School of
Electrical &
Computer
Engineering

Επεξεργασία και Διαχείριση Δεδομένων σε Δίκτυα Αιστητήρων

ΠΛΗ 516

Αναφορά πρώτης φάσης εργασίας στο tinyOS

Δημιουργία τοπολογίας, υλοποίηση συναθροιστικών επερωτήσεων με βάση το TAG

3/12/2018

Γιακουμάκης Παύλος Πάρις

Γκιώνης Νίκος

Βοηθητικό πρόγραμμα

Για να δημιουργήσουμε με αυτόματο τρόπο τα αρχεία τοπολογίας που είναι απαραίτητα προκειμένου να ελέγξουμε τη λειτουργία των προγραμμάτων, χρειάστηκε να υλοποιήσουμε ένα βοηθητικό πρόγραμμα για το οποίο επιλέξαμε τη γλώσσα Python. Το πρόγραμμα αυτό δέχεται ως παράμετρο τη διάμετρο D (ακέραιος) η οποία καθορίζει το μέγεθος του grid (άρα και τον αριθμό των κόμβων) καθώς και την εμβέλεια rng κάθε κόμβου (αριθμός κινητής υποδιαστολής). Έτσι, δημιουργήσαμε ένα grid μεγέθους $D \cdot D$ το οποίο αναπαραστήσαμε ως δυσδιάστατο πίνακα που περιέχει αριθμούς από 0 έως $D^2 - 1$ έτσι ώστε κάθε κόμβος j να ανήκει στη γραμμή j/D και τη στήλη j%D του πίνακα αυτού.

Εάν λοιπόν είχαμε, για παράδειγμα, ένα grid μεγέθους $D = 5$, τότε ο δυσδιάστατος πίνακας με τον οποίο το αναπαριστούμε θα είχε τη μορφή:

$$grid = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

Στη συνέχεια, για κάθε κόμβο j που υπάρχει στον πίνακα, βρήκαμε τους κόμβους με τους οποίους επικοινωνεί βάση της θέσης του και της εμβέλειας του (γείτονες του j), μέσω της συνάρτησης $findNeighbours(grid, j, D, rng)$ και τους κρατήσαμε σε έναν ξεχωριστό πίνακα neighbours.

Βέβαια, για να βρούμε αν δύο κόμβοι επικοινωνούν, θα πρέπει να βρούμε τη μεταξύ τους απόσταση. Αυτό, γίνεται μέσω της ευκλείδειας μετρικής, δηλαδή για κάθε κόμβο j που βρίσκεται στη γραμμή $row = j/D$ και στήλη $col = j\%D$, κάθε κόμβος που βρίσκεται στη γραμμή **x** και στήλη **y**, απέχει απόσταση:

$$dist = \sqrt{|x - row|^2 + |y - col|^2}$$

Τέλος, συμπληρώσαμε το αρχείο *topology.txt* βάση των αποτελεσμάτων που λάβαμε, σε μορφή παρόμοια με αυτή του αρχείου που μας δόθηκε και πιο συγκεκριμένα, κάθε γραμμή του αρχείου έχει τη μορφή:

$$[Κόμβος\ που\ μελεταμε]\ \ [Γειτονικός\ κόμβος]\ \ -50.0$$

Πρόγραμμα 1

Ο κώδικας του κύριου module βρισκόταν στο αρχείο SRTreeC.nc. Επίσης για τη διασύνδεση του κυρίως module με άλλα χρήσιμα modules, π.χ. timers, Active message, packets κ.ο.κ. χρησιμοποιήθηκε το αρχείο SRTreeApp.nc

Κατανόηση και εκκαθάριση κώδικα

Το πρώτο βήμα για την υλοποίηση της άσκησης ήταν η κατανόηση του κώδικα που δόθηκε και ο εντοπισμός παράλληλα τμημάτων που δεν χρησίμευαν με κάποιο τρόπο στην υλοποίηση σύμφωνα με το TAG, ώστε να αφαιρεθούν.

Αρχικά αφαιρέθηκε ο κώδικας που αφορούσε στη σειριακή επικοινωνία. Συγκεκριμένα αφαιρέθηκαν τα αντίστοιχα interfaces που χρησίμευαν για τη διαδικασία αυτή, καθώς και ο κώδικας που συμπεριλαμβανόταν μεταξύ των #ifdef SERIAL_EN και #endif. Επίσης αφαιρέθηκε ο κώδικας που αφορούσε στη χρήση και τη διαχείριση των LEDs, καθώς κάτι τέτοιο δε χρειαζόταν στην άσκηση.

Στη συνέχεια αφαιρέθηκαν και τμήματα κώδικα που αφορούσαν στη διαχείριση πιθανής απώλειας ενός task στην περίπτωση που ενώ ο κόμβος έκανε send ένα μύνημα, του ερχόταν και άλλο task για την αποστολή νέου μηνύματος, πρωτού προκύψει το event sendDone για το πρώτο μήνυμα. Συνεπώς αφαιρέθηκαν ο μετρητής LostTaskTimer όλες οι συναρτήσεις που αφορούσαν σε set Lost Task και set Send Busy. Ακόμη αφαιρέθηκαν τα τμήματα κώδικα που αφορούσαν στο PRINTFDBG_MODE, καθώς χρησιμοποιήθηκε η συνάρτηση dbg για την εκτύπωση των επιθυμητών μηνυμάτων.

Περνώντας στο κύριο τμήμα του κώδικα, αρχικά μελετήθηκαν τα τμήματα που σχετίζονταν με το routing. Με βάση το TAG, η διαδικασία του routing βασίζεται στη λογική του First-Heard-First για την επιλογή γονέα, ενώ τα παιδιά δεν απαιτείται να στείλουν κάποια επιβεβαίωση για την επιλογή του πατέρα τους. Στον κώδικα που δώθηκε, ο κόμβος 0, ξεκινά μετά από μία μικρή καθυστέρηση (ώστε να εξασφαλιστεί ότι όλοι οι κόμβοι έχουν κάνει boot), τη διαδικασία του routing. Αυτή αρχίζει με το event $routingMsgTimer.fired()$. Η ακριβής λειτουργία του συγκεκριμένου event θα αναλυθεί λεπτομερώς σε επόμενη ενότητα, ωστόσο είναι σημαντικό να αναφερθεί πώς αφαιρέθηκε η γραμμή που αφορά στην κλήση του μετρητή του routing περιοδικά. Στην άσκηση το ζητούμενο ήταν να γίνεται routing μία φορά κατά την πρώτη εποχή, και όχι επαναλαμβανόμενα. Επιπλέον αφαιρέθηκαν οι δύο κλήσεις commands setDestination και setPayloadLength πριν γίνει το enqueue στη RoutingSendQueue και γίνει post το αντίστοιχο task. Ο λόγος ήταν ότι το destination (broadcast) και το μέγεθος του μηνύματος (Routingmsg), είναι τα ίδια για κάθε κόμβο, οπότε αυτή η πληροφορία εισάγεται απευθείας εντός της AMSend command του αντίστοιχου task.

Η λογική του TAG όπως προαναφέρθηκε δεν προβλέπει ενημέρωση του γονέα από το παιδί ότι τον επέλεξε σαν πατέρα. Επομένως, στο receiveRoutingTask, όταν ένας κόμβος λαμβάνει μύνημα routing από κάποιον, τότε απευθείας τον θέτει ως πατέρα. Στη συνέχεια, ούτε

ενημερώνει, ούτε αναζητεί ενδεχόμενο καλύτερο πατέρα, οπότε αφαιρέθηκε όλο το αντίστοιχο τμήμα κώδικα. Γενικότερα, αφαιρέθηκαν όλα τα κομμάτια κώδικα που αφορούσαν στη λειτουργία ενημέρωσης του γονέα για την επιλογή μας (NotifyParent), αν και στη συνέχεια, για την προώθηση μυνημάτων με μετρήσεις, υλοποιήθηκε μια παρεμφερής λογική. Τέλος το μήνυμα routingMsg δεν χρειάζεται να έχει πεδίο για το senderID, καθώς αυτό συμπεριλαμβάνεται αυτόματα στο header του μηνύματος και είναι προσβάσιμο από τον παραλήπτη μέσω του command $AMPacket.source(\&RoutingMsg)$.

Επεξήγηση κώδικα και προσθήκες

Με βάση τις παραπάνω αφαιρέσεις και διορθώσεις, είχε υλοποιηθεί σωστά η διαδικασία του Routing. Ως εκ τούτου, κατά την πρώτη εποχή, δημιουργούταν το δέντρο διασύνδεσης των κόμβων, το οποίο στη συνέχεια αξιοποιούταν για τη δρομολόγηση των μετρήσεων προς τη ρίζα, υπολογίζοντας τις επιθυμητές επερωτήσεις.

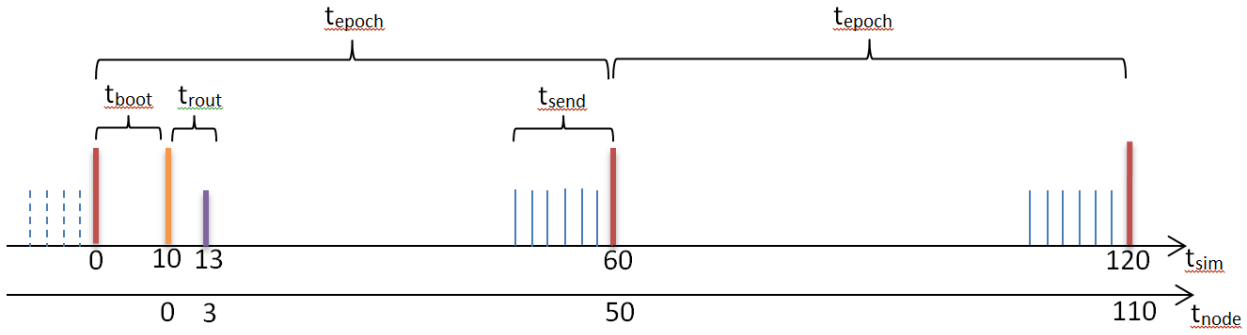
Στη συνέχεια έπρεπε κάθε κόμβος να λαμβάνει μέτρηση, να τη συναθροίζει με αυτές των παιδιών του και να την προωθεί στο γονέα του, σε κάθε εποχή. Αναγκαία προϋπόθεση για την παραπάνω διαδικασία, είναι να στέλνουν μύνημα πρώτα τα παιδιά και στη συνέχεια οι γονείς. Το πρώτο βήμα επομένως ήταν η δημιουργία του κατάλληλου timer κάθε κόμβου, που θα εξασφαλίζει την προαναφερθείσα προϋπόθεση. Για το λόγο αυτό, ο timer που θα δηλώνει πότε πρέπει να αποστείλει ο κάθε κόμβος δεδομένα, είναι αναγκαίο να εξαρτάται από το βάθος του κόμβου. Όσο μεγαλύτερο βάθος έχει ένας κόμβος, τόσο πιο νωρίς πρέπει να στείλει σε κάθε εποχή.

Με βάση τα παραπάνω, ο κάθε κόμβος πρέπει κατά τον ορισμό του timer του (SendMeasTimer) να γνωρίζει το βάθος του. Όμως την πληροφορία αυτή, την αποκτά ο κάθε κόμβος μετά το πέρας του routing του. Για το σκοπό αυτό, σε κάθε κόμβο εκκινεί αρχικά ο SendMeasTimer με $startOneShot(TIMER\_ROUTING\_DURATION)$, μέσα στο event startDone του radioControl. Η σταθερά αυτή ορίστηκε στα 3 δευτερόλεπτα, με τη λογική ότι μετά το πέρας των τριών δευτερολέπτων θα έχει τελειώσει το routing όλου του δέντρου. Όταν προκύψει κατά συνέπεια το event fired με το πέρας των τριών δευτερολέπτων, τότε ο κάθε κόμβος ορίζει τον μετρητή του για το πότε θα στέλνει δεδομένα από εδώ και στο εξής, έχοντας γνώση πλέον του βάθους του. Ο κάθε κόμβος ορίζει κατά συνέπεια:

$$callSendMeasTimer.startPeriodicAt$$
$$(((-(BOOT\_TIME) - ((curdepth + 1) * TIMER\_FAST\_PERIOD))$$
$$+(TOS\_NODE\_ID * 3)), TIMER\_PERIOD\_MILLI);$$

Η παραπάνω εντολή είναι κομβικής σημασίας για τη λειτουργία του προγράμματος. Αρχικά, ξεκινώντας από το δεύτερο όρισμα, αυτό είναι το TIMER_PERIOD_MILI, και αντιστοιχεί στα 60 δευτερόλεπτα που διαρκεί η μία εποχή. Δίνει δηλαδή την αναγκαία περιοδικότητα του timer ανά 60 δευτερόλεπτα.

Για την εξήγηση του πρώτου ορίσματος, δηλαδή τη στιγμή που θα αρχίσει να μετράει ο μετρητής, χρειάζεται λεπτομερής ανάλυση. Αρχικά είναι σημαντικό να αναφερθεί ότι η χρονική στιγμή $t = 0$ για τον μετρητή αντιστοιχίζεται στο χρόνο BOOT_TIME. Αυτός ο χρόνος χρησιμοποιείται στο mySimulation.py αρχείο και υποδηλώνει σε ποια χρονική στιγμή της προσομοίωσης θα κάνουν boot οι κόμβοι. Έχει οριστεί στα 10 δευτερόλεπτα, ως εκ τούτου, το σημείο $t = 0$ των κόμβων (και συνεπώς των μετρητών) είναι η στιγμή που κάνουν boot, άρα ο χρόνος προσομοίωσης των 10 δευτερολέπτων. Στο ακόλουθο σχήμα, φαίνεται η διαφορά μεταξύ του χρόνου των κόμβων και του χρόνου της προσομοίωσης, και ορίζονται μερικοί ενδιαφέροντες χρόνοι και γεγονότα που θα χρησιμεύσουν στη συνέχεια.



Υπολογισμός χρόνου timer

Σημαντικό είναι το γεγονός ότι οι εποχές ορίζονται με βάση το χρόνο προσομοίωσης, οπότε η πρώτη περίοδος έπρεπε να τελειώνει για $t_{sim} = 60$. Επίσης, σε κάθε επίπεδο του δέντρου, δώθηκε χρόνος ίσος με 250ms (TIMER_FAST_PERIOD), για να μεταδώσει προς το από πάνω επίπεδο. Ο χρόνος αυτός φαίνεται με τις μπλε γραμμές στο παραπάνω διάγραμμα. Κάθε επίπεδο παίρνει ένα τέτοιο slice χρόνου, και συνολικά όλα τα επίπεδα χρειάζονται χρόνο t_{send}. Όπως προαναφέρθηκε, το πιο βαθύ επίπεδο πρέπει να μεταδόσει πρώτο, πριν από τα προηγούμενα επίπεδα. Άρα θέλουμε να πάρει το πρώτο time slice. Αυτή η διαδικασία υλοποιείται μέσω του τμήματος $curdepth * TimerFastPeriod$. Όσο πιο βαθιά είναι ένας κόμβος, τόσο μεγαλύτερος ο προαναφερθείσας αριθμός. Επομένως, ο timer των κόμβων πρέπει να χτυπάει για t_{sim} ίσο με $60 - curdepth * TimerFastPeriod$. Ο timer των κόμβων όμως χρησιμοποιεί χρόνο t_{node} , οπότε ο αντίστοιχος χρόνος γίνεται $60 - curdepth * timerFastPeriod - BOOT\_TIME$. Αν όμως χρησιμοποιηθεί αυτός ο τύπος σαν όρισμα στην startPeriodicAt, τότε ο μετρητής θα ξεκινήσει να μετράει σε αυτό το σημείο, και θα κάνει fire μετά από 60 δευτερόλεπτα. Άρα στην πραγματικότητα θα έχουμε χάσει την πρώτη εποχή. Για τον λόγο αυτό χρειάζεται να αφαιρέσουμε από τον παραπάνω τύπο το χρόνο μίας περιόδου, δηλαδή το 60. Επομένως θα έχουμε:

$$60 - curdepth * timerFastPeriod - BOOT_{TIME} - 60 =$$

$$= - BOOT\_TIME - curdepth * timerFastPeriod$$

Ο παραπάνω χρόνος είναι αρνητικός. Αντιστοιχεί στις διακεκομμένες γραμμές που βρίσκονται πριν το μηδέν στο σχήμα, σε μία για κάθε κόμβο, ανάλογα το βάθος του. Με τον τρόπο αυτό, ο μετρητής για κάθε κόμβο θα κάνει fire 60 δευτερόλεπτά μετά την αντίστοιχη διακεκομμένη γραμμή, δηλαδή στην ακριβή στιγμή που πρέπει να στείλει για την πρώτη εποχή. Στη συνέχεια θα κάνει περιοδικά fire, την κατάλληλη στιγμή κάθε εποχή.

Με τον τρόπο αυτό, κάθε επίπεδο πλέον έστελνε μία ξεχωριστή χρονική στιγμή, με τα πιο βαθιά επίπεδα να στέλνουν πρώτα. Όμως οι κόμβοι που ανήκαν στο ίδιο επίπεδο, έστελναν όλοι ταυτόχρονα στην αρχή του time slice του επιπέδου τους. Το γεγονός αυτό δημιουργούσε πολλές συγκρούσεις, με συνέπεια να χάνονται αρκετά μηνύματα. Για τη βελτίωση του παραπάνω προβλήματος, έπρεπε να εισαχθεί μία διαφοροποίηση στο πότε θα στέλνει ο κάθε κόμβος μέσα στο slice του επιπέδου του. Θυμίζουμε ότι το κάθε time slice ορίστηκε ίσο με 250ms. Ο κάθε κόμβος πρέπει να στέλνει σε διαφορετική τιμή εντός των 250ms. Για το σκοπό αυτό χρησιμοποιήθηκε ένα μοναδικό χαρακτηριστικό για κάθε κόμβο, το TOS_NODE_ID, ώστε να εισαχθεί η απαραίτητη καθυστέρηση. Λαμβάνοντας υπ' όψιν ότι το μεγαλύτερο δυνατό TOS_NODE_ID είναι το 63 και αφήνοντας και 50ms στο τέλος του time slice, για να βεβαιώσουμε ότι κανένας κόμβος δεν θα παραβεί το χρόνο του επιπέδου του, προέκυψε ότι:

$$\frac{200}{63} = 3.17$$

Δηλαδή για κάθε κόμβο, η τελική του καθυστέρηση εντός του time slice μπορεί να προκύψει πολλαπλασιάζοντας το TOS_NODE_ID του με έναν παράγοντα 3. Με τον τρόπο αυτό μεγαλώνει η απόσταση μεταξύ των μεταδόσεων κόμβων του ίδιου επιπέδου, αλλά εξασφαλίζεται παράλληλα ότι κανένας κόμβος δε θα παραβεί τα 250ms του επιπέδου του.

Προσθέτοντας αυτή την τυχαιότητα εν τέλει, στον τύπο του μετρητή κάθε κόμβου, προκύπτει:

$$-BOOT_{TIME} - curdepth * timerFastPeriod + 3 * TOS\_NODE\_ID$$

Παρατηρώντας τον παραπάνω τύπο σε σχέση με το όρισμα της startPeriodic προκύπτει μία μόνο διαφορά, η οποία είναι ότι χρησιμοποιείται το curdepth+1 αντί του curdepth. Ο λόγος που συμβαίνει αυτό, είναι διότι η προσομοίωση διαρκεί ακριβώς 900s. Έτσι κατά τον τελευταίο γύρο, τον 15ο (γιατί (900s συνολικά) / (60s κάθε περιόδου) = 15 γύροι), η ρίζα δεν προλάβαινε να εκτυπώσει τα αποτελέσματα που είχε λάβει. Αυτό γινόταν γιατί, χρησιμοποιώντας τον τύπο χωρίς το +1, και με curdepth = 0, η ρίζα καλούνταν να γράψει τα αποτελέσματά της ακριβώς στα 900s, οπότε και χανόταν. Επομένως, με την εισαγωγή του +1, στην πραγματικότητα κάθε επίπεδο γίνεται shifted κατά ένα time slice νωρίτερα, ώστε να προλαβαίνει η ρίζα να εκτυπώνει το αποτέλεσμα και στον τελευταίο γύρο.

Για λόγους καθαρά ευκολίας στην ανάγνωση των αποτελεσμάτων, χρησιμοποιήθηκε ένας επιπλέον timer (RoundTimer), ο οποίος ανά 60 δευτερόλεπτα, εκτύπωνε τον αριθμό του γύρου που ξεκινάει.

Έχοντας υλοποιήσει το δυσκολότερο κομμάτι του χρονισμού, χρειαζόταν η δημιουργία των κατάλληλων στοιχείων για τη δημιουργία, αποστολή, λήψη και επεξεργασία των μυνημάτων. Αρχικά δημιουργήθηκε μία νέα δομή, το MeasMsg, το οποίο περιείχε τρία πεδία. Το sum, το count και το max. Ακολουθώντας τη λογική της διαδικασίας που υπήρχε για το NotifyParent, δημιουργήθηκαν τα αντίστοιχα MeasAMPacket, MeasAMSend και MeasPacket, καθώς και οι ουρές MeasSendQueue και MeasReceiveQueue. Όταν έκανε fire ο μετρητής που αναλύθηκε προηγουμένως, ο SendMeasTimer, τότε ο κόμβος αυτός έπαιρνε αρχικα τη δική του μέτρηση. Η μέτρηση αυτή έπρεπε να είναι ένας τυχαίος αριθμός μεταξύ 0 και 50, οπότε χρησιμοποιήθηκε το command $Random.rand16()\%50$ μέσω των έτοιμων interfaces της Random. Στη συνέχεια, με βάση αυτή την τιμή που μέτρησε, υπολόγιζε χρησιμοποιώντας και τις τιμές των παιδιών του, το sum, το count, και το max, που χρειάζεται να προωθήσει στο παραπάνω επίπεδο. Αυτές οι τιμές έμπαιναν σε μια δομή MeasMsg, οριζόταν ως προορισμός το parentID του κόμβου, και γινόταν enqueue. Έπειτα, το αντίστοιχο sendMeasTask έκανε dequeue το μήνυμα και το έστελνε στον πατέρα.

Όταν ο πατέρας λάμβανε το μύνημα, τότε το event $MeasReceive.receive()$ έβαζε το μήνυμα στην ουρά, και στο $receiveMeasTask$ ανανέωνε την τιμή που κρατούσε για το sum, count και max του συγκεκριμένου παιδιού.

Όπως προέκυψε, ο κάθε κόμβος έπρεπε να κρατάει τις τελευταίες τιμές που έχει λάβει από τα παιδιά του. Για το σκοπό αυτό κάθε κόμβος είχε έναν στατικό πίνακα 32 θέσεων (32 είναι ο μέγιστος αριθμός παιδιών ενός κόμβου). Όπως προαναφέρθηκε στο receiveMeasTask, ανανεώνεται η τιμή του παιδιού με βάση το νέο μύνημα που έλαβε, ή εάν δεν έχει λάβει ξανά μύνημα από αυτό το παιδί, τότε το προσθέτει στη λίστα των παιδιών του με την τιμή που το συνόδευε στο μύνημα. Υπενθυμίζεται ότι ένας κόμβος μπορεί να λάβει μήνυμα μόνο από τα παιδιά του, αφού αυτά στέλνουν με unicast χρησιμοποιώντας το parentID τους. Με την παραπάνω λειτουργία, ακόμη και αν χαθεί κάποια στιγμή ένα μύνημα από κάποιο παιδί, τότε ο κόμβος-πατέρας, χρησιμοποιεί την τελευταία αποθηκευμένη τιμή του στον υπολογισμό των επερωτήσεων. Να σημειωθεί ότι για την αποθήκευση της πληροφορίας των παιδιών του κάθε κόμβου σε πίνακα, χρησιμοποιείται μία δομή, η ChildInfo, η οποία έχει ως πεδία το senderID, το sum, το count και το max. Αυτές οι τιμές αφορούν στις συναθροίσεις του υποδέντρου του συγκεκριμένου παιδιού.

Τέλος, τα μηνύματα MeasMsg είχαν τρία πεδία, τα sum, count και max. Το ζητούμενο της πρώτης φάσης ήταν να υπολογίζονται ταυτόχρονα με βάση το TAG οι συναρτήσεις MAX και AVG. Οπότε, για την πρώτη συνάρτηση, κάθε κόμβος έπρεπε να προωθεί τη μέγιστη τιμή του υποδέντρου του, ενώ για τη δεύτερη, έπρεπε να προωθηθεί στη ρίζα η πληροφορία για το

συνολικό SUM του δέντρου και το συνολικό count του δέντρου, έτσι ώστε να μπορεί να υπολογιστεί το AVG στη ρίζα, ως η διαίρεση των δύο.

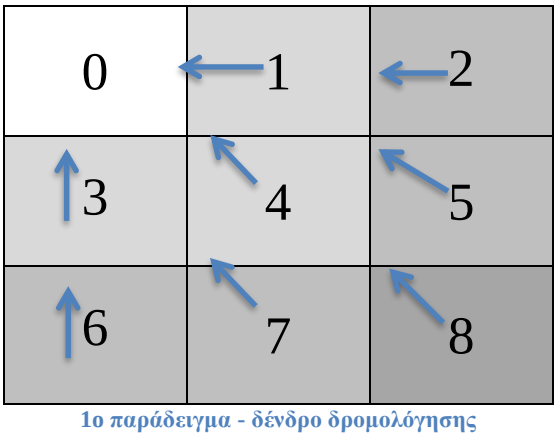
Παράδειγμα λειτουργίας 1

Στη συνέχεια παρουσιάζεται ένα παράδειγμα με τοπολογία 3x3 και συνολική εμβέλεια κόμβου 1.5. Από την προσομοίωση παρουσιάζεται ο πρώτος γύρος, έτσι ώστε να φανεί και η διαδικασία του routing.

```
349   0:0:10.250000010 DEBUG (0): ####################################
350   0:0:10.250000010 DEBUG (0): #######   ROUND   1    ##############
351   0:0:10.250000010 DEBUG (0): ####################################
352   0:0:10.257598862 DEBUG (4): New parent for NodeID= 4 : curdepth= 1 , parentID= 0
353   0:0:10.257598862 DEBUG (1): New parent for NodeID= 1 : curdepth= 1 , parentID= 0
354   0:0:10.257598862 DEBUG (3): New parent for NodeID= 3 : curdepth= 1 , parentID= 0
355   0:0:10.509552016 DEBUG (5): New parent for NodeID= 5 : curdepth= 2 , parentID= 1
356   0:0:10.509552016 DEBUG (2): New parent for NodeID= 2 : curdepth= 2 , parentID= 1
357   0:0:10.511367801 DEBUG (7): New parent for NodeID= 7 : curdepth= 2 , parentID= 3
358   0:0:10.511367801 DEBUG (6): New parent for NodeID= 6 : curdepth= 2 , parentID= 3
359   0:0:10.517913784 DEBUG (8): New parent for NodeID= 8 : curdepth= 2 , parentID= 4
360   0:0:13.000000010 DEBUG (0): FinishedRouting!
361   0:0:13.000000010 DEBUG (1): FinishedRouting!
362   0:0:13.000000010 DEBUG (2): FinishedRouting!
363   0:0:13.000000010 DEBUG (3): FinishedRouting!
364   0:0:13.000000010 DEBUG (4): FinishedRouting!
365   0:0:13.000000010 DEBUG (5): FinishedRouting!
366   0:0:13.000000010 DEBUG (6): FinishedRouting!
367   0:0:13.000000010 DEBUG (7): FinishedRouting!
368   0:0:13.000000010 DEBUG (8): FinishedRouting!
```

1ο παράδειγμα - routing

Στην παραπάνω εικόνα εκτυπώνονται αρχικά για κάθε κόμβο το βάθος του και ο πατέρας που διάλεξε. Στη συνέχεια, όταν χτυπήσει ο μετρητής των τριών δευτερολέπτων που διατίθενται συνολικά για το routing, κάθε κόμβος ενημερώνει ότι τελείωσε την φάση του routing και εισέρχεται στην φάση αποστολής μετρήσεων. Με βάση την παραπάνω πληροφορία μπορεί να σχεδιαστεί το δέντρο που δημιουργήθηκε με το πέρας του routing.



1ο παράδειγμα - δένδρο δρομολόγησης

Με άσπρο εμφανίζεται η ρίζα, στο επίπεδο με βάθος μηδέν. Με ανοικτό γκρι είναι οι κόμβοι 1, 3, 4 που ανήκουν στο επίπεδο με βάθος ένα, ενώ με σκούρο γκρί είναι οι κόμβοι 2, 5, 6, 7, 8 που ανήκουν στο επίπεδο με βάθος δύο.

Στη συνέχεια παρουσιάζεται η φάση υπολογισμού και μετάδοσης μετρήσεων από τα φύλλα προς τη ρίζα.

```
369  0:0:59.255859385 DEBUG (2): NodeID = 2 curdepth= 2
370  0:0:59.255859385 DEBUG (2): Starting Data transmission to parent!
371  0:0:59.255859385 DEBUG (2): measurement is: 21
372  0:0:59.255859385 DEBUG (2): Node has sum: 21, count: 1, max: 21
373  0:0:59.264511088 DEBUG (1): Receive from child: 2 values - sum:21, count: 1, max: 21
374  0:0:59.264648448 DEBUG (5): NodeID = 5 curdepth= 2
375  0:0:59.264648448 DEBUG (5): Starting Data transmission to parent!
376  0:0:59.264648448 DEBUG (5): measurement is: 6
377  0:0:59.264648448 DEBUG (5): Node has sum: 6, count: 1, max: 6
378  0:0:59.267578135 DEBUG (6): NodeID = 6 curdepth= 2
379  0:0:59.267578135 DEBUG (6): Starting Data transmission to parent!
380  0:0:59.267578135 DEBUG (6): measurement is: 13
381  0:0:59.267578135 DEBUG (6): Node has sum: 13, count: 1, max: 13
382  0:0:59.267852795 DEBUG (1): Receive from child: 5 values - sum:6, count: 1, max: 6
383  0:0:59.271484385 DEBUG (7): NodeID = 7 curdepth= 2
384  0:0:59.271484385 DEBUG (7): Starting Data transmission to parent!
385  0:0:59.271484385 DEBUG (7): measurement is: 34
386  0:0:59.271484385 DEBUG (7): Node has sum: 34, count: 1, max: 34
387  0:0:59.273223897 DEBUG (3): Receive from child: 7 values - sum:34, count: 1, max: 34
388  0:0:59.273437510 DEBUG (8): NodeID = 8 curdepth= 2
389  0:0:59.273437510 DEBUG (8): Starting Data transmission to parent!
390  0:0:59.273437510 DEBUG (8): measurement is: 41
391  0:0:59.273437510 DEBUG (8): Node has sum: 41, count: 1, max: 41
392  0:0:59.275695784 DEBUG (3): Receive from child: 6 values - sum:13, count: 1, max: 13
393  0:0:59.278747558 DEBUG (4): Receive from child: 8 values - sum:41, count: 1, max: 41
394  0:0:59.502929697 DEBUG (1): NodeID = 1 curdepth= 1
395  0:0:59.502929697 DEBUG (1): Starting Data transmission to parent!
396  0:0:59.502929697 DEBUG (1): measurement is: 14
397  0:0:59.502929697 DEBUG (1): Child 2 has sum: 21, count: 1 and max: 21
398  0:0:59.502929697 DEBUG (1): Child 5 has sum: 6, count: 1 and max: 6
399  0:0:59.502929697 DEBUG (1): Node has sum: 41, count: 3, max: 21
400  0:0:59.505004900 DEBUG (0): Receive from child: 1 values - sum:41, count: 3, max: 21
401  0:0:59.508789072 DEBUG (3): NodeID = 3 curdepth= 1
402  0:0:59.508789072 DEBUG (3): Starting Data transmission to parent!
403  0:0:59.508789072 DEBUG (3): measurement is: 42
404  0:0:59.508789072 DEBUG (3): Child 7 has sum: 34, count: 1 and max: 34
405  0:0:59.508789072 DEBUG (3): Child 6 has sum: 13, count: 1 and max: 13
406  0:0:59.508789072 DEBUG (3): Node has sum: 89, count: 3, max: 42
407  0:0:59.511383071 DEBUG (0): Receive from child: 3 values - sum:89, count: 3, max: 42
408  0:0:59.511718760 DEBUG (4): NodeID = 4 curdepth= 1
409  0:0:59.511718760 DEBUG (4): Starting Data transmission to parent!
410  0:0:59.511718760 DEBUG (4): measurement is: 49
411  0:0:59.511718760 DEBUG (4): Child 8 has sum: 41, count: 1 and max: 41
412  0:0:59.511718760 DEBUG (4): Node has sum: 90, count: 2, max: 49
413  0:0:59.518737782 DEBUG (0): Receive from child: 4 values - sum:90, count: 2, max: 49
414  0:0:59.750000010 DEBUG (0): NodeID = 0 curdepth= 0
415  0:0:59.750000010 DEBUG (0): measurement is: 7
416  0:0:59.750000010 DEBUG (0): Child 1 has sum: 41, count: 3 and max: 21
417  0:0:59.750000010 DEBUG (0): Child 3 has sum: 89, count: 3 and max: 42
418  0:0:59.750000010 DEBUG (0): Child 4 has sum: 90, count: 2 and max: 49
419  0:0:59.750000010 DEBUG (0): Node has sum: 227, count: 9, max: 49
420  0:0:59.750000010 DEBUG (0):
421  0:0:59.750000010 DEBUG (0): FINAL RESULTS: AVG: 25.22 , Max: 49
422  0:0:59.750000010 DEBUG (0):
```

1ο παράδειγμα - αποστολή μηνυμάτων

Αρχικά ξεκινά την αποστολή μηνυμάτων ο κόμβος 2, ο οποίος παίρνει τη μέτρηση 21, υπολογίζει τις συναθροιστικές τιμές και στέλνει στον πατέρα του το μήνυμα. Απευθείας, ο κόμβος 1 λέει ότι έλαβε από τον κόμβο 2 το μήνυμα. Στη συνέχεια ο κόμβος 5 μετράει 6 και στέλνει μήνυμα στον 1, ο οποίος το λαμβάνει. Το ίδιο συμβαίνει και για τους κόμβους 6 και 7 με τιμές 13 και 34 αντίστοιχα, με τον κόμβο 3 να λαμβάνει αυτές τις τιμές. Ομοίως ο κόμβος 8 μετράει την τιμή 41 και τη στέλνει στον πατέρα του, τον 4, ο οποίος λαμβάνει το μήνυμα. Στο σημείο αυτό τελείωσε η μετάδοση των κόμβων του επιπέδου 2, και ακολουθούν οι κόμβοι του επιπέδου 1.

Ξεκινάει ο κόμβος 1, μετρώντας 14. Ανατρέχει στις τιμές των παιδιών του, και με βάση αυτές υπολογίζει τις συναθροιστικές τιμες, τις οποίες και αποστέλλει στη ρίζα. Ομοίως κάνουν οι κόμβοι 3 με μέτρηση 42 και 4 με μέτρηση 49. Η ρίζα εν τέλει, μετράει 7, και λαμβάνει από τα τρία παιδιά της τις τιμές τους. Έτσι υπολογίζει τα τελικά αποτελέσματα τα οποία και εκτυπώνει.

Με βάση τις παραπάνω μετρήσεις αυτά προκύπτουν:

MAX τιμή είναι όντως το 49,

count είναι όντως 9, διότι έχουμε 9 κόμβους

SUM είναι $21 + 6 + 13 + 34 + 41 + 14 + 42 + 49 + 7 = 227$

$$AVG = \frac{SUM}{count} = \frac{227}{9} = 25.22$$

Άρα όντως τα επερωτήματα υπολογίστηκαν σωστά.

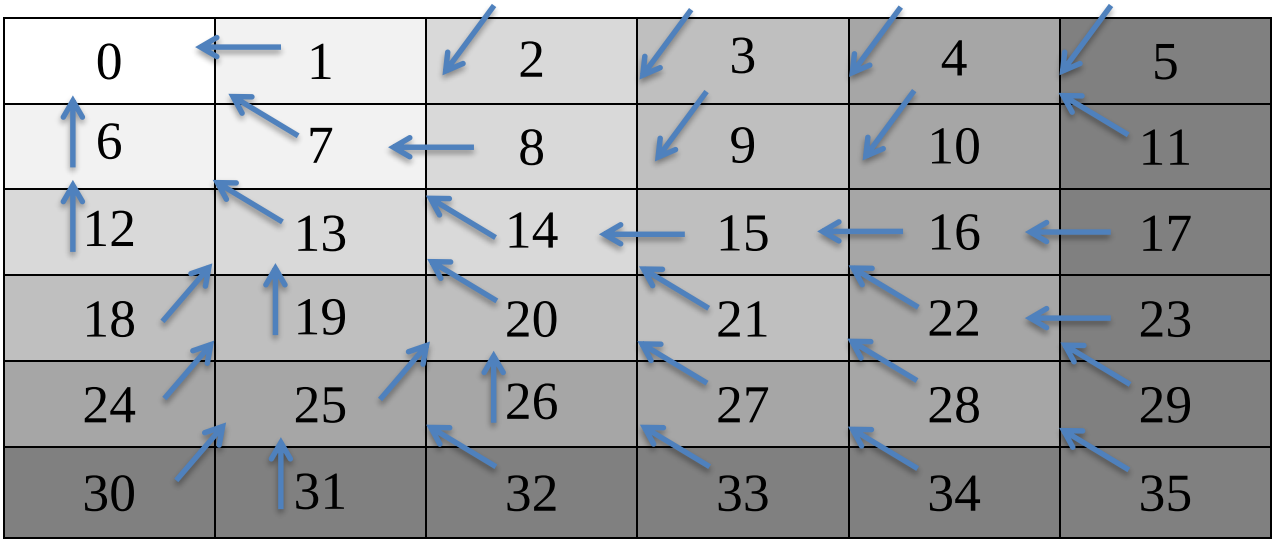
Παράδειγμα λειτουργίας 2

Το δεύτερο παράδειγμα έγινε με δίκτυο 6x6 κόμβων και εμβέλεια 1.5. Αρχικά εμφανίζεται το routing:

```
1447  0:0:10.250000010 DEBUG (0): ####################################
1448  0:0:10.250000010 DEBUG (0): #######   ROUND   1    ##############
1449  0:0:10.250000010 DEBUG (0): ####################################
1450  0:0:10.251800556 DEBUG (7): New parent for NodeID= 7 : curdepth= 1 , parentID= 0
1451  0:0:10.251800556 DEBUG (6): New parent for NodeID= 6 : curdepth= 1 , parentID= 0
1452  0:0:10.251800556 DEBUG (1): New parent for NodeID= 1 : curdepth= 1 , parentID= 0
1453  0:0:10.506469725 DEBUG (13): New parent for NodeID= 13 : curdepth= 2 , parentID= 6
1454  0:0:10.506469725 DEBUG (12): New parent for NodeID= 12 : curdepth= 2 , parentID= 6
1455  0:0:10.508560167 DEBUG (8): New parent for NodeID= 8 : curdepth= 2 , parentID= 7
1456  0:0:10.508560167 DEBUG (14): New parent for NodeID= 14 : curdepth= 2 , parentID= 7
1457  0:0:10.508560167 DEBUG (2): New parent for NodeID= 2 : curdepth= 2 , parentID= 7
1458  0:0:10.758209246 DEBUG (18): New parent for NodeID= 18 : curdepth= 3 , parentID= 13
1459  0:0:10.758209246 DEBUG (20): New parent for NodeID= 20 : curdepth= 3 , parentID= 13
1460  0:0:10.758209246 DEBUG (19): New parent for NodeID= 19 : curdepth= 3 , parentID= 13
1461  0:0:10.760772719 DEBUG (9): New parent for NodeID= 9 : curdepth= 3 , parentID= 14
1462  0:0:10.760772719 DEBUG (21): New parent for NodeID= 21 : curdepth= 3 , parentID= 14
1463  0:0:10.760772719 DEBUG (15): New parent for NodeID= 15 : curdepth= 3 , parentID= 14
1464  0:0:10.763748165 DEBUG (3): New parent for NodeID= 3 : curdepth= 3 , parentID= 8
1465  0:0:11.011032117 DEBUG (25): New parent for NodeID= 25 : curdepth= 4 , parentID= 20
1466  0:0:11.011032117 DEBUG (27): New parent for NodeID= 27 : curdepth= 4 , parentID= 20
1467  0:0:11.011032117 DEBUG (26): New parent for NodeID= 26 : curdepth= 4 , parentID= 20
1468  0:0:11.013885494 DEBUG (24): New parent for NodeID= 24 : curdepth= 4 , parentID= 19
1469  0:0:11.016769405 DEBUG (22): New parent for NodeID= 22 : curdepth= 4 , parentID= 15
1470  0:0:11.016769405 DEBUG (16): New parent for NodeID= 16 : curdepth= 4 , parentID= 15
1471  0:0:11.016769405 DEBUG (10): New parent for NodeID= 10 : curdepth= 4 , parentID= 15
1472  0:0:11.019119244 DEBUG (4): New parent for NodeID= 4 : curdepth= 4 , parentID= 9
1473  0:0:11.021163911 DEBUG (28): New parent for NodeID= 28 : curdepth= 4 , parentID= 21
1474  0:0:11.265213019 DEBUG (31): New parent for NodeID= 31 : curdepth= 5 , parentID= 25
1475  0:0:11.265213019 DEBUG (30): New parent for NodeID= 30 : curdepth= 5 , parentID= 25
1476  0:0:11.265213019 DEBUG (32): New parent for NodeID= 32 : curdepth= 5 , parentID= 25
1477  0:0:11.266983028 DEBUG (33): New parent for NodeID= 33 : curdepth= 5 , parentID= 26
1478  0:0:11.269577041 DEBUG (23): New parent for NodeID= 23 : curdepth= 5 , parentID= 22
1479  0:0:11.269577041 DEBUG (29): New parent for NodeID= 29 : curdepth= 5 , parentID= 22
1480  0:0:11.270034801 DEBUG (5): New parent for NodeID= 5 : curdepth= 5 , parentID= 10
1481  0:0:11.271774260 DEBUG (34): New parent for NodeID= 34 : curdepth= 5 , parentID= 27
1482  0:0:11.272415158 DEBUG (17): New parent for NodeID= 17 : curdepth= 5 , parentID= 16
1483  0:0:11.277999853 DEBUG (11): New parent for NodeID= 11 : curdepth= 5 , parentID= 4
1484  0:0:11.278915389 DEBUG (35): New parent for NodeID= 35 : curdepth= 5 , parentID= 28
```

2ο παράδειγμα - routing

Όπως προκύπτει από τον παραπάνω πίνακα ο κάθε ένας από τους 35 κόμβους διαλέγει τον πατέρα του και εισέρχεται στο αντίστοιχο επίπεδο με βάθος +1 από αυτό του πατέρα του. Το δέντρο που προκύπτει παρουσιάζεται στον ακόλουθο πίνακα:



1ο παράδειγμα - δένδρο δρομολόγησης

Το δέντρο που δημιουργείται εν τέλει είναι 5 επιπέδων. Η αποστολή και η προώθηση των μηνυμάτων προς τη ρίζα δεν παρουσιάζεται αναλυτικά λόγω του μεγάλου όγκου πληροφορίας. Παρακάτω ωστόσο εμφανίζονται τα αποτελέσματα που εκτυπώνει η ρίζα για τρεις διαδοχικούς γύρους, τα οποία παρουσιάζουν ενδιαφέρον.

```
1722   0:0:59.750000010 DEBUG (0): NodeID = 0 curdepth= 0
1723   0:0:59.750000010 DEBUG (0): measurement is: 7
1724   0:0:59.750000010 DEBUG (0): Child 1 has sum: 14, count: 1 and max: 14
1725   0:0:59.750000010 DEBUG (0): Child 6 has sum: 294, count: 13 and max: 43
1726   0:0:59.750000010 DEBUG (0): Child 7 has sum: 459, count: 17 and max: 49
1727   0:0:59.750000010 DEBUG (0): Node has sum: 774, count: 32, max: 49
1728   0:0:59.750000010 DEBUG (0):
1729   0:0:59.750000010 DEBUG (0): FINAL RESULTS: AVG: 24.19 , Max: 49
1730   0:0:59.750000010 DEBUG (0):
```

2ο παράδειγμα - 1ος γύρος αποστολής μηνυμάτων

Στο γύρο 1 η ρίζα υπολογίζει συνολικό count 32. Αυτό πάει να πει ότι κατά το routing χάθηκαν τα μηνύματα τεσσάρων κόμβων, κάτι αναμενόμενο, αφού παρά τη βελτιστοποίηση που αναλύθηκε προηγουμένως, υπάρχει πιθανότητα να προκύψουν συγκρούσεις. Συγκεκριμένα προκύπτει ότι χάθηκε ένας κόμβος στο υποδέντρο του παιδιού 6 και άλλοι τρεις στο υποδέντρο του παιδιού 7.

```
1935   0:1:59.750000010 DEBUG (0): NodeID = 0 curdepth= 0
1936   0:1:59.750000010 DEBUG (0): measurement is: 39
1937   0:1:59.750000010 DEBUG (0): Child 1 has sum: 28, count: 1 and max: 28
1938   0:1:59.750000010 DEBUG (0): Child 6 has sum: 397, count: 13 and max: 49
1939   0:1:59.750000010 DEBUG (0): Child 7 has sum: 418, count: 19 and max: 48
1940   0:1:59.750000010 DEBUG (0): Node has sum: 882, count: 34, max: 49
1941   0:1:59.750000010 DEBUG (0):
1942   0:1:59.750000010 DEBUG (0): FINAL RESULTS: AVG: 25.94 , Max: 49
1943   0:1:59.750000010 DEBUG (0):
```

2ο παράδειγμα - 2ος γύρος αποστολής μηνυμάτων

Στο δεύτερο γύρω εμφανίζεται count 34, το οποίο σημαίνει ότι δύο από τους τέσσερις κόμβους που απέτυχαν να στείλουν στον πρώτο γύρο, στείλανε επιτυχώς στον δεύτερο. Αυτό δε σημαίνει ότι όλοι οι υπόλοιποι κόμβοι στείλανε επιτυχώς σε αυτό το γύρο, όμως όπως αναφέρθηκε, ο κάθε κόμβος κρατάει πληροφορία για τα παιδιά του. Επομένως στον υπολογισμό των επερωτημάτων, εάν χάθηκαν κάποιες τιμές κόμβων, χρησιμοποιήθηκαν οι προηγούμενες τιμές τους.

```
2363   0:3:59.750000010 DEBUG (0): NodeID = 0 curdepth= 0
2364   0:3:59.750000010 DEBUG (0): measurement is: 14
2365   0:3:59.750000010 DEBUG (0): Child 1 has sum: 28, count: 1 and max: 28
2366   0:3:59.750000010 DEBUG (0): Child 6 has sum: 288, count: 15 and max: 41
2367   0:3:59.750000010 DEBUG (0): Child 7 has sum: 509, count: 19 and max: 46
2368   0:3:59.750000010 DEBUG (0): Node has sum: 839, count: 36, max: 46
2369   0:3:59.750000010 DEBUG (0):
2370   0:3:59.750000010 DEBUG (0): FINAL RESULTS: AVG: 23.31 , Max: 46
2371   0:3:59.750000010 DEBUG (0):
```

4ος γύρος αποστολής μηνυμάτων

Τελικώς, επιτυγχάνουν να στείλουν και οι άλλοι δύο κόμβοι στον γύρο 4. Έτσι το count πλέον είναι 36, δηλαδή υπάρχει πλέον πληροφορία για κάθε κόμβο στο δίκτυο.

Κατανομή εργασίας

Όλα τα παραπάνω κομμάτια που αποτελούν το πρώτο μέρος του project, αλλά και η παρούσα αναφορά, έγιναν από κοινού με συνεργασία σε κάθε στάδιο ξεχωριστά κατανέμοντας έτσι δίκαια το φόρτο εργασίας.