10       Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:29.145507833 DEBUG (5): SendMyTask(): Send Count Only
0:0:29.145507833 DEBUG (5): SEND DONE True
0:0:29.154891933 DEBUG (4): MyReceive: A value received!!!  from 5
0:0:29.154891943 DEBUG (4): ======================================== Count Received ========================================
0:0:29.154891943 DEBUG (4): Received from childId 5  - count 1
0:0:29.375000020 DEBUG (7): Node's COUNT 1
0:0:29.375000020 DEBUG (7): The new Count value is ACCEPTED.
0:0:29.375000020 DEBUG (7): Tct: 10       Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:29.375000020 DEBUG (7): SendMyTask(): Send Count Only
0:0:29.375000020 DEBUG (7): SEND DONE True
0:0:29.379882832 DEBUG (2): Node's COUNT 1
0:0:29.379882832 DEBUG (2): The new Count value is ACCEPTED.
0:0:29.379882832 DEBUG (2): Tct: 10       Previous Count: 0      New Count: 1    Reject-Range:[0.000000,0.000000]
0:0:29.379882832 DEBUG (2): SendMyTask(): Send Count Only
0:0:29.379882832 DEBUG (2): SEND DONE True
0:0:29.382461553 DEBUG (1): MyReceive: A value received!!!  from 2
0:0:29.382461563 DEBUG (1): ======================================== Count Received ========================================
0:0:29.382461563 DEBUG (1): Received from childId 2  - count 1
0:0:29.384521449 DEBUG (1): MyReceive: A value received!!!  from 7
0:0:29.384521459 DEBUG (1): ======================================== Count Received ========================================
0:0:29.384521459 DEBUG (1): Received from childId 7  - count 1
0:0:29.486328145 DEBUG (4): ChildrenId: 5        Of: 4   has COUNT 1
0:0:29.486328145 DEBUG (4): Node's COUNT 2
0:0:29.486328145 DEBUG (4): The new Count value is ACCEPTED.
0:0:29.486328145 DEBUG (4): Tct: 10       Previous Count: 0      New Count: 2    Reject-Range:[0.000000,0.000000]
0:0:29.486328145 DEBUG (4): SendMyTask(): Send Count Only
0:0:29.486328145 DEBUG (4): SEND DONE True
0:0:29.491348257 DEBUG (1): MyReceive: A value received!!!  from 4
0:0:29.491348267 DEBUG (1): ======================================== Count Received ========================================
0:0:29.491348267 DEBUG (1): Received from childId 4  - count 2
0:0:29.553710957 DEBUG (1): ChildrenId: 2        Of: 1   has COUNT 1
0:0:29.553710957 DEBUG (1): ChildrenId: 7        Of: 1   has COUNT 1
0:0:29.553710957 DEBUG (1): ChildrenId: 4        Of: 1   has COUNT 2
0:0:29.553710957 DEBUG (1): Node's COUNT 5
0:0:29.553710957 DEBUG (1): The new Count value is ACCEPTED.
0:0:29.553710957 DEBUG (1): Tct: 10       Previous Count: 0      New Count: 5    Reject-Range:[0.000000,0.000000]
0:0:29.553710957 DEBUG (1): SendMyTask(): Send Count Only
0:0:29.553710957 DEBUG (1): SEND DONE True
0:0:29.562103242 DEBUG (0): MyReceive: A value received!!!  from 1
0:0:29.562103252 DEBUG (0): ======================================== Count Received ========================================
0:0:29.562103252 DEBUG (0): Received from childId 1  - count 5
0:0:29.926757832 DEBUG (0): ChildrenId: 1        Of: 0   has COUNT 5
0:0:29.926757832 DEBUG (0): Node's COUNT 6
0:0:29.926757832 DEBUG (0): The new Count value is ACCEPTED.
0:0:29.926757832 DEBUG (0): Tct: 10       Previous Count: 0      New Count: 6    Reject-Range:[0.000000,0.000000]
0:0:29.926757832 DEBUG (0): SendMyTask(): Send Count Only
0:0:29.926757832 DEBUG (0): SEND DONE True
0:0:29.926757832 DEBUG (0): Final Result: Count = 6
0:0:30.000000010 DEBUG (0):
```

Μετά το πέρας του routing αρχίζει να γίνεται η αποστολή και παραλαβή μετρήσεων. Συγκεκριμένα βλέπουμε το Routing να τελειώνει μετά από 12s προσομοίωσης. Έπειτα υπάρχει ένα κενό στο δίκτυο το οποίο διαλύεται στο τέλος της περιόδου όταν αρχίζει να αποστέλλει για πρώτη φορά ο κόμβος 5 που βρίσκεται στο μεγαλύτερο βάθος(3). Οι κόμβοι οι οποίοι δεν έλαβαν

κάτι κατά το routing βλέπουμε ότι δεν αποστέλλουν τίποτα καθώς δεν προσδιορίστηκε για εκείνους κάποιο operation οπότε μέσω κατάλληλης if κάνουν return πριν δοκιμάσουν να στείλουν οτιδήποτε.

Έτσι λοιπόν από την συνάρτηση calculate_count() τυπώνεται η νέα τιμή count του κόμβου η οποία συγκρίνεται με το Tct ώστε να προσδιοριστεί αν θα πρέπει να αποσταλεί. Στην πρώτη περίοδο, στην οποία αναφέρεται το παραπάνω screenshot εφόσον δεν υπάρχει προηγούμενη τιμή για count, η σύγκριση με το tct θα είναι πάντα αληθής καθώς δεν υπάρχει κάποιο εύρος αποκλεισμού. Όλη η άνωθεν διαδικασία τυπώνεται εντός της sendMyTask() καθώς και προσδιορίζεται το είδος του operation για το οποίο στέλνουμε μετρήσεις με το SendMyTask() *Send Only Count.* Στην φάση αυτή τυπώνεται κατάλληλο μήνυμα το οποίο προσδιορίζει αν έγινε επιτυχής αποστολή.

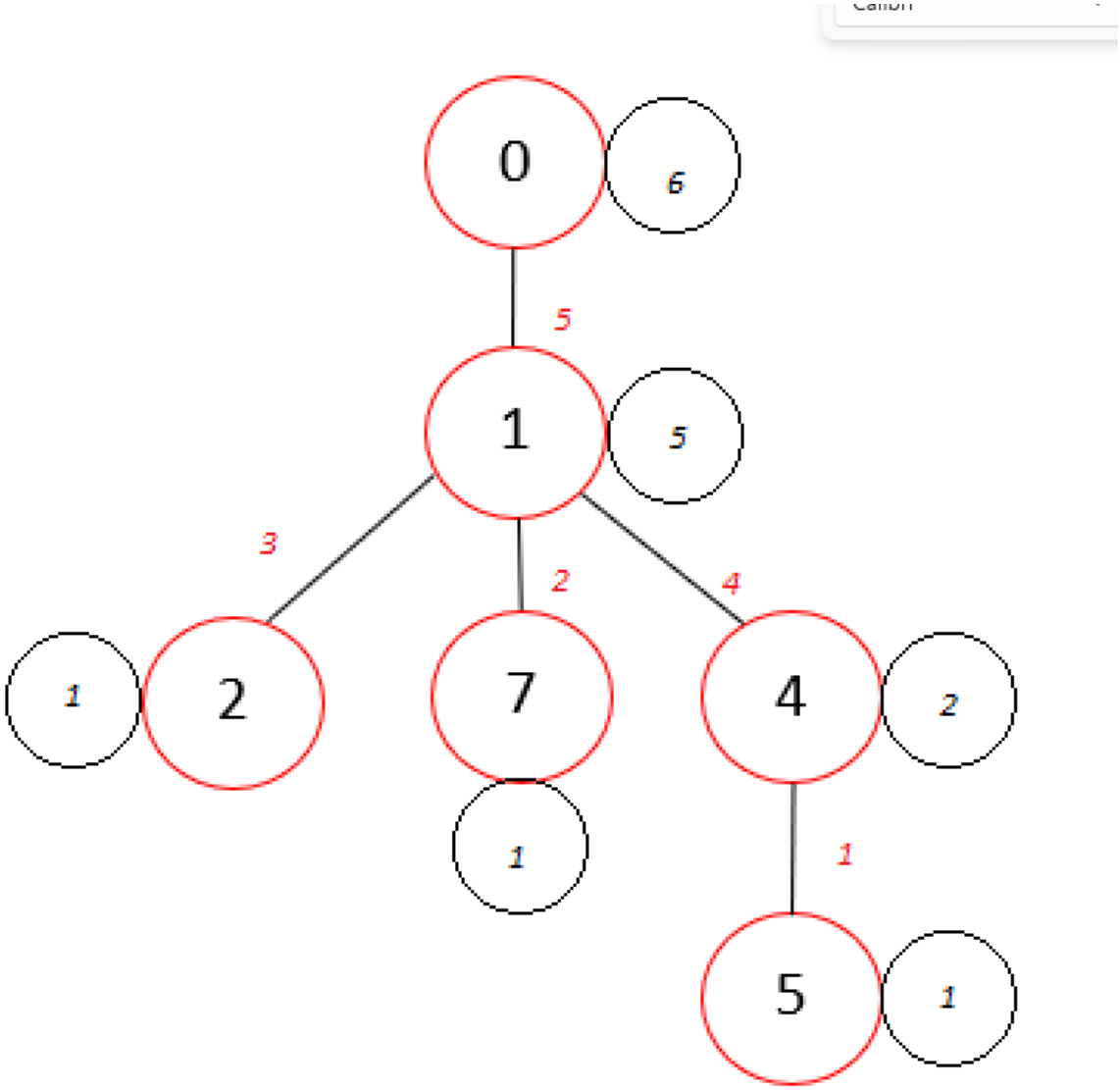
Με τον τρόπο αυτό ολοκληρώνεται η διαδικασία της αποστολής και ξεκινάει η διαδικασία της λήψης. Όπως βλέπουμε από το παραπάνω στιγμιότυπο η λειτουργικότητα μεταφέρεται στον κόμβο πατέρα του 5, τον κόμβο 4, ο οποίος αρχίζει λίγο αργότερα, να λαμβάνει το μήνυμα που έφθασε, ενώ τυπώνει τι παρέλαβε καθώς και από ποιο παιδί προέρχεται.

Λίγο χρόνο αργότερα, στο επόμενο timeslice βλέπουμε τους κόμβους αμέσως χαμηλότερου βάθους να αρχίζουν με την σειρά τους να εκτελούν την ίδια διαδικασία αποστολής και παραλαβής μηνυμάτων. Αξίζει ωστόσο να σταθούμε στον υπολογισμό του count 4 όπου για πρώτη φορά βλέπουμε τον συναθροιστικό υπολογισμό καθώς κατά των υπολογισμό ο κόμβος αθροίζει το count του εαυτού του με ότι παρέλαβε από τα παιδιά του.

Με αντίστοιχο τρόπο μεταφέρεται η πληροφορία έως την ρίζα η οποία τελικά τυπώνει το αποτέλεσμα που κατέφθασε σε αυτήν. Το σημαντικό από όλα τα παραπάνω είναι το γεγονός ότι είναι φανερό ότι οι κόμβοι έχουν οργανωθεί επαρκώς σε σχέση με το TiNa και μιλάει κάθε κόμβος στο timeslice που παρέχεται για το επίπεδό του. Οπότε ξεκινάμε με μετάδοση από κόμβους μεγάλου βάθους, καταλήγοντας στην ρίζα που τυπώνει τα συγκεντρωτικά αποτελέσματα που υπολογίστηκαν κατά τις υπόλοιπες περιόδους.

Για αποφυγή επανάληψης δεν θα προστεθεί επιπλέον στιγμιότυπο από επόμενες περιόδους, ωστόσο η συμπεριφορά που παρατηρείται για το Count είναι ότι λόγω τις έλλειψης απωλειών (όχι συγκρούσεις και 0 στο αρχείο topology.txt) το tct δεν ξεπερνιέται και με τον τρόπο αυτό έως και την 30^{η} περίοδο δεν έχουμε άλλη αποστολή count.

Η μεταβίβαση των αποτελεσμάτων στο δέντρο γίνεται με τον ακόλουθο τρόπο:

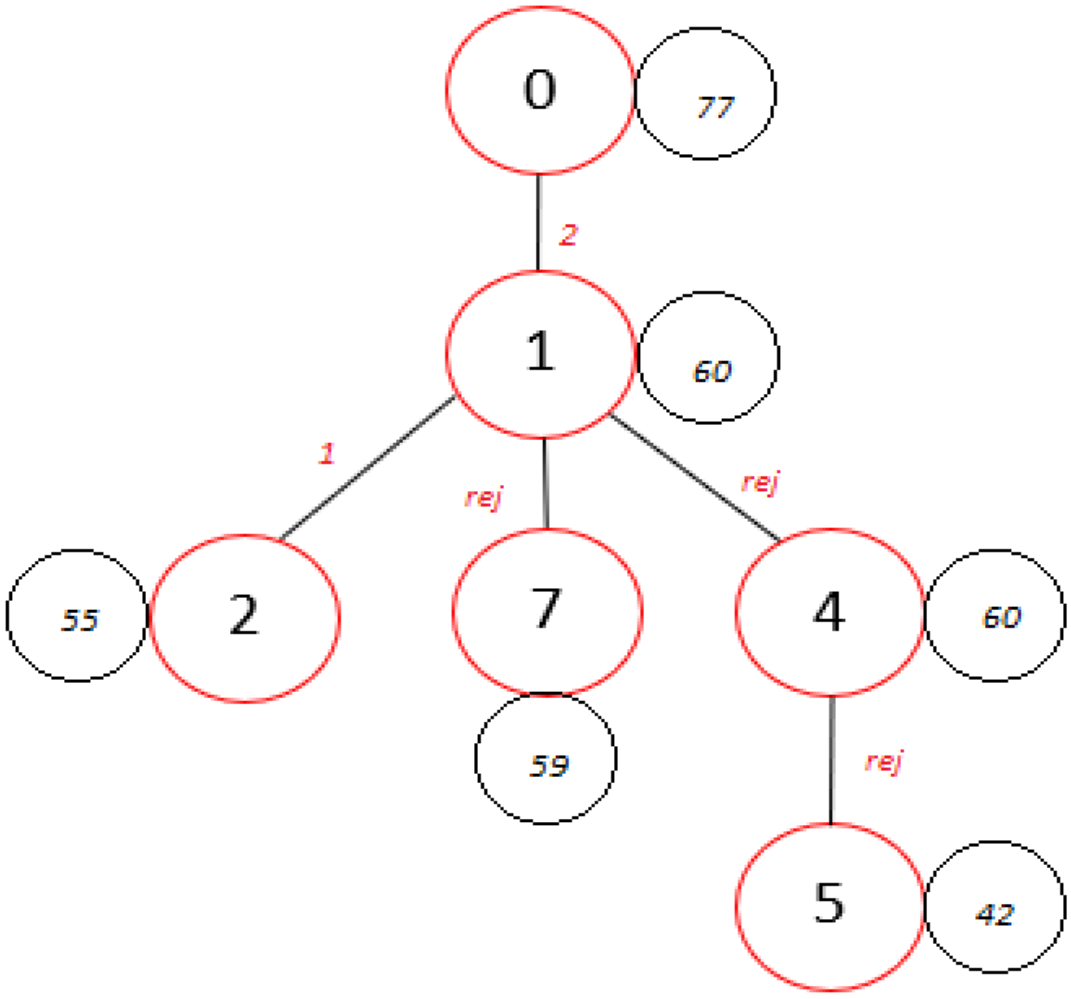


Όπου στους κύκλους περιέχεται η τιμή της συναθροιστικής συνάρτησης ενώ με κόκκινη αρίθμηση παρουσιάζεται η σειρά αποστολής. Αν κάποιος δεν αποστέλλετια τότε απλά στρο transition συμπληρώνεται η περιγραφή rej.

Με όμοιο τρόπο προκύπτουν τα αποτελέσματα και για το MAX function μόνο που στην περίπτωση αυτή λαμβάνοντας υπόψιν τόσο το tct όσο και το random step size, υπάρχει μια μεικτή συμπεριφορά με τα αποτελέσματα κάποιες φορές να αποστέλλονται και κάποιες όχι. Συγκεκριμένα στο ακόλουθο στιγμιότυπο μια ενδιάμεσης περιόδου βλέπουμε ότι μονάχα σε δύο κόμβους η αλλαγή του max ήταν αρκετά σημαντική για να ξεπεραστεί το tct όριο και να αποσταλεί η τιμή τους. Κατά τα υπόλοιπα οι εκτυπώσεις γίνονται όπως και στο Count.

```
======================================== ROUND 11 ========================================
0:5:29.079101583 DEBUG (5): Node's MAX 44
0:5:29.079101583 DEBUG (5): The new Max value is REJECTED.
0:5:29.079101583 DEBUG (5): Tct: 20       Previous Max: 42       New Max: 44    Reject-Range:[33.600002,50.400002]
0:5:29.411132832 DEBUG (2): Node's MAX 55
0:5:29.411132832 DEBUG (2): The new Max value is ACCEPTED.
0:5:29.411132832 DEBUG (2): Tct: 20       Previous Max: 69       New Max: 55    Reject-Range:[55.200001,82.800003]
0:5:29.411132832 DEBUG (2): SendMyTask(): Calculate Max Only
0:5:29.411132832 DEBUG (2): SEND DONE True
0:5:29.414047244 DEBUG (1): MyReceive: A value received!!!  from 2
0:5:29.414047254 DEBUG (1): ======================================== MAX Received ========================================
0:5:29.414047254 DEBUG (1): Received from childId 2  - max 55
0:5:29.432617207 DEBUG (4): ChildrenId: 5        Of: 4   has MAX 42
0:5:29.432617207 DEBUG (4): Node's MAX 60
0:5:29.432617207 DEBUG (4): The new Max value is REJECTED.
0:5:29.432617207 DEBUG (4): Tct: 20       Previous Max: 60       New Max: 60    Reject-Range:[48.000000,72.000000]
0:5:29.487304708 DEBUG (7): Node's MAX 58
0:5:29.487304708 DEBUG (7): The new Max value is REJECTED.
0:5:29.487304708 DEBUG (7): Tct: 20       Previous Max: 59       New Max: 58    Reject-Range:[47.200001,70.800003]
0:5:29.668945332 DEBUG (1): ChildrenId: 2        Of: 1   has MAX 55
0:5:29.668945332 DEBUG (1): ChildrenId: 4        Of: 1   has MAX 60
0:5:29.668945332 DEBUG (1): ChildrenId: 7        Of: 1   has MAX 59
0:5:29.668945332 DEBUG (1): Node's MAX 60
0:5:29.668945332 DEBUG (1): The new Max value is ACCEPTED.
0:5:29.668945332 DEBUG (1): Tct: 20       Previous Max: 77       New Max: 60    Reject-Range:[61.600002,92.400002]
0:5:29.668945332 DEBUG (1): SendMyTask(): Calculate Max Only
0:5:29.668945332 DEBUG (1): SEND DONE True
0:5:29.674255360 DEBUG (0): MyReceive: A value received!!!  from 1
0:5:29.674255370 DEBUG (0): ======================================== MAX Received ========================================
0:5:29.674255370 DEBUG (0): Received from childId 1  - max 60
0:5:29.930664082 DEBUG (0): ChildrenId: 1        Of: 0   has MAX 60
0:5:29.930664082 DEBUG (0): Node's MAX 90
0:5:29.930664082 DEBUG (0): The new Max value is REJECTED.
0:5:29.930664082 DEBUG (0): Tct: 20       Previous Max: 77       New Max: 90    Reject-Range:[61.600002,92.400002]
0:5:29.930664082 DEBUG (0): Final Result: Max = 77
0:5:30.000000010 DEBUG (0):
```

Τα αποτελέσματα του παραπάνω query φαίνονται ακολούθως:



E.3. Παρουσίαση Λειτουργίας MAX & COUNT σε Αρχείο myTopology.txt

Τέλος στην φάση αυτή παρουσιάζεται ένα ιδιαίτερα μεγάλο παράδειγμα καθώς το topology του ερωτήματος είναι διαμέτρου 6 ενώ ως range των αισθητήρων έχει δοθεί ως 1 ώστε ένας ενδιάμεσος κόμβος να έχει 4 συνδέσεις με γείτονες. Η επιλογή αυτή έγινε για να μπορέσουν να επεξηγηθούν επαρκώς τα αποτελέσματα καθώς με εύρος 1.5 κάθε τέτοιος κόμβος θα είχε 8 γείτονες.

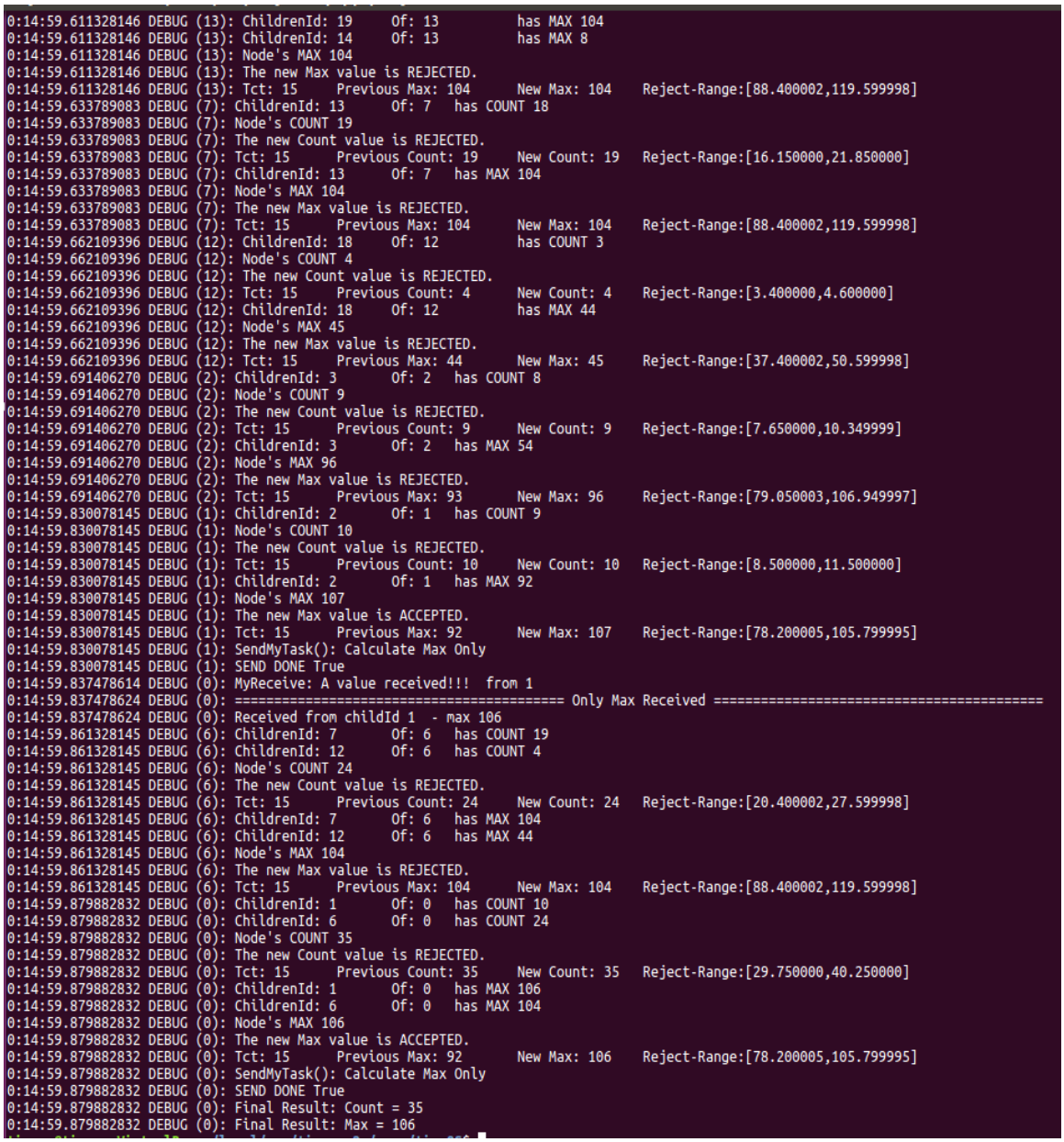
Συγκεκριμένα θα παραχθούν 2 πειράματα. Στο πρώτο η παράμετρος θα θορύβου για το sim θα είναι -50 ενώ στο άλλο θα έχει τιμή 0. Ξεκινώντας με την πρώτη περίπτωση έχουμε την ακόλουθη εκτύπωση ως αποτέλεσμα:

Για το πείραμα αυτό αξίζει να σημειωθεί ότι στο simulation.py αρχείο μεταβλήθηκε το range ώστε να γίνεται εκτέλεση για όλους τους αισθητήρες.

Το παρακάτω παράδειγμα επιλέγεται εσκεμμένα με σκοπό να γίνει φανερό ότι με το πολλούς κόμβους οι οποίοι δεν έχουν πάρα πολλές συνδέσεις μεταξύ τους (4 οι ενδιάμεσοι) ο παράγοντας του θορύβου μπορεί να προκαλέσει απώλεια. Για τον λόγο αυτό παρατηρείται το count να μην προκύπτει το αναμενόμενο. Ωστόσο ακόμη και με αυτή την σύνδεση, εφόσον υπάρχουν κι άλλες επιλογές, τις περισσότερες φορές η εκτέλεση επιστρέφει το επιθυμητό count 35.

```
0:14:59.541992209 DEBUG (18): The new Max value is REJECTED.
0:14:59.541992209 DEBUG (18): Tct: 15     Previous Max: 38      New Max: 41     Reject-Range:[32.299999,43.700001]
0:14:59.563476582 DEBUG (3): ChildrenId: 9       Of: 3   has COUNT 15
0:14:59.563476582 DEBUG (3): ChildrenId: 4       Of: 3   has COUNT 2
0:14:59.563476582 DEBUG (3): Node's COUNT 18
0:14:59.563476582 DEBUG (3): The new Count value is REJECTED.
0:14:59.563476582 DEBUG (3): Tct: 15      Previous Count: 18     New Count: 18   Reject-Range:[15.300000,20.699999]
0:14:59.563476582 DEBUG (3): ChildrenId: 9       Of: 3   has MAX 102
0:14:59.563476582 DEBUG (3): ChildrenId: 4       Of: 3   has MAX 42
0:14:59.563476582 DEBUG (3): Node's MAX 102
0:14:59.563476582 DEBUG (3): The new Max value is REJECTED.
0:14:59.563476582 DEBUG (3): Tct: 15      Previous Max: 102      New Max: 102    Reject-Range:[86.700005,117.299995]
0:14:59.651367208 DEBUG (7): ChildrenId: 13      Of: 7   has COUNT 9
0:14:59.651367208 DEBUG (7): Node's COUNT 10
0:14:59.651367208 DEBUG (7): The new Count value is REJECTED.
0:14:59.651367208 DEBUG (7): Tct: 15      Previous Count: 10     New Count: 10   Reject-Range:[8.500000,11.500000]
0:14:59.651367208 DEBUG (7): ChildrenId: 13      Of: 7   has MAX 88
0:14:59.651367208 DEBUG (7): Node's MAX 88
0:14:59.651367208 DEBUG (7): The new Max value is REJECTED.
0:14:59.651367208 DEBUG (7): Tct: 15      Previous Max: 88       New Max: 88     Reject-Range:[74.800003,101.199997]
0:14:59.665039082 DEBUG (2): ChildrenId: 3       Of: 2   has COUNT 18
0:14:59.665039082 DEBUG (2): Node's COUNT 19
0:14:59.665039082 DEBUG (2): The new Count value is REJECTED.
0:14:59.665039082 DEBUG (2): Tct: 15      Previous Count: 19     New Count: 19   Reject-Range:[16.150000,21.850000]
0:14:59.665039082 DEBUG (2): ChildrenId: 3       Of: 2   has MAX 102
0:14:59.665039082 DEBUG (2): Node's MAX 102
0:14:59.665039082 DEBUG (2): The new Max value is REJECTED.
0:14:59.665039082 DEBUG (2): Tct: 15      Previous Max: 102      New Max: 102    Reject-Range:[86.700005,117.299995]
0:14:59.701171896 DEBUG (12): ChildrenId: 18     Of: 12          has COUNT 2
0:14:59.701171896 DEBUG (12): Node's COUNT 3
0:14:59.701171896 DEBUG (12): The new Count value is REJECTED.
0:14:59.701171896 DEBUG (12): Tct: 15     Previous Count: 3      New Count: 3    Reject-Range:[2.550000,3.450000]
0:14:59.701171896 DEBUG (12): ChildrenId: 18     Of: 12          has MAX 38
0:14:59.701171896 DEBUG (12): Node's MAX 71
0:14:59.701171896 DEBUG (12): The new Max value is REJECTED.
0:14:59.701171896 DEBUG (12): Tct: 15     Previous Max: 67       New Max: 71     Reject-Range:[56.950001,77.049995]
0:14:59.810546895 DEBUG (1): ChildrenId: 7       Of: 1   has COUNT 10
0:14:59.810546895 DEBUG (1): ChildrenId: 2       Of: 1   has COUNT 19
0:14:59.810546895 DEBUG (1): Node's COUNT 30
0:14:59.810546895 DEBUG (1): The new Count value is REJECTED.
0:14:59.810546895 DEBUG (1): Tct: 15      Previous Count: 30     New Count: 30   Reject-Range:[25.500000,34.500000]
0:14:59.810546895 DEBUG (1): ChildrenId: 7       Of: 1   has MAX 88
0:14:59.810546895 DEBUG (1): ChildrenId: 2       Of: 1   has MAX 102
0:14:59.810546895 DEBUG (1): Node's MAX 102
0:14:59.810546895 DEBUG (1): The new Max value is REJECTED.
0:14:59.810546895 DEBUG (1): Tct: 15      Previous Max: 102      New Max: 102    Reject-Range:[86.700005,117.299995]
0:14:59.849609395 DEBUG (6): ChildrenId: 12      Of: 6   has COUNT 0
0:14:59.849609395 DEBUG (6): Node's COUNT 1
0:14:59.849609395 DEBUG (6): The new Count value is REJECTED.
0:14:59.849609395 DEBUG (6): Tct: 15      Previous Count: 1      New Count: 1    Reject-Range:[0.850000,1.150000]
0:14:59.849609395 DEBUG (6): ChildrenId: 12      Of: 6   has MAX 66
0:14:59.849609395 DEBUG (6): Node's MAX 66
0:14:59.849609395 DEBUG (6): The new Max value is REJECTED.
0:14:59.849609395 DEBUG (6): Tct: 15      Previous Max: 58       New Max: 66     Reject-Range:[49.300003,66.699997]
0:14:59.940429707 DEBUG (0): ChildrenId: 1       Of: 0   has COUNT 30
0:14:59.940429707 DEBUG (0): ChildrenId: 6       Of: 0   has COUNT 1
0:14:59.940429707 DEBUG (0): Node's COUNT 32
0:14:59.940429707 DEBUG (0): The new Count value is REJECTED.
0:14:59.940429707 DEBUG (0): Tct: 15      Previous Count: 32     New Count: 32   Reject-Range:[27.200001,36.799999]
0:14:59.940429707 DEBUG (0): ChildrenId: 1       Of: 0   has MAX 102
0:14:59.940429707 DEBUG (0): ChildrenId: 6       Of: 0   has MAX 58
0:14:59.940429707 DEBUG (0): Node's MAX 102
0:14:59.940429707 DEBUG (0): The new Max value is REJECTED.
0:14:59.940429707 DEBUG (0): Tct: 15      Previous Max: 102      New Max: 102    Reject-Range:[86.700005,117.299995]
0:14:59.940429707 DEBUG (0): Final Result: Count = 32
0:14:59.940429707 DEBUG (0): Final Result: Max = 102
```

Από την άλλη πλευρά αν το πρόγραμμα εκτελεστεί με παράγοντα θορύβου 0 τα αποτελέσματα γίνονται ακόμη πιο αξιόπιστα και το αποτέλεσμα που επιτρέφεται είναι count 35. Έχουμε λοιπόν:



Λόγω του μεγάλου μεγέθους του αρχείου παραλείφθηκε η δημιουργία του δέντρου καθώς οι συνδέσεις θα ήταν υπεράριθμες. Ελέχθηκε ωστόσο η αποστολή να γίνεται με την κατάλληλη σειρά επιπέδων και να μην πραγματοποιείται σύγχυση μεταξύ των timesices. Αν επιθυμούμε να επιτύχουμε ακόμη μεγαλύτερη ασφάλεια κατά των συγγρούσεων σε πολύ μεγάλα αρχεία, θα μπορούσαμε να επαναφέρουμε το TIMER_FAST_PERIOD στην αρχική τιμή 256 ms. Ωστόσο στα δεδομένα που ελέγχθηκαν έως και αυτό το πείραμα δεν τίθηκε κάποιο θέμα αξιοπιστίας.

ΣΤ. Διαμοιρασμός Εργασιών

Το βοηθητικό πρόγραμμα καθώς και οι αρχικές συναρτήσεις για υπολογισμό MAX και COUNT χωριστά υλοποιήθηκαν από τον Χρυσή Επαμεινώνδα. Επίσης αφαιρέθηκαν τον ίδιο τα τμήματα κώδικα που δεν ήταν απαραίτητα για την εργασία. Έπειτα οι διαδικασίες υπολογισμού count και max, τροποποιήθηκαν για να φθάσουν στην τελική τους μορφή από τον Παναγιώτη Σκλάβο ο οποίος συμπλήρωσε και το κομμάτι του χειρισμού και των 2 συναρτήσεων ταυτόχρονα μαζί με την τροποποίηση για ελλαχιστοποίηση μεταδιδόμενης πληροφορίας.

Σε γενικές γραμμές ωστόσο καθώς τα κομμάτια αυτά είχαν μεγάλη επικάλυψη ο εργασίες αυτές έγιναν σε μεγάλο βαθμό από κοινού.