Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Αποδοτική δρομολόγηση σε περιβάλλοντα χαμηλής ενεργειακής κατανάλωσης

Μουστακάκης Κώστας
ΑΜ: 4223

Επιβλέπων καθηγητής: Παύλος Σπυράκης
Συνεπιβλέπων καθηγητής: Ιωάννης Χατζηγιαννάκης

Ευχαριστίες

Για την βοήθεια και συνεργασία θα ήθελα να ευχαριστήσω τους:

Σπυράκη Παύλο, Καθηγητής Τμήματος Μηχανικών Η/Υ & Πληροφορικής , Πανεπιστήμιο Πατρών.

Χατζηγιαννάκη Ιωάννη, Διευθυντής στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ & Ακαδημαϊκός Συνεργάτης του Τμήματος Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστήμιο Πατρών.

Αντωνίου Αθανάσιο, Ερευνητής στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ & Ακαδημαϊκός Συνεργάτης του Τμήματος Μηχανικών Η/Υ & Πληροφορικής , Πανεπιστήμιο Πατρών.

Αμαξιλάτη Δημήτρη, Ερευνητής στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ & Ακαδημαϊκός Συνεργάτης του Τμήματος Μηχανικών Η/Υ & Πληροφορικής , Πανεπιστήμιο Πατρών.

Περίληψη εργασίας:

Ο σκοπός της ακόλουθης διπλωματικής εργασίας χωρίζεται σε δυο μέρη: Την υλοποίηση του αρχικού πρωτοκόλλου όπως σχεδιάστηκε από την εταιρία Cisco και την βελτιστοποίηση του πρωτοκόλλου δρομολόγησης ως προς την πολυπλοκότητα του αλλά και ως προς την διαχείριση μνήμης. Επίσης στα πλαίσια της διπλωματικής περάσαμε από το θεωρητικό μέρος στο πειραματικό σε φυσικές συσκευές των οποίων τα χαρακτηριστικά θα αναφερθούν εκτενέστερα στα επόμενα κεφάλαια.

Στο πρώτο κεφάλαιο θα αναφέρουμε τον λόγω για τον οποίο δημιουργήθηκε αρχικά το πρωτόκολλο αλλά και ποιο νέο πρόβλημα εισάγει.

Στο δεύτερο κεφάλαιο θα αναλυθούν οι συσκευές που χρησιμοποιήθηκαν για την εκτέλεση πειραμάτων, οι δυνατότητες τους και οι περιορισμοί που εισάγουν.

Στο τρίτο κεφάλαιο θα εμβαθύνουμε στις λειτουργίες του πρότυπου πρωτοκόλλου και θα κάνουμε μια ανάλυση της πολυπλοκότητας του αλγορίθμου που χρησιμοποιεί αλλά και για την μνήμη που απαιτεί για την λειτουργία του.

Έπειτα, στο επόμενο κεφάλαιο θα παρουσιαστεί η λύση των προβλημάτων που εισήγαγε το πρότυπο πρωτόκολλο και μαζί με την λύση θα παρουσιαστεί εκτενής η ανάλυση και λειτουργία του βελτιωμένου πρωτοκόλλου.

Θα παρουσιάσουμε διάφορα αποτελέσματα από πειράματα, τόσο από εξομοίωση όσο και από πραγματικές συσκευές και θα μιλήσουμε για το τελικό συμπέρασμα του πρωτοκόλλου που δημιούργησα σε αντίθεση με το πρότυπο πρωτόκολλο.

Περιεχόμενα:

Πινακας εικονων: Σελιδα:

Κεφάλαιο 1 – Εισαγωγή

1.1 Κίνητρο και σημασία της εργασίας

Κίνητρο για την παρούσα διπλωματική εργασία, στάθηκαν τα κατανεμημένα ασύρματα δίκτυα χαμηλής κατανάλωσης[1]. Τα δίκτυα χαμηλής κατανάλωσης έχουν αρχίσει να χρησιμοποιούνται όλο και περισσότερο, κυρίως σε βιομηχανικές εγκαταστάσεις, ερευνητικές μονάδες και σύντομα θα εισέλθουν στην καθημερινότητα μας. Με την εξέλιξη του απλού νοικοκυριού σε έξυπνο σπίτι αυτά τα δίκτυα θα βρίσκονται πλέον σε κάθε οικιακό χώρο, παρέχοντας μας ευελιξία, ασφάλεια και εύκολο χειρισμό. Καθώς αυτά τα δίκτυα μεγαλώνουν σε έκταση, τόσο από άποψη συσκευών όσο και γεωγραφική, καταλαβαίνουμε πως τα ήδη υπάρχοντα πρωτόκολλα δρομολόγησης αρχίζουν να αποτυγχάνουν να αντεπεξέλθουν σε αυτή την εξέλιξη κάνοντας την επικοινωνία μεταξύ των συσκευών πιο δύσκολη έως αδύνατη.

Για αυτό τον λόγο απαιτείται η δημιουργία ενός νέου πρωτοκόλλου το οποίο θα μπορέσει να αντεπεξέλθει στην εξέλιξη που προαναφέρθηκε. Υπάρχουν πολλές προτάσεις για δρομολόγηση[2] αλλά οι συγκεκριμένες απαιτήσεις των δικτύων χαμηλής κατανάλωσης δημιούργησαν την ανάγκη για ένα πρωτόκολλο σαν το RPL.

Το νέο αυτό πρωτόκολλο θα πρέπει να εισάγει έναν νέο τρόπο δρομολόγησης. Ο αλγόριθμος δρομολόγησης θα πρέπει να είναι φτιαγμένος από την αρχή και να μην βασίζεται στον τρόπο σχεδιασμοί των πρωτοκόλλων δρομολόγησης που ήδη υπάρχουν. Ο λόγος που θα πρέπει να ξεχωρίσει από τα προγενέστερα πρωτοκολλά είναι διότι με την επέκταση των κατανεμημένων δικτύων επέρχεται επέκταση των απαιτήσεων των ήδη υπάρχων πρωτοκόλλων σε ενεργεία, επεξεργαστική ισχύ και χρήση μνήμης. Η χρήση ήδη υπάρχων πρωτοκόλλων σε τέτοιες συσκευές οι οποίες έχουν αρκετά περιορισμένη μνήμη και επεξεργαστική ισχύ θα είχε ως αποτέλεσμα μια πολύ αργή και μεγάλου ενεργειακού κόστους δρομολόγηση πακέτων μέσα στο δίκτυο. Επιπλέον, κάποια από αυτά τα πρωτόκολλα θα είναι αδύνατον να λειτουργήσουν σε ορισμένες συσκευές, όταν εξαιτίας του μεγέθους του δικτύου, οι απαιτήσεις του πρωτοκόλλου σε μνήμη θα ξεπεράσουν την διαθέσιμη μνήμη που παρέχουν οι συσκευές. Αποτελεί λοιπόν πρόκληση η δημιουργία ενός πρωτοκόλλου στο οποίο οι απαιτήσεις του αυτές δεν θα επηρεάζονται από το μέγεθος του δικτύου αλλά να είναι πάντα σταθερές ώστε το ίδιο πρωτόκολλο να μπορεί να λειτουργήσει από το πιο μικρο οικιακό δίκτυο έως το μεγαλύτερο και σύνθετο βιομηχανικό δίκτυο.

1.2 Στόχος

Έχοντας παρουσιάσει τα κίνητρα της εργασίας, ως στόχος τίθεται η ανακάλυψη ενός αλγορίθμου δρομολόγησης ο οποίος έχει σταθερή απαίτηση σε επεξεργαστική ισχύ και αρκετά μικρή και σταθερή απαίτηση σε μνήμη. Αφότου βρεθεί ένας τέτοιος αλγόριθμος θα πρέπει να ενσωματωθεί σε ένα πρωτόκολλο ώστε να είναι λειτουργικός. Οι αλγόριθμοι που έχουν ήδη υλοποιηθεί οι οποίοι υποστηρίζουν επικοινωνία από σημείο σε σημείο (point-to-point), κάνουν χρήση πινάκων δρομολόγησης. Όμως οι πίνακες δρομολόγησης μεγαλώνουν όσο μεγαλώνει το δίκτυο και η εύρεση εγγραφών μέσα στους πίνακες γίνετε πιο κοστοβορα από άποψη επεξεργαστικής ισχύς.

Άρα λοιπόν ως βασικός στόχος τίθεται η δημιουργία ενός αλγορίθμου

διευθυνσιοδότησης και δρομολόγησης όπου δεν θα χρησιμοποιεί πίνακες δρομολόγησης αλλά θα μπορεί να υποστηρίξει επικοινωνία από σημείο σε σημείο. Έπειτα αυτός ο αλγόριθμος θα πρέπει να εξελιχθεί σε ένα πλήρες πρωτόκολλο δρομολόγησης και επικοινωνίας.

1.3 Συνεισφορά

Το πρωτόκολλο που αναπτύχθηκε στα πλαίσια της διπλωματικής παρουσιάζει έναν νέο τρόπο διευθυνσιοδότησης και δρομολόγησης μηνυμάτων μέσα στο δίκτυο. Το πρωτόκολλο εξαλείφει την αναγκαιότητα χρήσης πινάκων δρομολόγησης για να επιτευχθεί επικοινωνία από σημείο σε σημείο. Χάρη στο ότι δεν χρησιμοποιούνται πίνακες δρομολόγησης, το πρωτόκολλο είναι πιο αποδοτικό από άποψη επεξεργαστικής ισχύς (άρα και ενεργειακής κατανάλωσης) και μνήμης σε σχέση με πρωτόκολλα τα οποία χρησιμοποιούν πίνακες δρομολόγησης. Το πρωτόκολλο δοκιμάστηκε και σε εξομοιωτή αλλά και σε δίκτυο πραγματικών συσκευών ώστε να εξασφαλιστεί η σωστή λειτουργία του πρωτοκόλλου.

1.4 Δομή

Στο κεφάλαιο που ακολουθεί παρουσιάζω ένα νέο πρωτόκολλο που δημιουργήθηκε από την Cisco, το Routing Protocol for Lossy and low power networks ή RPL. Το νέο αυτό πρωτόκολλο εισάγει αρκετές καινοτομίες για την δημιουργία και σύσταση ενός δικτύου κατανεμημένων συσκευών χαμηλής κατανάλωσης αλλά δεν μπορεί να λύσει το πρόβλημα με την μεγέθυνση των απαιτήσεων όπου αναφέρθηκε πιο πάνω. Θα αναλύσω τις βασικές λειτουργίες του καθώς και τον αλγόριθμο δρομολόγησης πακέτων ο οποίος λειτουργεί με τ χρήση πινάκων δρομολόγησης. Το πρωτόκολλο RPL δεν έχει υλοποιηθεί, υπάρχει μέχρι τώρα ως προσχέδιο το οποίο χρησιμοποίησα ως αναφορά για να το υλοποιήσω εγώ στα πλαίσια της διπλωματικής μου ώστε να μπορώ να συγκρίνω την αρχική σχεδίαση με πινάκες δρομολόγησης με τον νέο αλγόριθμο δρομολόγησης όπου ανέπτυξα.

Στο κεφάλαιο 3 ακολουθεί η ανάλυση του αλγορίθμου όπου ανέπτυξα,και την αρχή όπου βασίζεται. Μαζί θα αναλυθούν και οι λόγοι που η διευθυνσιοδότηση γίνετε με έναν συγκεκριμένο τρόπο αλλά και πως επιτελούμαστε από την συγκεκριμένη διευθυνσιοδότηση. Επίσης θα αναλυθεί η ενσωμάτωση του νέου αλγορίθμου δρομολόγησης στο πρωτόκολλο RPL, ποια μέρη του αρχικού πρωτοκόλλου έμειναν ίδια αλλά και ποια άλλαξαν. Τέλος θα αναφερθώ σε κάποιες νέες λειτουργίες όπου εισήγαγα στο βελτιωμένο πρωτόκολλο.

Στο κεφάλαιο 4 θα παρουσιάσω την θεωρητική ανάλυση και αξιολόγηση της πολυπλοκότητας των αλγορίθμων δρομολόγησης, την πολυπλοκότητα επικοινωνίας και τις απαιτήσεις σε μνήμη των δυο πρωτοκόλλων. Έπειτα θα υπάρξει μια μικρή αναφορά στα σενάρια των πειραμάτων που εκτελεσα, απο τα οποία κάποια έγιναν σε πραγματικό περιβάλλον και άλλα σε περιβάλλον εξομοιωτή, τα αποτελέσματα τους και συμπεράσματα.

Στο κεφάλαιο 5 θα γίνει μια σύντομη αναφορά σε άλλες δουλειές που έχουν γίνει με βάση το πρωτόκολλο RPL και τέλος στο κεφάλαιο 6 θα αναφερθούν κάποια τρέχων ζητήματα καθώς και μελλοντική δουλειά που θα μπορούσε να γίνει πάνω στο βελτιωμένο πρωτόκολλο το οποίο ανέπτυξα.

Κεφάλαιο 2 – Το πρωτόκολλο δρομολόγησης RPL

2.1 Εισαγωγή και ορισμοί του RPL

2.1.1 Εισαγωγή

Τα χαμηλής ενέργειας και αδύνατα κατανεμημένα δίκτυα (Low power and Lossy Networks – LLNs) αποτελούνται από συσκευές περιορισμένων δυνατοτήτων, τόσο από άποψη επεξεργαστικής ισχύς και μνήμης, αλλά και από άποψη ενέργειας. Ως επακόλουθο, οι συσκευές αυτές είναι διασυνδεδεμένες με μη έμπιστους διαύλους επικοινωνίας. Άλλο ένα χαρακτηριστικό αυτών των δικτύων είναι ότι η επικοινωνία μεταξύ τους είναι κόμβος προς πλήθος κόμβων ή πλήθος κόμβων προς κόμβο. Στόχος της δημιουργίας του RPL[3] είναι να λύσει τα προβλήματα επικοινωνίας που δημιουργούνται σε αυτά τα δίκτυα να προσφέρει έναν τρόπο επικοινωνίας από κόμβο σε κόμβο και να μορφοποιήσει το δίκτυο βάση κάποιων μετρικών.

2.1.2 Ο σκοπός του RPL

Το RPL (Routing Protocol for Low power and Lossy Networks), δημιουργήθηκε για να λύσει το πρόβλημα που εισάγεται από τα κατανεμημένα δίκτυα ασύρματης επικοινωνίας: Την αδυναμία του δικτύου να παρέχει έμπιστους διαύλους επικοινωνίας μεταξύ των συσκευών του δικτύου, την δυναμική αλλαγή της μορφολογίας του δικτύου και την ανάγκη για επικοινωνία χαμηλού ενεργειακού κόστους.

2.1.3 Οι καινοτομίες του RPL

Το RPL εισήγαγε μια πιο έμπιστη γραμμή επικοινωνίας μεταξύ των κόμβων του δικτύου καθώς πλέον στο κατανεμημένο δίκτυο ο κάθε κόμβος επέλεγε ποιοι κόμβοι θα ανήκαν στην γειτονία του. Το κατάφερε αυτό εισάγοντας την έννοια της μετρικής. Ο κάθε κόμβος πλέον διαλέγει την γειτονία του βάση των ανάλογων μετρικών. Οι μετρικές που χρησιμοποιούνται διαφέρουν σε κάθε υλοποίηση αλλά οι δυο πιο πολύ χρησιμοποιούμενες είναι οι μετρικές ενέργειας/κόστους και ενέργειας/ασφάλειας. Η άλλη καινοτομία του RPL είναι πως χρησιμοποιεί ήδη υπάρχουσες τεχνολογίες όπως η IPv6. Η τελευταία, αλλά ίσως και η μεγαλύτερη καινοτομία που εισάγει είναι η δυνατότητα επικοινωνίας από κόμβο προς κόμβο. Μέχρι τώρα όλα τα κατανεμημένα δίκτυα είχαν κυρίως επικοινωνία από κόμβο προς πηγή (root node) ενώ όσα υλοποιούσαν λειτουργίες επικοινωνίας από κόμβο σε κόμβο, η υλοποίηση τους ήταν με αρκετά μεγάλο ενεργειακό κόστος με αποτέλεσμα να κάνει την επικοινωνία από κόμβο σε κόμβο ακριβή, με μεγάλο αριθμό πακέτων άρα και σφαλμάτων και πάνω από όλα μη αποδοτική. Ένα παράδειγμα μιας τέτοιας επικοινωνίας είναι και ο αλγόριθμος floodset στον οποίο κάθε κόμβος αναμεταδίδει το μήνυμα που έλαβε με τελικό σκοπό να παραληφθεί στον κόμβο-προορισμό, βεβαία αυτός ο τρόπος επικοινωνίας δεν εγγυάται πάντα την επιτυχή μετάδοση ενός μηνύματος.

2.1.4 Το νέο πρόβλημα που δημιουργείται

Το πρότυπο RPL εισάγει την επικοινωνία από κόμβο σε κόμβο (peer-to-peer) χρησιμοποιώντας το ήδη υπάρχον πρωτόκολλο IPv6. Για να επιτευχθεί δρομολόγηση των

πακέτων, κάθε κόμβος του δικτύου θα πρέπει να έχει πινάκες δρομολόγησης. Επακόλουθα, όσο μεγαλύτερο είναι το δίκτυο μας, τόσο μεγαλύτερους πινάκες δρομολόγησης θα χρειαστούμε. Επίσης, εκτός από το κόστος σε μνήμη που δημιουργείται από την απαίτηση να υπάρχουν πινάκες δρομολόγησης σε κάθε κόμβο, η προσπέλαση του εκάστοτε πινάκα σε έναν κόμβο ώστε να βρεθεί ο επόμενος προορισμός του μηνύματος, γίνεται πιο δαπανηρή όσο μεγαλώνει το δίκτυο. Καθώς το πρότυπο RPL δημιουργήθηκε για χρίση πάνω σε κατανεμημένα δίκτυα από τα οποία πολλά δίκτυα αποτελούνται από συσκευές πολύ περιορισμένων δυνατοτήτων, το πρότυπο RPL εισάγει στο δίκτυο αρχικά έναν περιορισμό στον αριθμό των συσκευών που θα βρίσκονται σε αυτό το δίκτυο (λόγω της περιορισμένης μνήμης δεν θα μπορεί να υπάρχουν μεγάλοι πινάκες δρομολόγησης) και εν συνεχεία καθυστέρηση στην διάδοση μηνυμάτων (καθώς θα απαιτείται ο κάθε κόμβος να προσπελάσει για κάθε μήνυμα τους πινάκες δρομολόγησης).

Το πρόβλημα της περιορισμένης μνήμης γίνετε πιο αισθητό όταν η υλοποίηση του πρότυπου RPL γίνεται πάνω σε μια πλατφόρμα, στην περίπτωση μας η πλατφόρμα που έχουμε χρησιμοποιήσει είναι η wiselib[4]. Η πλατφόρμα που χρησιμοποιήσαμε, επιλέχθηκε γιατί προσφέρει ένα περιβάλλον ανάπτυξης κωδικά το οποίο μπορεί να εκτελεστεί στις πιο γνώστες και πιο συχνά χρησιμοποιούμενες συσκευές, αλλά και σε υπολογιστή. Η συγκεκριμένη πλατφόρμα όμως δεν επιτρέπει την χρήση δυναμικής μνήμης. Η απαγόρευση της χρίσης δυναμική μνήμης οφείλεται στο γεγονός ότι κάποιες από τις συσκευές δεν επιτρέπουν την χρήση δυναμικής μνήμης. Ως αποτέλεσμα, αν στο πρότυπο RPL ο κάθε κόμβος χρειαζόταν έως n^2 (όπου n το πλήθος των συσκευών του δικτύου) θέσεις μνήμης για να αποθηκεύσει τους πίνακες δρομολόγησης, με την χρίση της wiselib θα χρειάζεται πάντα n^2 θέσεις μνήμης. Παράλληλα καταλαβαίνουμε πως η προσπέλαση μνήμης τάξεως του n^2 έχει θεωρητικά χειρότερη πολυπλοκότητα $O(n^2)$ αν και πρακτικά είναι μεγαλύτερης τάξης αν υποθέσουμε πως γίνεται προσπέλαση και αναζήτηση αλφαριθμητικοί 128 bits (IPv6 διευθύνσεις). Καταλαβαίνουμε λοιπόν πως η υποστήριξη ενός δικτύου με 100+ κόμβους, αν δεν είναι απίθανη λόγω της μνήμης που απαιτείται, είναι αρκετά δαπανηρή ως προς την δρομολόγηση των μηνυμάτων.

2.1.5 Αρχή σχεδιασμού

Το εκάστοτε κατανεμημένο δίκτυο μπορεί να αποτελείται από πολλές διαφορετικές οντότητες του πρωτοκόλλου RPL. Κάθε οντότητα μπορεί να έχει την δικιά της χρήση και ακόμα να είναι ανταγωνιστική προς τις άλλες οντότητες. Στο κεφάλαιο αυτό θα αναλύσουμε πως μια οντότητα λειτουργεί.

Στην προσπάθεια του το πρωτόκολλό RPL να είναι χρήσιμο σε διάφορα ήδη LLN, το RPL διαχωρίζει την διαδικασία επεξεργασίας και προώθησης πακέτων από την βελτιστοποίηση της δρομολόγησης πακέτων. Παραδείγματα τέτοιων βελτιστοποιήσεων μπορεί να είναι η μείωση ενέργειας, η μείωση καθυστέρησης κ.λ.π.. Κάθε οντότητα του RPL εχει μια συνάρτηση σκοπού όπου αυτή καθορίζει τον τρόπο με τον οποίο το κάθε δίκτυο διαμορφώνεται.

Η λειτουργία του RPL απαιτεί οι δίαυλοι επικοινωνίας να είναι αμφίδρομοι. Παρόλα αυτά κάποια LLN μπορεί να παρουσιάζουν ασύμμετρη επικοινωνία. Για αυτόν τον λόγο απαιτείται πριν ένας κόμβος επιλεχθεί να χρησιμοποιηθεί από κάποιον άλλον κόμβο, ο δεύτερος να επαληθεύσει αν ο πρώτος εχει αμφίδρομη επικοινωνία με τον δεύτερο.

Επίσης, το RPL απαιτεί έναν εξωτερικό μηχανισμό ικανό να μεταδώσει πακέτα έλεγχου του RPL.

2.1.6 Ορισμοί

DAG: Κατευθυνόμενος άκυκλος γραφος (Directed Acyclic Graph). Ένας Κατευθυνόμενος άκυκλος γραφος έχει την ιδιότητα ότι κάθε ακμή συνδέεται με τέτοιον τρόπο ώστε οι ακμές του δικτύου να μην σχηματίζουν κύκλο. Ένας τέτοιος γραφος λέγεται επίσης και γράφος δέντρο.

DAG root: Η ρίζα του DAG. Εφόσον το DAG εξ'ορισμού είναι άκυκλο, κάθε DAG πρέπει να έχει μια ρίζα.

Destination Oriented DAG (DODAG): Ένα DAG με μια μόνο ρίζα

DODAG root: Η ρίζα του DODAG.

Κατεύθυνση προς τα πάνω: Αναφέρεται στην κατεύθυνση που ταξιδεύει ένα πακέτο/μήνυμα. Η συγκεκριμένη κατεύθυνση ξεκινά από τους κόμβους και καταλήγει στην ριζά.

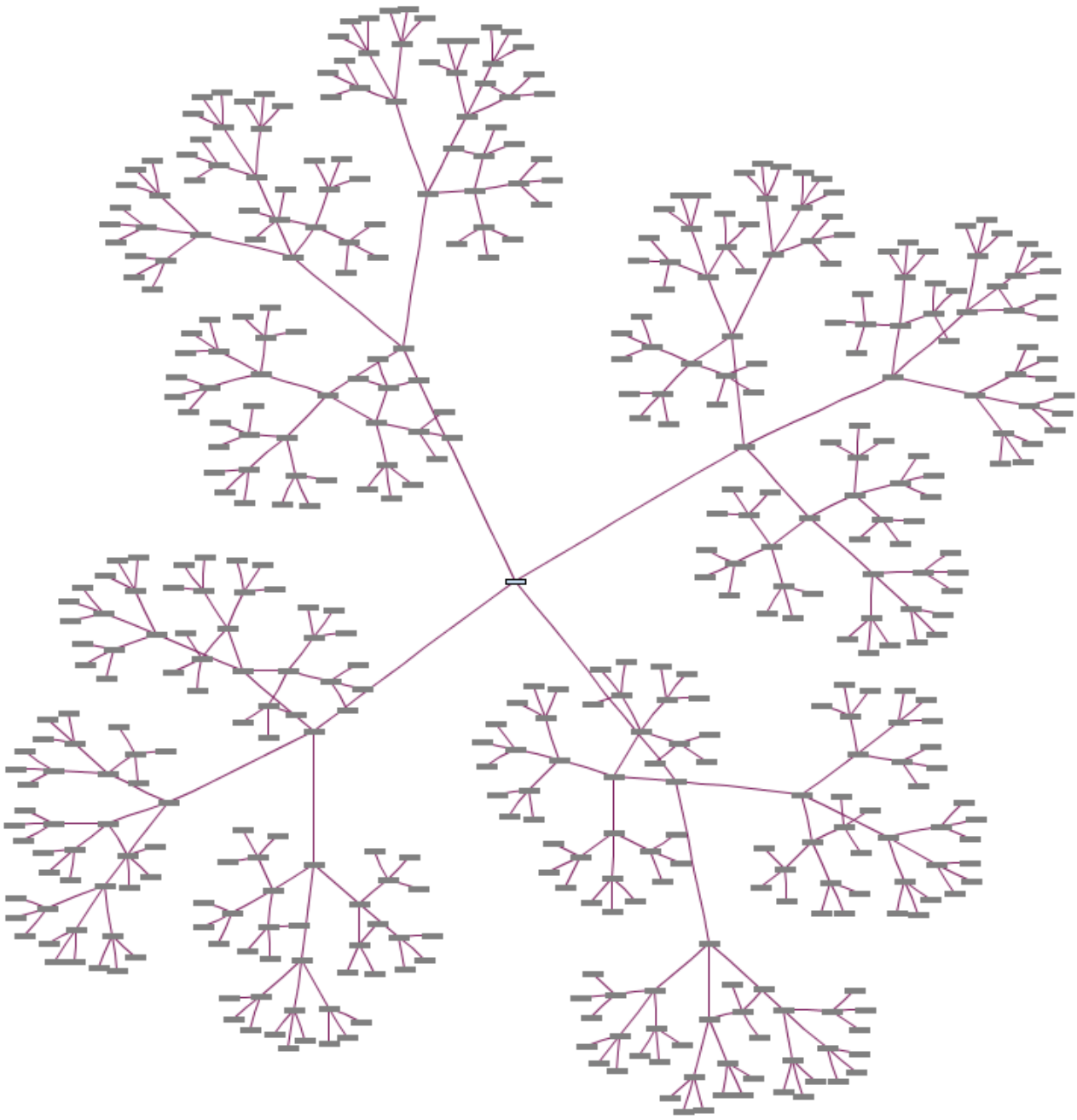
Κατεύθυνση προς τα κάτω: Αναφέρεται στην κατεύθυνση που ταξιδεύει ένα πακέτο/μήνυμα. Η συγκεκριμένη κατεύθυνση ξεκινά από την ριζά και καταλήγει σε έναν η περισσότερους κόμβους.

Τάξη κόμβου: Η τάξη κόμβου ορίζει την θέση ενός κόμβου σε σχέση με το υπόλοιπο δίκτυο. Όσο πιο κοντά στην ριζά βρίσκετε ένας κόμβος, τόσο μικρότερη τάξη έχει και αντίστροφα, όσο πιο μακριά από την ρίζα, τόσο μεγαλύτερη η τάξη του.

Συνάρτηση σκοπού: Ορίζει πως οι μετρικές δρομολόγησης, βελτιστοποίησης δικτύου και οι σχετικές συναρτήσεις υπολογίζουν την καλύτερη μορφή του δικτύου.

DIO: Πακέτο πληροφορίας του DODAG (DODAG information object)

DAO: Πακέτο εύρεσης μονοπατιού δρομολόγησης



Εικόνα 2.1.1: Άκυκλος γράφος / γράφος δέντρο

2.2 Ανάλυση των τμημάτων του πρωτοκόλλου

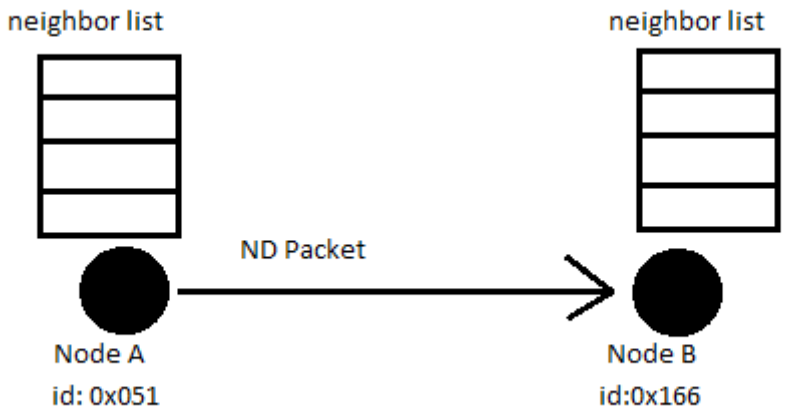
2.2.1 Εντοπισμός γειτόνων (Neighbor discovery)

Κατά την διάρκεια λειτουργίας ενός κόμβου, είτε ανήκει σε ένα δίκτυο RPL, είτε δεν έχει ακόμα εισαχθεί σε ένα τέτοιο δίκτυο, θα διαφημίζει την θέση του. Αυτή η λειτουργία εκτελείτε καθ'ολη την διάρκεια ζωής του κόμβου. Η διαφήμιση αυτή επαναλαμβάνετε συχνά. Ο ρυθμός επανάληψης της διαφήμισης αφήνετε στον κάθε σχεδιαστή του δικτύου, αλλά για τα δικά μας πειράματα η διαφήμιση του κάθε κόμβου γινόταν κάθε δυο δευτερόλεπτα.

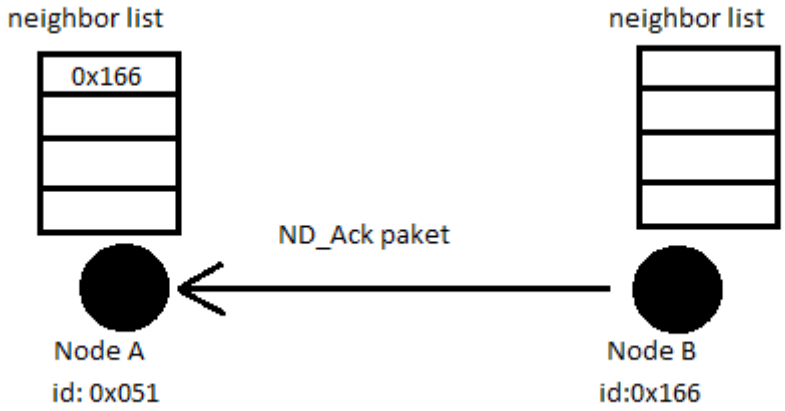
Η διαδικασία εντοπισμού γειτόνων αποτελείται από τρία επιμέρους στοιχειά: την αποστολή ενός μηνύματος εντοπισμού γειτόνων, την παραλαβή και επεξεργασία ενός τέτοιου μηνύματος και την αποστολή μηνύματος επιβεβαίωσης.

Αρχικά ένας κόμβος θα στείλει το μήνυμα εντοπισμού γειτόνων (έστω για το παράδειγμα μας αυτός ο κόμβος είναι ο κόμβος Α). Εν συνεχεία αυτό το μήνυμα παραλαμβάνεται από έναν δεύτερο κόμβο (έστω για το παράδειγμα μας ο κόμβος αυτός είναι ο κόμβος Β). Όταν ο κόμβος Β παραλάβει το μήνυμα θα το επεξεργαστεί για να βρει από που στάλθηκε το μήνυμα. Έπειτα ο κόμβος Β θα απαντήσει με ένα μήνυμα επιβεβαίωσης. Μονό αν ο κόμβος Α λάβει το μήνυμα επιβεβαίωσης, ο κόμβος Β μπορεί να προστεθεί στην λίστα γειτόνων του κόμβου Α αφού πρώτα γίνει έλεγχος για να επιβεβαιώσει ο κόμβος Α ότι ο κόμβος Β δεν υπάρχει στην λίστα γειτόνων. Κατά την διαδικασία αυτή μόνο ο κόμβος Α έκανε εισαγωγή στην λίστα γειτόνων του γιατί επιβεβαίωσε ότι υπάρχει αμφίδρομη επικοινωνία μεταξύ τους. Κατά την διάρκεια αυτής της λειτουργίας, μόνο ο κόμβος Α επαλήθευσε ότι υπάρχει αμφίδρομη επικοινωνία μεταξύ του κόμβου Α και Β. Ο κόμβος Β δεν έχει γνώση αν το μήνυμα επιβεβαίωσης που έστειλε παραλήφθηκε από τον κόμβο Α. Θα είναι δυνατόν να το επιβεβαιώσει όταν ξεκινήσει εκείνος την διαδικασία εντοπισμού γειτόνων.

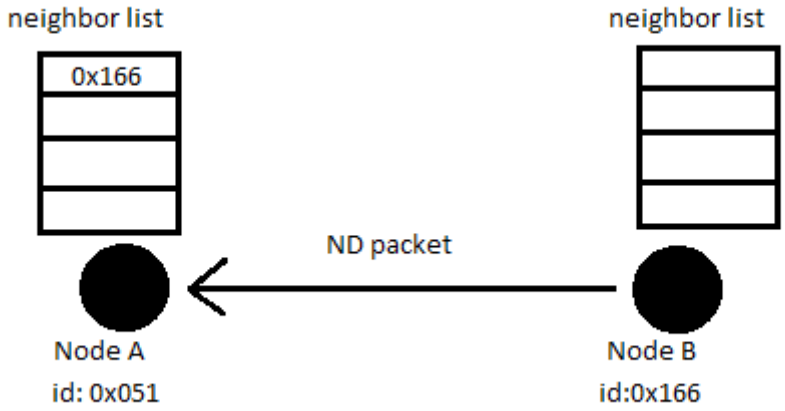
Εικόνα 2.2.1: παράδειγμα εντοπισμού γειτόνων



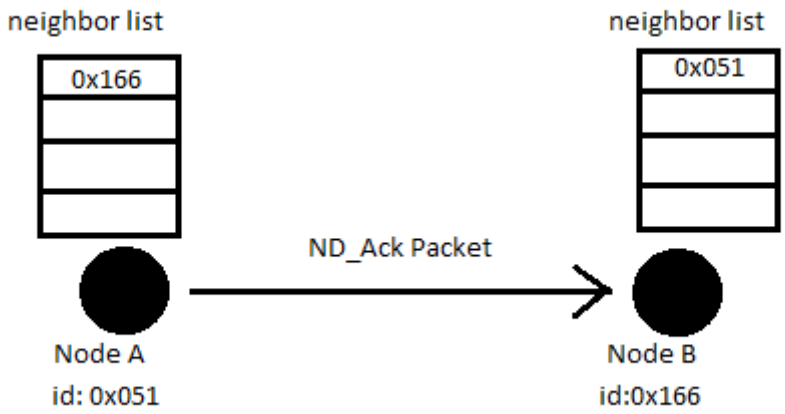
Εικόνα 2.2.2: παράδειγμα εντοπισμού γειτόνων



Εικόνα 2.2.3: παράδειγμα εντοπισμού γειτόνων



Εικόνα 2.2.4: παράδειγμα εντοπισμού γειτόνων



Εικόνα 2.2.5: παράδειγμα εντοπισμού γειτόνων

2.2.2 Συνάρτηση σκοπού (Objective function)

Όταν ένας κόμβος ανήκει σε μια οντότητα RPL, διαφημίζει συνεχώς τα στοιχεία της οντότητας όπου ανήκει (DIO πακέτα – θα αναλυθούν σε παρακάτω παράγραφο). Όταν ένας αυτόνομος κόμβος θελήσει να εισέλθει σε μια ήδη υπάρχον οντότητα RPL, θα απαντήσει αναλόγως με τα στοιχεία που ζητάει η εκάστοτε συνάρτηση σκοπού. Κάθε οντότητα RPL περιεχέι μόνο μια συνάρτηση σκοπού. Ο κάθε κόμβος που θα λάβει αυτά τα στοιχεία, θα αποφασίσει βάσει της συνάρτησης σκοπού της εκάστοτε οντότητας. Αν ο κόμβος που αποστέλλει το αίτημα εισαγωγής στην οντότητα, δεχθεί αυτόν τον κόμβο που διαφήμισε τα στοιχειά της οντότητας του, ως πάτερα του, ο κόμβος που έστειλε το αίτημα εισέρχεται στο δίκτυο. Αν απορριφθεί συνεχίζει την αναζήτηση του. Επειδή το δίκτυο είναι δυναμικό, οι προϋποθέσεις που πρέπει να πληρεί ο κάθε κόμβος, επανεξετάζονται σε τακτά χρονικά διαστήματα από τον εκάστοτε κόμβο. Αν βρεθεί πως ο πατέρας του κόμβου δεν πληρεί πια τις προϋποθέσεις, αποκόπτεται από το δίκτυο και ξεκινά την αναζήτηση από την αρχή, αλλιώς κρατάει την θέση του στο δίκτυο

Δίνονται κάποια παραδείγματα συναρτήσεων σκοπού:

-Least hop count: η οντότητα RPL φτιάχνει ένα δεντρό όσο δυνατόν μικρότερου ύψους.

-Strongest communication link: η οντότητα RPL φτιάχνει ένα δεντρό βασιζόμενο στο πόσο δυνατό είναι το σήμα μεταξύ των κόμβων.

-Security/trust: η οντότητα RPL σχηματίζει το δίκτυο βάσει του ποιοι κόμβοι είναι αξιόπιστοι να χρησιμοποιηθούν. Καθώς το δίκτυο δεν μπορεί να αποφανθεί εκ των προτέρων για την εγκυρότητα ενός κόμβου, το δίκτυο διαμορφώνεται συνεχώς μέχρι να βρεθούν οι βέλτιστοι κόμβοι.

-Hybrid (Least hop count/Strongest communication link): Συνδυάζει τις δυο συναρτήσεις σκοπού ώστε να φτιάξει ένα αποδοτικό δίκτυο μικρού ύψους αλλά με έμπιστους διαύλους επικοινωνίας.

Για τα πειράματα μας χρησιμοποιήθηκε η τελευταία συνάρτηση σκοπού, η οποία

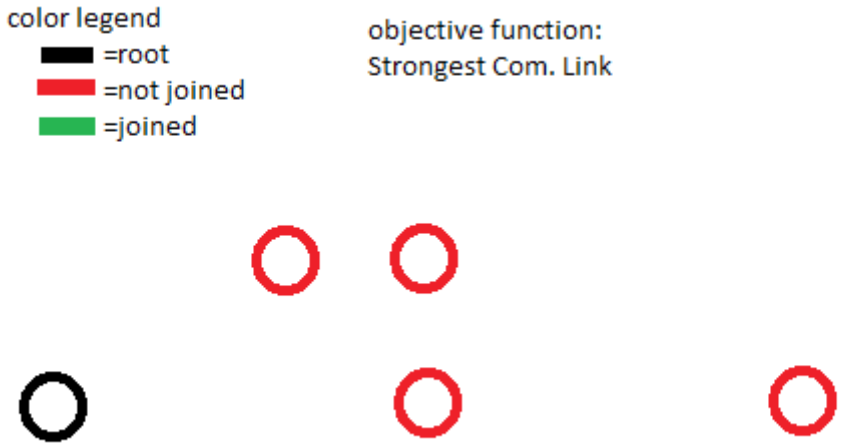
φτιάχθηκε να είναι δυναμική.

2.2.3 διαφήμιση στοιχείων οντότητας RPL (DIO packet)

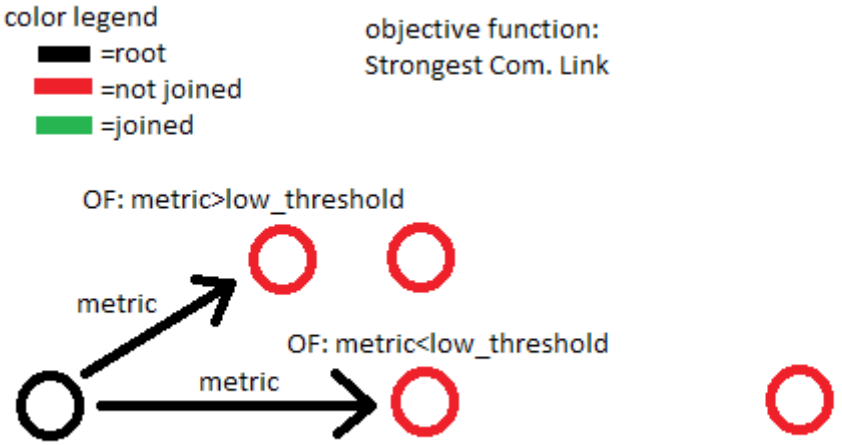
Από την στιγμή που ένας κόμβος εισέλθει σε μια οντότητα RPL, αρχίζει να διαφημίζει τα στοιχειά του δικτύου τα οποία ζητάει η συνάρτηση σκοπού (π.χ.: αν η συνάρτηση σκοπού είναι η Least hop count ο κάθε κόμβος περιλαμβάνει την απόσταση από την ρίζα στα στοιχεία που διαφημίζει. Αν είναι η Strongest Communication Link, ο κόμβος διαφημίζει τη ισχύ σήματος του). Στην ειδική περίπτωση που επιλεχθεί από τον σχεδιαστή του δικτύου ένας κόμβος ως ρίζα, ο κόμβος αρχίζει την εκπομπή των στοιχείων άμεσα κατά την εκκίνηση του. Ο κάθε κόμβος αποστέλλει αυτά τα μηνύματα μόνο στους κόμβους που έχει ανακαλύψει η λειτουργία Εντοπισμού Γειτόνων.

Κατά την λήψη ενός πακέτου DIO, ο κόμβος αρχικά θα ελέγξει αν το μήνυμα στάλθηκε από έναν έγκυρο γείτονα και πως δεν παραλήφθηκε από κάποιον κόμβο ο οποίος δεν έχει αμφίδρομη επικοινωνία με τον κόμβο παραλαβής. Αφού το επιβεβαιώσει αυτό, θα τροφοδοτήσει τις μετρικές που έλαβε στην συνάρτηση σκοπού. Αν η συνάρτηση σκοπού αποφανθεί πως ο κόμβος αποστολής δεν είναι ιδανικός ώστε να γίνει πατέρας στον κόμβο παραλαβής, ο κόμβος παραλαβής συνεχίζει να είναι αυτόνομος έως ότου ένας ιδανικός κόμβος-πατέρας βρεθεί. Αν η συνάρτηση σκοπού δεχθεί τελικά τον κόμβο αποστολής ως πάτερα, ο κόμβος παραλαβής εισέρχεται στο δίκτυο και ξεκινά την αποστολή των δικών του στοιχείων ώστε να μπορέσουν άλλοι κόμβοι να τον επιλέξουν ως πατέρα.

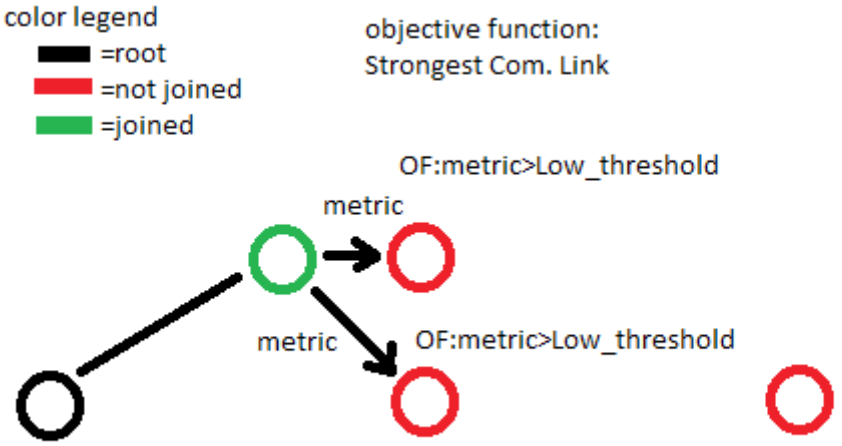
Επειδή όμως το δίκτυο είναι δυναμικό, ο κάθε κόμβος συνεχίζει να ψάχνει συνεχεία έναν πιο ιδανικό πατέρα. Αν βρεθεί ένας ιδανικότερος πατέρας, ο κόμβος θα εισέλθει στον νέο πατέρα. Η διαδικασία επαναλαμβάνεται επ'άπειρον, αν και η διαμόρφωση του δικτύου θα σταματήσει αρκετά σύντομα.



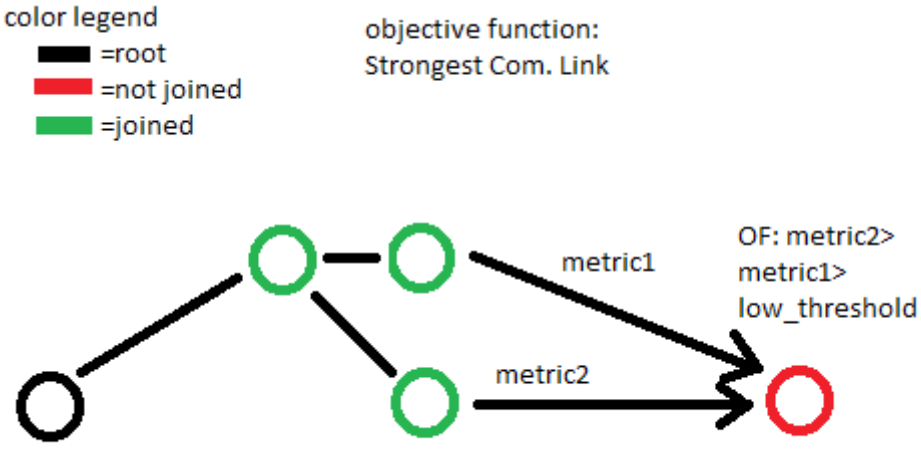
Εικόνα 2.2.6: παράδειγμα διαφήμισης στοιχείων οντότητας και σχηματισμός δικτύου



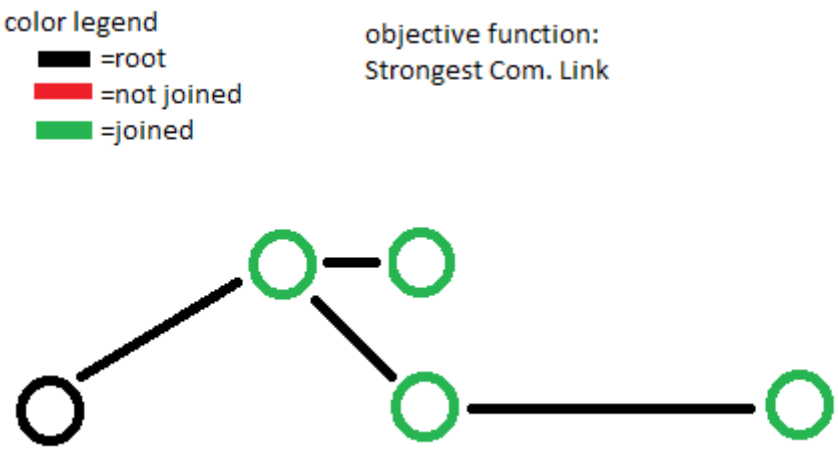
Εικόνα 2.2.7: παράδειγμα διαφήμισης στοιχείων οντότητας και σχηματισμός δικτύου



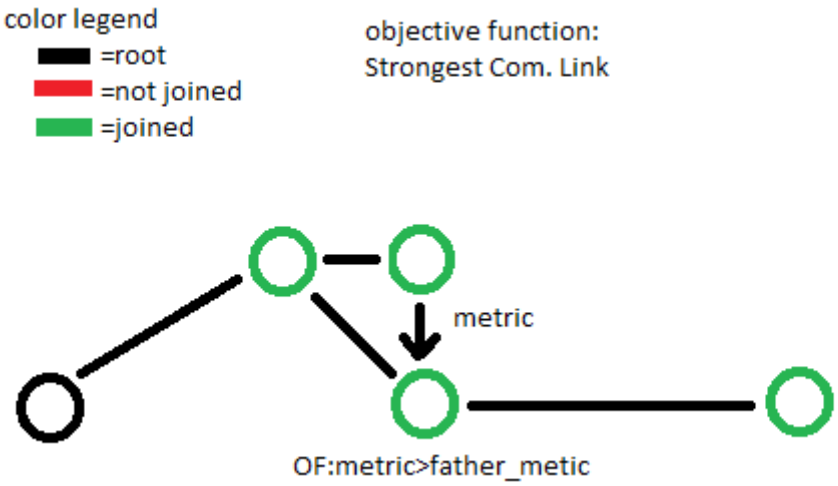
Εικόνα 2.2.8: παράδειγμα διαφήμισης στοιχείων οντότητας και σχηματισμός δικτύου



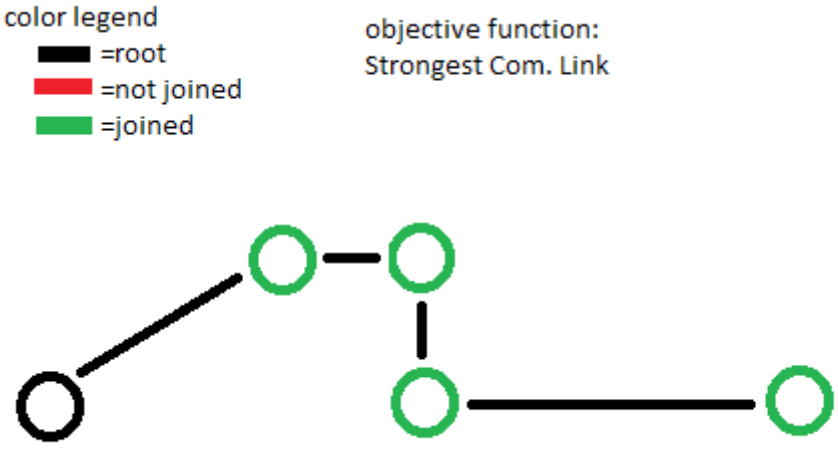
Εικόνα 2.2.9: παράδειγμα διαφήμισης στοιχείων οντότητας και σχηματισμός δικτύου



Εικόνα 2.2.10: παράδειγμα διαφήμισης στοιχείων οντότητας και σχηματισμός δικτύου



Εικόνα 2.2.11: παράδειγμα διαφήμισης στοιχείων οντότητας και σχηματισμός δικτύου



Εικόνα 2.2.12: παράδειγμα διαφήμισης στοιχείων οντότητας και σχηματισμός δικτύου

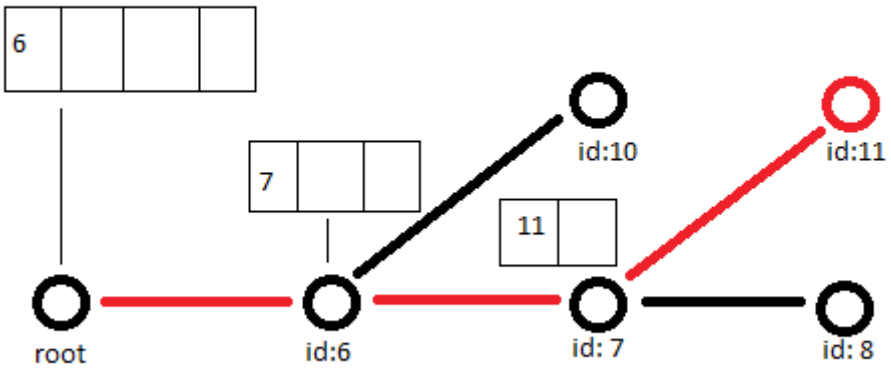
2.2.4 Δημιουργία Πινάκων δρομολόγησης (DAO packet)

Την στιγμή που ένας κόμβος ενσωματωθεί σε μια οντότητα RPL, στέλνει ένα πακέτο DAO προς την ρίζα του δέντρου μέσω των γονέων του κάθε κόμβου. Κάθε κόμβος που θα

λάβει το πακέτο αυτό, θα το προωθήσει προς την ρίζα. Για την ανάγκη αυτή δημιουργήθηκε ένα σύστημα ουράς ώστε να μην υπάρχει περίπτωση να χαθεί το πακέτο ενός κόμβου. Από κάθε κόμβο που περνάει το μήνυμα, αφήνει την πληροφορία για το που βρίσκετε ο κόμβος αποστολής. Καταλαβαίνουμε λοιπόν δυο πράματα: Αρχικά πως αν δυο κόμβοι βρίσκονται σε διαφορετικά υποδέντρα, κανένας από τους δυο αυτούς κόμβους δεν θα έχει γνώση για να δρομολογήσει πακέτο προς τον άλλον κόμβο. Δεύτερον, πως η ριζά του δέντρου γνωρίζει για κάθε κόμβο πως να δρομολογήσει τα πακέτα προς αυτούς.

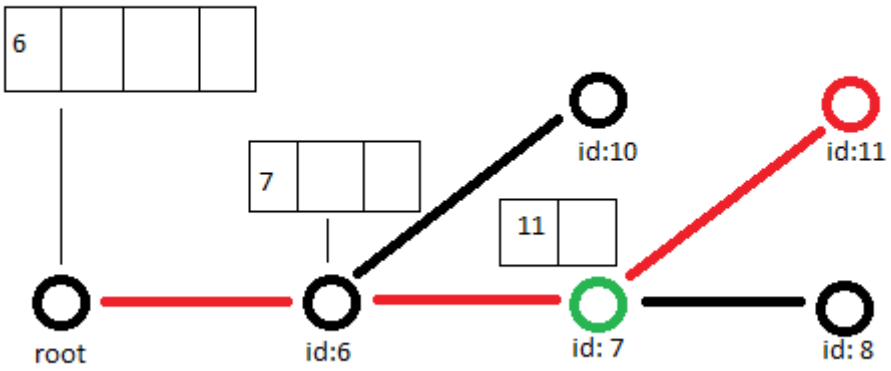
Η γνώση που αποκτιέται από τους ενδιάμεσους κόμβους εμπλουτίζει τους πινάκες δρομολόγησης. Πιο αναλυτικό παράδειγμα θα ακολουθήσει σε σχήμα παρακάτω.

Επειδή η πληροφορία που διαμοιράζεται μεσώ των πακέτων DAO είναι άκρως σημαντική για την επικοινωνία μεταξύ κόμβων του δικτύου, έχει υλοποιηθεί η λειτουργία πακέτου επιβεβαίωσης. Με αυτόν τον τρόπο ο κάθε κόμβος θα στέλνει τα πακέτα DAO σε τακτά χρονικά διαστήματα μέχρι ο πατέρας του κόμβου αυτού να επιβεβαιώσει πως το έλαβε.

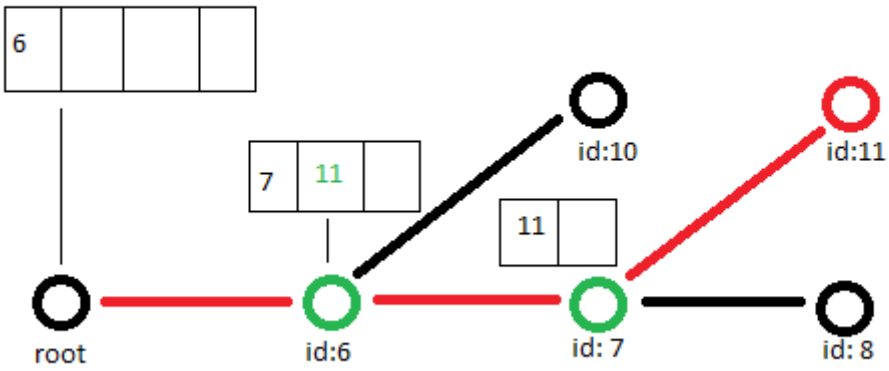


Εικόνα 2.2.13: παράδειγμα δημιουργίας πινάκων δρομολόγησης

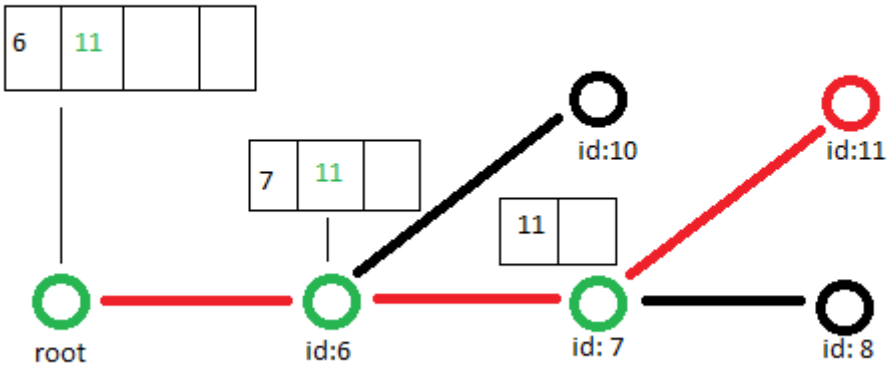
Η κόκκινη γραμμή αναπαριστά την διαδρομή που θα ακολουθήσει το πακέτο DAO, ενώ ο κόκκινος κόμβος είναι ο κόμβος αποστολής. Το πρώτο στοιχείο του κάθε πινάκα είναι ο άμεσος γείτονας του κόμβου.



Εικόνα 2.2.14: παράδειγμα δημιουργίας πινάκων δρομολόγησης

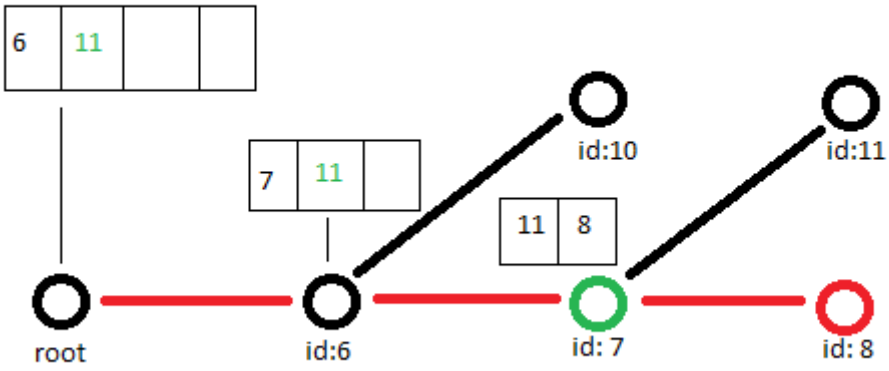


Εικόνα 2.2.15: παράδειγμα δημιουργίας πινάκων δρομολόγησης



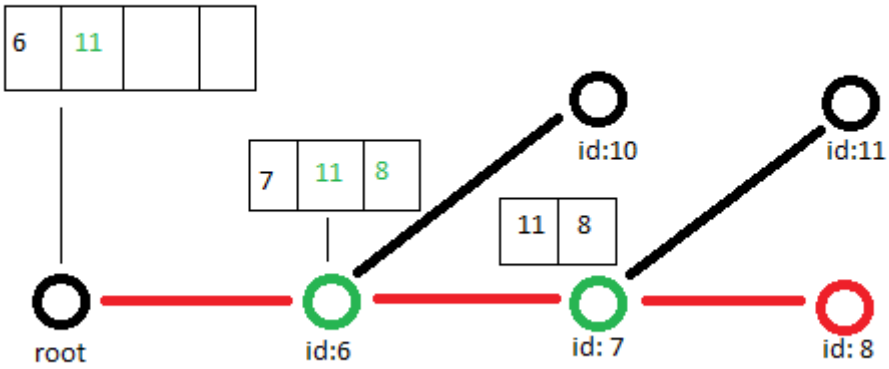
Εικόνα 2.2.16: παράδειγμα δημιουργίας πινάκων δρομολόγησης

Φτάνοντας στην ρίζα, οι πινάκες δρομολόγησης διαμορφώθηκαν ανάλογα. Σε αυτό το στιγμιότυπο, είναι δυνατό να δρομολογηθεί ένα πακέτο προς τον κόμβο 11.

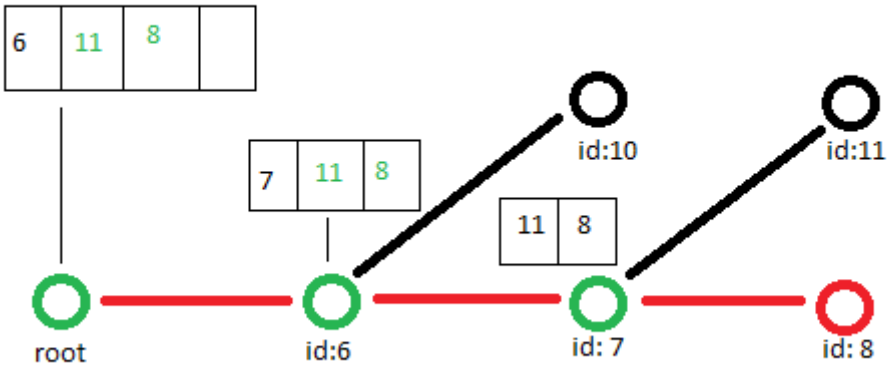


Εικόνα 2.2.17: παράδειγμα δημιουργίας πινάκων δρομολόγησης

Εδώ ξεκινάει η διαδικασία από τον κόμβο 8.



Εικόνα 2.2.18: παράδειγμα δημιουργίας πινάκων δρομολόγησης



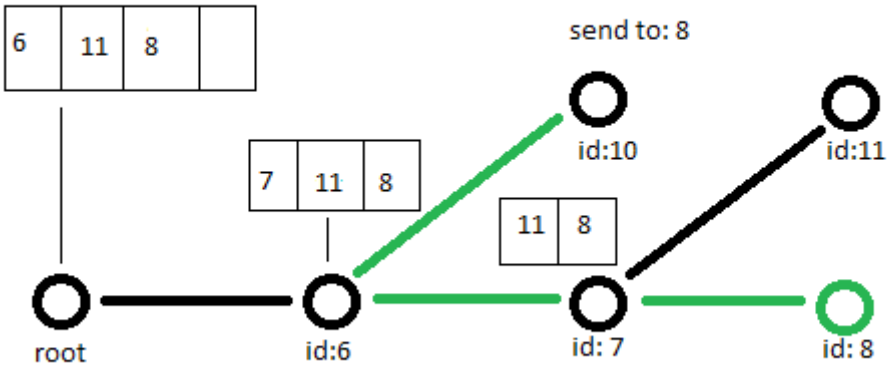
Εικόνα 2.2.19: παράδειγμα δημιουργίας πινάκων δρομολόγησης

Στο τέλος της διαδικασίας, έχει δοθεί γνώση στο δίκτυο, πως να προωθηθεί ένα μήνυμα προς τον κόμβο 8.

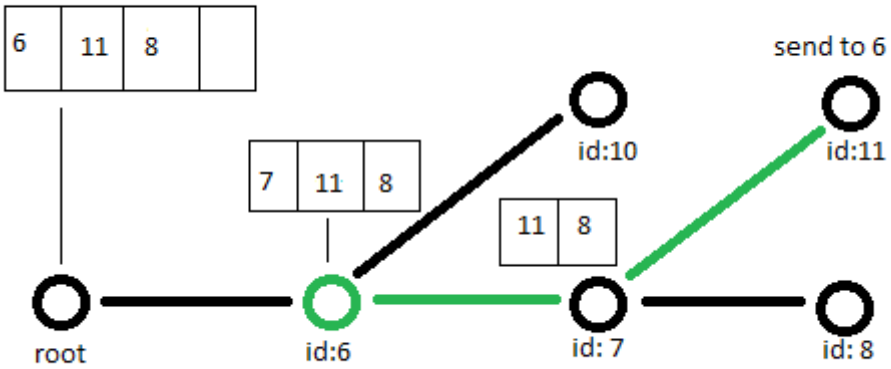
2.2.5 Δρομολόγηση μηνυμάτων

Η δρομολόγηση πακέτων, μετά την διαδικασία των πινάκων δρομολόγησης, είναι σχετικά απλή. Απλή αλλά δαπανηρή διαδικασία. Όταν ένας κόμβος θέλει να στείλει ένα μήνυμα, θα ψάξει σε όλον τον πίνακα δρομολόγησης ώστε να βρει το ID του κόμβου παραλαβής που θέλει να το στείλει. Αν δεν βρεθεί το ID στον κόμβο, σημαίνει πως ο κόμβος δεν ανήκει σε υποδέντρο του κόμβου αποστολής και έτσι στέλνετε στον πατέρα όπου η διαδικασία επαναλαμβάνεται έως ότου βρεθεί ένας κόμβος που περιέχει το ID του κόμβου παραλαβής στους πίνακες δρομολόγησης. Αν το ID του κόμβου παραλαβής βρεθεί στους πινάκες δρομολόγησης, τότε ο κόμβος βρίσκει το πρώτο στοιχείο της σειράς όπου βρέθηκε το ID του κόμβου παραλαβής. Το πρώτο αυτό στοιχείο είναι το ID του γείτονα που θα πρέπει ο κόμβος αποστολής να προωθήσει το μήνυμα ώστε να μπορέσει εν τέλει να φτάσει στον επιθυμητό παραλήπτη. Στην ειδική περίπτωση που το ID του κόμβου

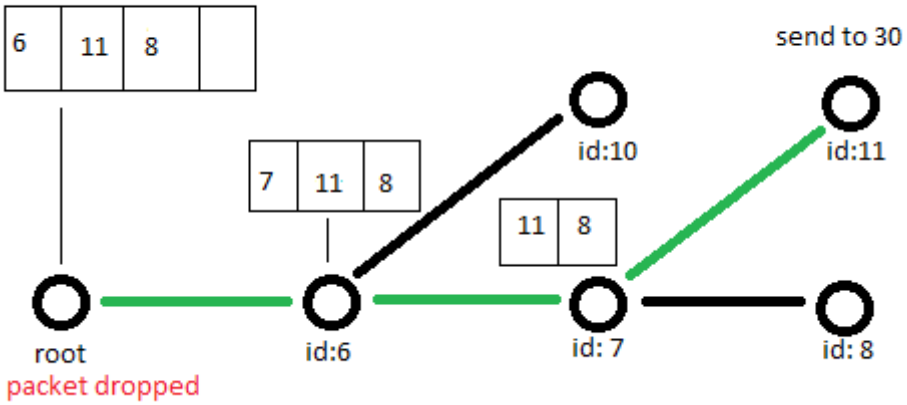
παραλαβής δεν υπάρχει στο δίκτυο, το μήνυμα θα φτάσει μέχρι την ρίζα. Η ρίζα περιέχει όλα τα ID των κόμβων που ανήκουν στο δίκτυο. Αν όπως είπαμε, το ID του κόμβου δεν υπάρχει στο δίκτυο, η ρίζα δεν θα στείλει πουθενά το μήνυμα απορρίπτοντας το τελικά.



Εικόνα 2.2.20: παράδειγμα δρομολόγησης πακέτων με χρήση πινάκων δρομολόγησης



Εικόνα 2.2.21: παράδειγμα δημιουργίας πινάκων δρομολόγησης



Εικόνα 2.2.22: παράδειγμα δημιουργίας πινάκων δρομολόγησης

2.2.6 Σχόλια και παρατηρήσεις

Το πρότυπο RPL περιγράφει ένα σύστημα επικοινωνίας με διευθυνσιοδότηση βάσει του πρωτοκόλλου IPv6. Όπως αναφέρθηκε και πιο πάνω, στην χειρότερη περίπτωση (στον κόμβο-ρίζα) χρειαζόμαστε έναν πινάκα δρομολόγησης μεγέθους n^2 . Κατά την υλοποίηση όμως, ένας τέτοιος πινάκας δεν είναι εφικτός να υπάρξει. Οι περιορισμοί στην μνήμη είναι πάρα πολλοί. Για αυτό, κατά την εγγραφή του κώδικα προτιμήθηκε να χρησιμοποιηθεί ως διεύθυνση συσκευής το μοναδικό ID της συσκευής το οποίο είναι μεγέθους δυο bytes έχοντας το συνολικό μέγεθος του πίνακα στα $2 \times n^2$ bytes. Αν αντιθέτως χρησιμοποιούσαμε το πρωτόκολλο IPv6, το αντίστοιχο μέγεθος θα ανερχόταν στα $16 \times n^2$ bytes. Επιπλέον, εκτός της αύξησης της μνήμης που θα απαιτούταν, θα αυξανόταν και ο χρόνος που απαιτεί η διαδικασία προσπέλασης και εύρεσης στοιχείων μέσα στον πίνακα.

Το πρόβλημα αυτό έχει ήδη αναφερθεί πιο πάνω, αλλά αρχίζει να γίνεται πιο αντιληπτό από τους αναγνώστες της διπλωματικής. Φτάνοντας στα πειράματα και τα αποτελέσματα αυτών των πειραμάτων θα είναι πλέον σαφές το πρόβλημα και το μέγεθος του.

Κεφάλαιο 3 – Σχεδιασμός και ανάπτυξη ενός νέου πρωτοκόλλου

3.1 Εισαγωγή

Ο Σχεδιασμός και η ανάπτυξη του νέου πρωτοκόλλου βασίστηκε στο πρωτόκολλο RPL. Κρατήσαμε τα βασικά χαρακτηριστικά του RPL για την σύσταση και μορφή του δικτύου (εντοπισμός γειτόνων, διαφήμιση στοιχείων οντότητας, συνάρτηση σκοποί) αλλά αλλάξαμε τον τρόπο διευθυνσιοδότησης των κόμβων του δικτύου και αναπτύξαμε έναν πιο αποδοτικό αλγόριθμο δρομολόγησης. Το νέο πρωτόκολλο που αναπτύχθηκε ονομαστικέ improved RPL (iRPL). Το νέο πρωτόκολλο αναπτύχθηκε με στόχο να βελτιώσει την δρομολόγηση πακέτων μέσα στο δίκτυο. Οι βασικές αρχές σχεδιασμού ήταν να εξαλειφθεί η ανάγκη για χρήση πινάκων δρομολόγησης αλλά να μπορεί να υποστηριχθεί επικοινωνία από σημείο σε σημείο. Βασικός λόγος που θέλησα να εξαλείψω τους πίνακες δρομολόγησης ήταν πως όσο μεγαλώνει το δίκτυο αυξάνονται οι απαιτήσεις των πινάκων σε μνήμη και σε επεξεργαστική ισχύ κατά την αναζήτησης εγγράφων.

3.2 Ανάλυση τμημάτων του πρωτοκόλλου iRPL

Όπως αναφέραμε, η διαδικασία του εντοπισμού γειτόνων και η διαφήμιση στοιχείων οντότητας είναι ίδιες με το πρότυπο πρωτόκολλο. Για αυτόν τον λόγο, δεν θα αναλυθούν περαιτέρω.

3.2.1 Διευθυνσιοδότηση (DAO packet)

Είδαμε πως στο πρότυπο RPL κατά την εισαγωγή ενός κόμβου στο δίκτυο, ο κόμβος θα στείλει ένα πακέτο DAO για να εμπλουτίσει του πίνακες δρομολόγησης από τον κόμβο έως την ρίζα. Ένα πρόβλημα που είναι πολύ πιθανό να δημιουργηθεί, είναι κατά την εισαγωγή πολλών κόμβων ταυτόχρονα, να δημιουργηθεί συνωστισμός πακέτων όσο πλησιάζουμε προς την ρίζα. Με τον τρόπο που θα παρουσιάσουμε, ακόμα και αυτό το πρόβλημα βρίσκει λύση.

Η λειτουργία που ακολουθείται κατά την εισαγωγή ενός κόμβου στο δίκτυο, θυμίζει την 3-way-handshake. Όταν ένας κόμβος αποφασίσει βάση της συνάρτησης σκοπού, ποιος θέλει να είναι ο πατέρας του, στέλνει το αρχικό μήνυμα DAO. Το μήνυμα αυτό δεν ταξιδεύει μέχρι την ρίζα όπως προηγούμενος είδαμε για το πρότυπο RPL άλλα μένει στον κόμβο-πατέρα.

Η διευθυνσιοδότηση θυμίζει περισσότερο το πρωτόκολλο IPv6. Πιο συγκεκριμένα, η μορφή του και οι ιδιότητες της είναι ακριβώς ίδιες με του IPv6. Μια διεύθυνση 2^{128} θέσεων. Με άλλα λόγια, το πρωτόκολλο που σχεδίασα μπορεί να υποστηρίξει έως 2^{128} συσκευές και η μνήμη που χρειάζεται για να μπορεί να δρομολογηθεί ένα πακέτο σε τόσες συσκευές είναι ένας πίνακας αλφαριθμητικών στοιχείων των τριάντα δυο θέσεων. Η ακόμα πιο σωστή υλοποίηση θα ήταν μεγέθους 16 bytes, όπως το IPv6, αλλά σχεδιάστηκε με αλφαριθμητικό ώστε αυτό το μέγεθος να μπορεί να αλλάξει ώστε να μπορεί να υποστηρίξει ακόμα περισσότερες (η ακόμα και λιγότερες) συσκευές από όσες αναφέρθηκαν πιο πάνω.

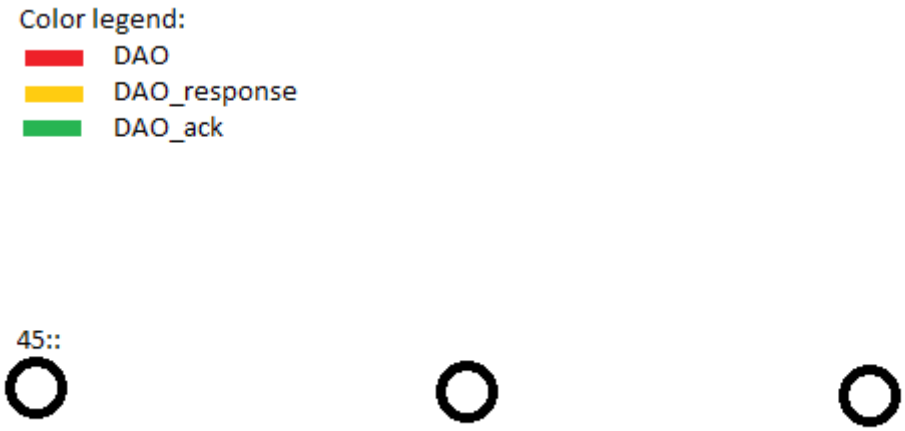
Όταν λοιπόν ο κόμβος-πατέρας λάβει το μήνυμα DAO, θα λειτουργήσει ως ένας DHCP σερβερ. Θα ανατρέξει της εγγραφές του ώστε να δει αν έχει ελεύθερη θέση να δώσει στον κόμβο που ζητάει να εισέλθει στο δίκτυο. Η επιλογή όμως της διεύθυνσης δεν είναι τυχαία. Ο κάθε κόμβος μπορεί να υποστηρίξει μέχρι 256 παιδιά (00 έως FF). Όταν βρει ότι έχει λιγότερα από 256 παιδιά, παίρνει την πρώτη ελεύθερη θέση μεταξύ του 0 και του 256

και την ενθέτει πίσω από την δικιά του διεύθυνση. Για παράδειγμα: Έστω ο κόμβος Α έχει διεύθυνση 24ba:: και η πρώτη ελεύθερη θέση για διεύθυνση παιδιών είναι η 8. η διεύθυνση που θα επιλεγεί για τον κόμβο που θέλει να εισέλθει θα είναι η 24ba:08::. Οποιοσδήποτε κόμβος έχει πάτερα, στην διεύθυνση του θα περιλαμβάνεται η διεύθυνση του πάτερα του. Καταλαβαίνουμε λοιπόν πως η ριζά έχει μοναδική διεύθυνση που περιλαμβάνει μόνο τον εαυτό της, σε κάθε διεύθυνση βρίσκονται οι διευθύνσεις όλων τον κόμβων-πατέρων του μονοπατιού που σχεδιάζεται από τον κόμβο έως την ρίζα και πως από την διεύθυνση μπορούμε να καταλάβουμε την σχετική θέση του κόμβου μέσα στο δίκτυο.

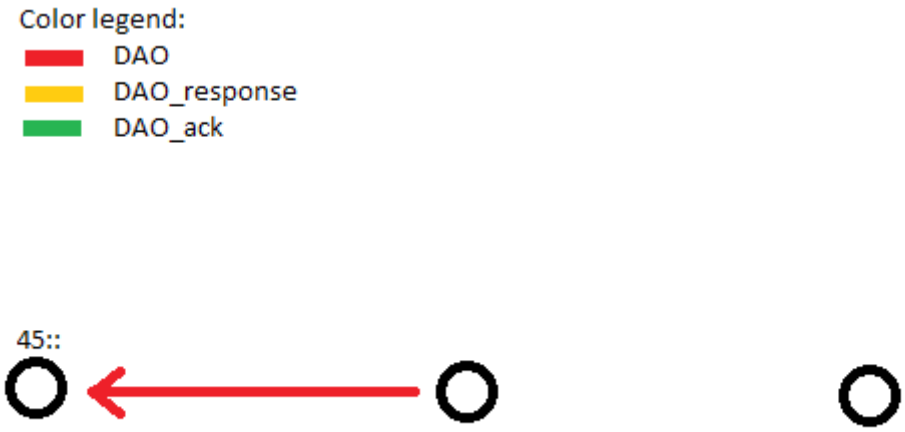
Ακολουθούν δυο περιπτώσεις μετά από αυτό: ο κόμβος πάτερα να μην έχει πλέον κάποια διαθέσιμη διεύθυνση και άρα δεν θα απαντήσει, με αποτέλεσμα ο κόμβος που θέλει να εισέλθει στο δίκτυο να πρέπει να βρει άλλον πατέρα ή να βρεθεί διεύθυνση. Αν βρεθεί διεύθυνση, αυτή μεταδίδεται στον κόμβο που θέλει να εισέλθει στο δίκτυο με ένα μήνυμα DAO_response.

Λαμβάνοντας το μήνυμα, ο κόμβος θέτει την διεύθυνση που του δόθηκε ως δικιά του, ξεκινά να λειτουργεί κ αυτός ως DHCP σέρβερ και στέλνει ένα μήνυμα DAO_ack στον κόμβο-πατερα να τον ενημερώσει πως η διαδικασία ήταν επιτυχής.

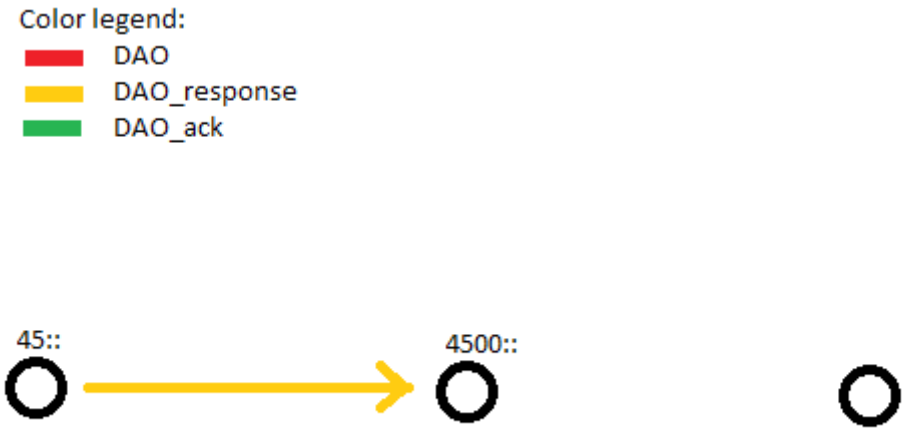
Λόγω της φύσεως του δικτύου (αδύναμοι δίαυλοι επικοινωνίας) κάθε ένα από αυτά τα μηνύματα στέλνετε παραπάνω από μια φορά. Ο κάθε σχεδιαστής μπορεί να θέσει πόσες φορές θα θέλει αυτά τα μηνύματα να στέλνονται. Στην δικιά μου υλοποίηση έχω ορίσει να στέλνονται το πολύ 4 φορές κάθε μήνυμα με μια συγκεκριμένη χρονική καθυστέρηση μεταξύ τους. Προφανώς αν ληφθεί ένα από τα επόμενα μηνύματα, σταματάει η αποστολή ενός προηγούμενου μηνύματος.



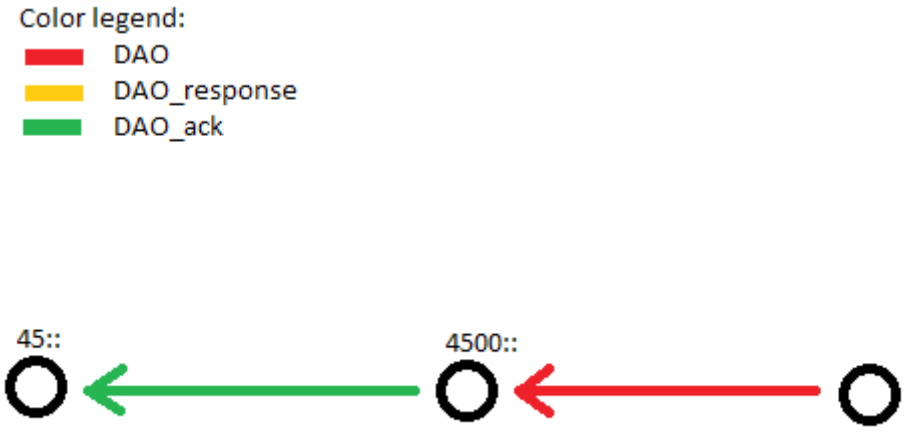
Εικόνα 3.2.1: *Παράδειγμα εισαγωγής 2 κόμβων και ανάθεση διεύθυνσης σε αυτούς.*



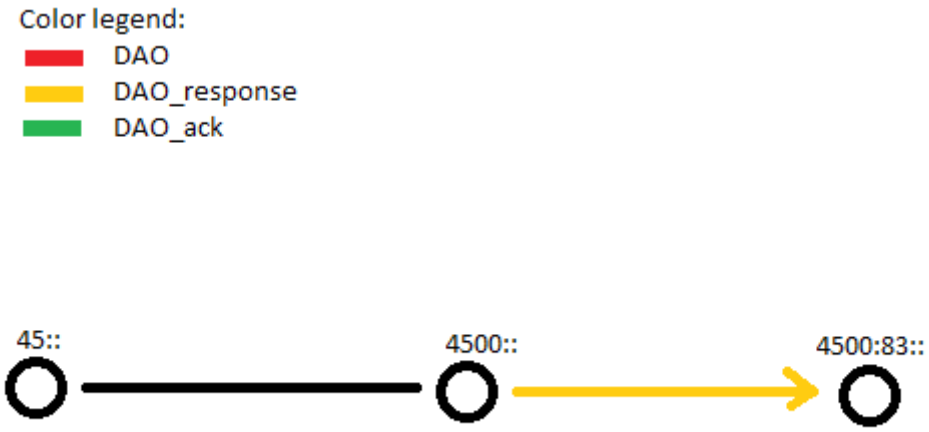
Εικόνα 3.2.2: *Παράδειγμα εισαγωγής 2 κόμβων και ανάθεση διεύθυνσης σε αυτούς.*



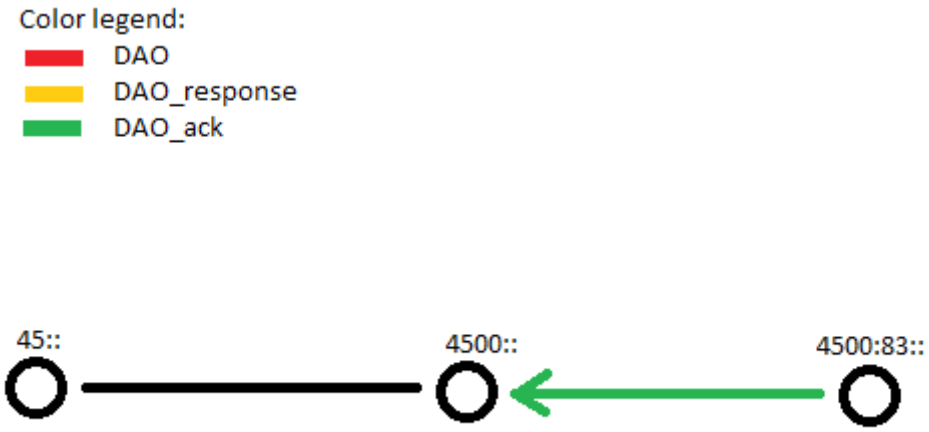
Εικόνα 3.2.3: *Παράδειγμα εισαγωγής 2 κόμβων και ανάθεση διεύθυνσης σε αυτούς.*



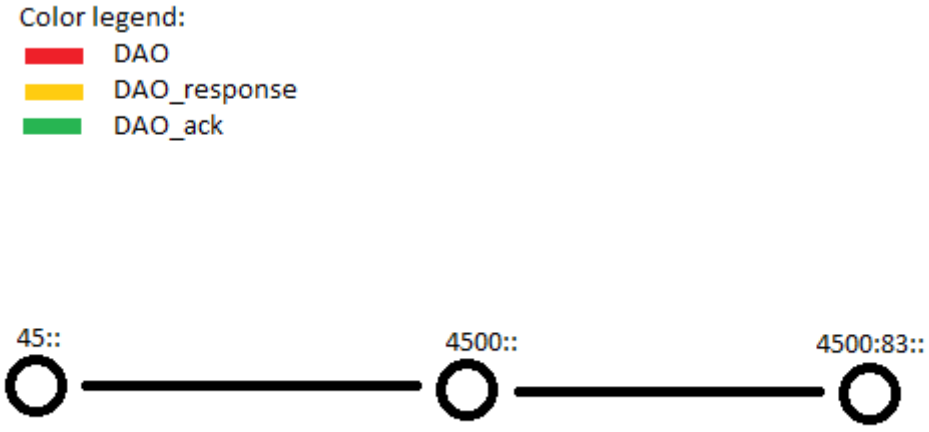
Εικόνα 3.2.4: *Παράδειγμα εισαγωγής 2 κόμβων και ανάθεση διεύθυνσης σε αυτούς.*



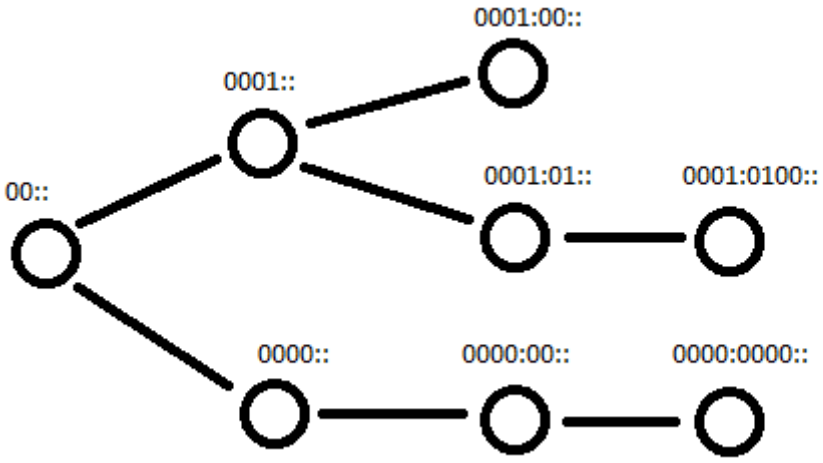
Εικόνα 3.2.5: *Παράδειγμα εισαγωγής 2 κόμβων και ανάθεση διεύθυνσης σε αυτούς.*



Εικόνα 3.2.6: *Παράδειγμα εισαγωγής 2 κόμβων και ανάθεση διεύθυνσης σε αυτούς.*



Εικόνα 3.2.8: *Παράδειγμα εισαγωγής 2 κόμβων και ανάθεση διεύθυνσης σε αυτούς.*



Εικόνα 3.2.9: *Παράδειγμα διευθύνσεων σε ένα μεγαλύτερο δίκτυο.*

3.2.2 Δρομολόγηση πακέτων στο δίκτυο

Σε αυτή την παράγραφο θα γίνει πιο αντιληπτός ο λόγος ύπαρξης του συγκεκριμένου τρόπου διευθυνσιοδότησης, θα αναλύσουμε τον τρόπο που λειτουργεί η δρομολόγηση και θα δούμε πως επιτυγχάνεται η ελάχιστη πολυπλοκότητα.

Όταν ένας κόμβος θέλει να στείλει ένα μήνυμα δεδομένων, το μόνο που χρειάζεται να γνωρίζει είναι η διεύθυνση του κόμβου αποστολής. Λόγω της μορφής των διευθύνσεων, η κάθε διεύθυνση παρέχει και την γνώση για την σχετική θέση του κόμβου στο δίκτυο, τόσο για το βάθος, όσο και για το υποδεντρο όπου ανήκει. Έτσι λοιπόν, όταν ένας κόμβος θέλει να στείλει (ή να προωθήσει) ένα μήνυμα, ελέγχει την διεύθυνση του παραλήπτη. Υπάρχουν τρία πιθανά σενάρια:

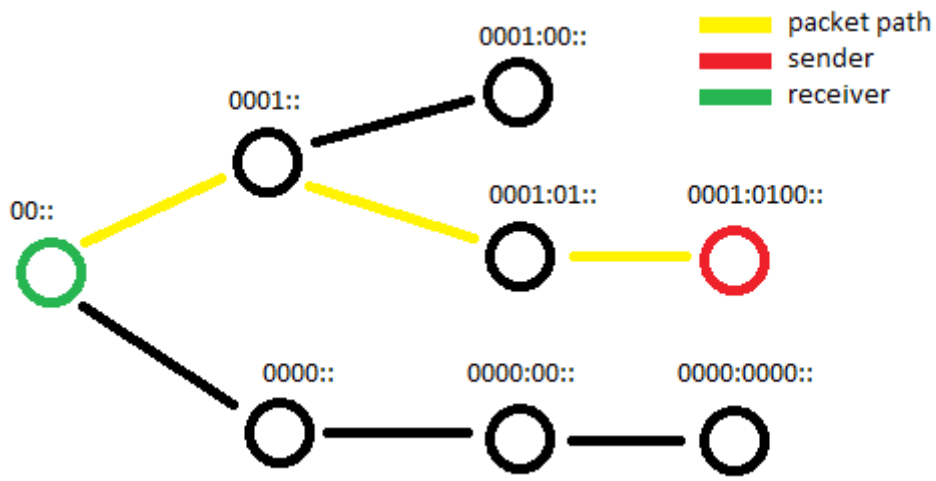
Η διεύθυνση του παραλήπτη δεν περιέχει ολόκληρη την διεύθυνση του κόμβου-αποστολέα, άρα ο παραλήπτης δεν βρίσκεται στο υποδέντρο κάτω από τον αποστολέα άρα θα προωθηθεί προς τα πάνω στο δέντρο.

Η διεύθυνση του παραλήπτη περιέχει ολόκληρη την διεύθυνση του κόμβου-αποστολέα, άρα ο παραλήπτης βρίσκεται στο υποδέντρο κάτω από τον αποστολέα άρα θα προωθηθεί προς τα κάτω στο δεντρό.

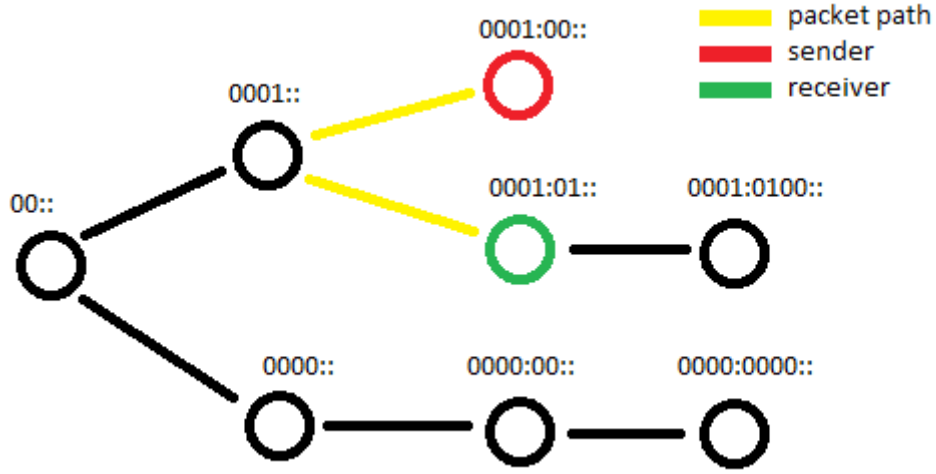
Η διεύθυνση του παραλήπτη είναι ίδια με τον κόμβο που έλαβε το μήνυμα, άρα το μήνυμα προοριζόταν για τον κόμβο αυτό. Από εκεί και πέρα το μήνυμα προωθείται στα άλλα επίπεδα επικοινωνίας και επεξεργάζεται από την ανάλογη εφαρμογή.

Σε επίπεδο κώδικα, η διαδικασία έλεγχου της διεύθυνσης του παραλήπτη είναι μια απλή strncmp() η οποία δεν εξαρτάτε από το μέγεθος η την μορφή του δικτύου. Για κάθε προώθηση απαιτούνται μόνο πέντε συγκρίσεις, άρα μπορούμε να πούμε πως η πολυπλοκότητα του αλγορίθμου είναι τάξης του O(1).

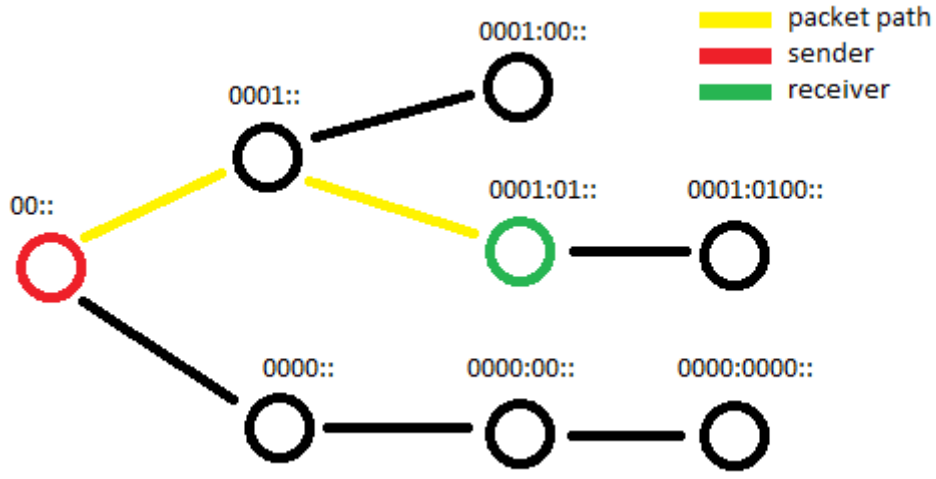
Μπορούμε ήδη να αποφανθούμε πως θεωρητικά το iRPL είναι βελτιωμένο, τόσο στην διαχείριση μνήμης όσο και στην πολυπλοκότητα του αλγορίθμου, αλλά πιο κάτω θα παρουσιαστούν και τα αποτελέσματα των πειραμάτων που θα επαληθεύουν τους ισχυρισμούς αυτούς.



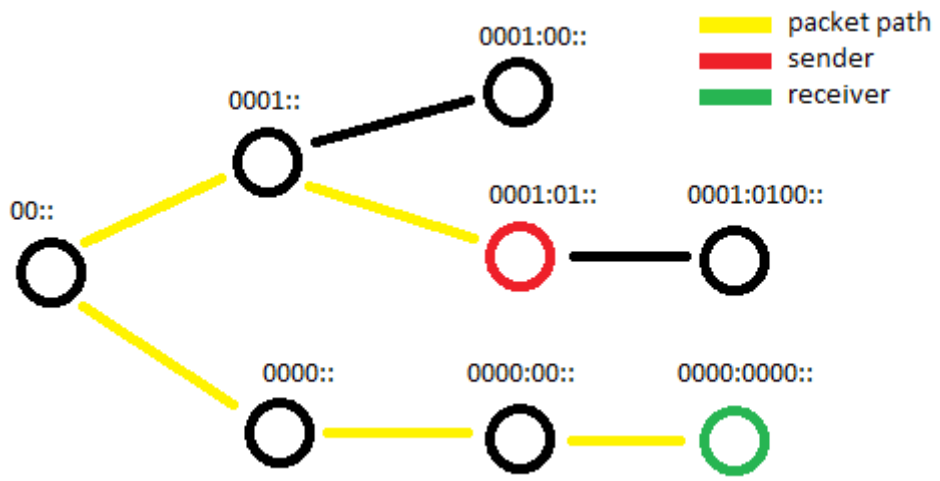
Εικόνα 3.2.10: *Παράδειγμα δρομολόγησης πακέτου μέσα στο δίκτυο.*



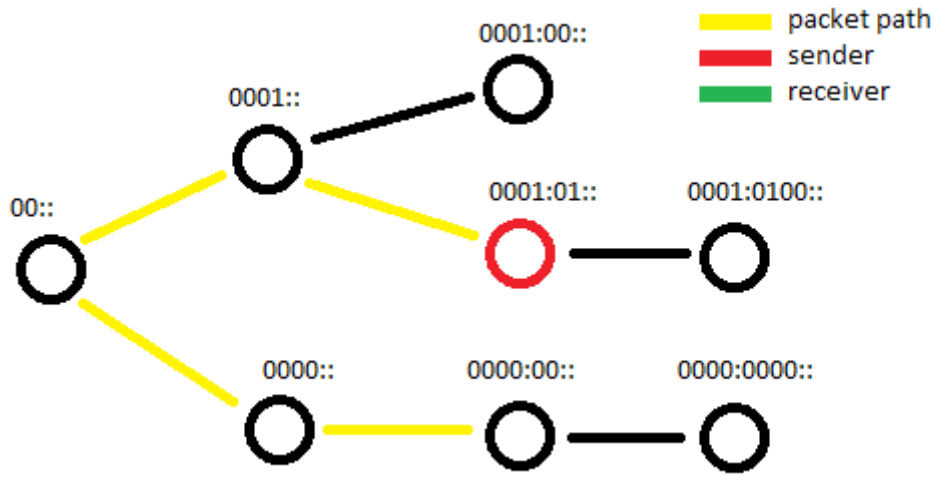
Εικόνα 3.2.11: *Παράδειγμα δρομολόγησης πακέτου μέσα στο δίκτυο.*



Εικόνα 3.2.12: *Παράδειγμα δρομολόγησης πακέτου μέσα στο δίκτυο.*



Εικόνα 3.2.13: *Παράδειγμα δρομολόγησης πακέτου μέσα στο δίκτυο.*



Εικόνα 3.2.14: Στο παραπάνω παράδειγμα, ο παραλήπτης είναι ο κόμβος 0000:0010 (ο οποίος δεν υπάρχει). Παρουσιάζεται το μονοπάτι που θα ακολουθήσει το μήνυμα πριν τελικά απορριφθεί από το δίκτυο καθώς δεν θα υπάρχει κόμβος να το λάβει.

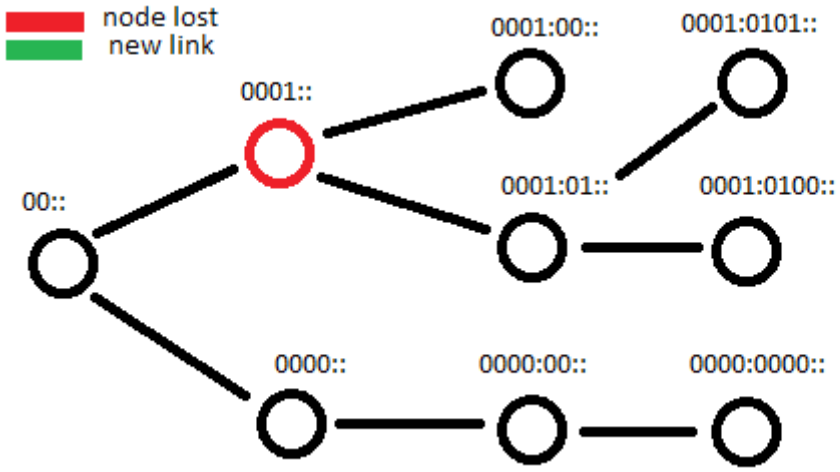
3.3 Επιπλέον λειτουργίες που εκτελεί το iRPL

3.3.1 Έλεγχος λειτουργίας δικτύου

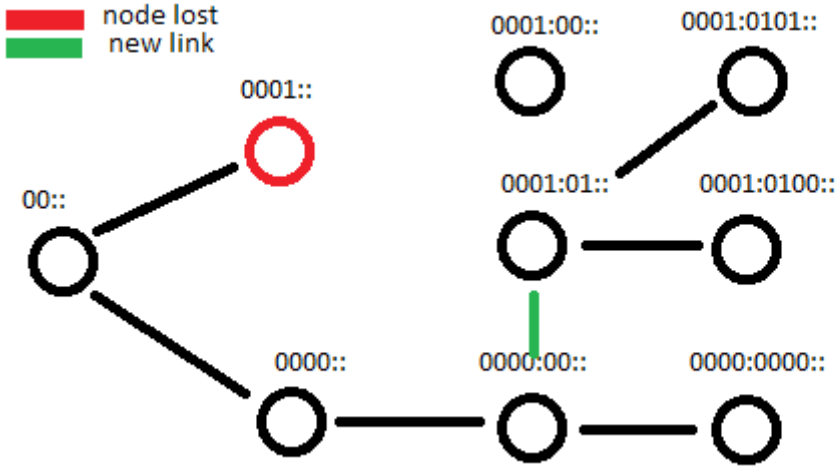
Ο κάθε κόμβος είναι υπεύθυνος να εξασφαλίσει πως ο πατέρας του βρίσκεται ακόμα στο δίκτυο και ότι δεν έχει χαθεί η επικοινωνία του με το υπόλοιπο δίκτυο. Αφού κάθε κόμβος όσο βρίσκεται συνδεδεμένος σε ένα δίκτυο διαφημίζει πακέτα DIO, το μόνο που αρκεί είναι ένας χρονομετρητής. Αν ο πατέρας του εκάστοτε κόμβου έχει ξεπεράσει το δοσμένο όριο για να στείλει πακέτο DIO, ο κόμβος θεωρεί πως έχει χάσει τον πάτερα του και πως η επικοινωνία με το υπόλοιπο δίκτυο έχει χαθεί. Οι κόμβοι όμως που βρίσκονται κάτω από τον κόμβο που έχασε τον πάτερα του δεν έχουν γνώση για την αλλαγή που έγινε στο δίκτυο, για αυτόν τον λόγο ο κόμβος που έχασε τον πάτερα του αναλαμβάνει την επανένταξη όλου του υποδέντρου στο δίκτυο.

3.3.2 Επιδιόρθωση δικτύου

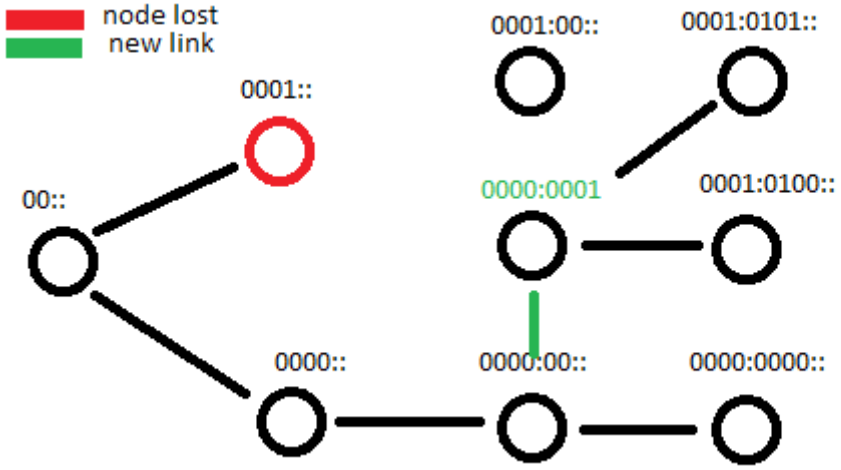
Αφότου ένας κόμβος εντοπίσει πως έχασε τον πατέρα του, μπαίνει ξανά στην κατάσταση αναζήτησης πάτερα, όμως αυτή τη φορά είναι ενήμερο για τους κόμβους που είναι συνδεδεμένοι ήδη πάνω του. Κατά την διαδικασία αυτή η δρομολόγηση πακέτων συνεχίζεται αλλά μόνο μέσα στο υποδέντρο καθώς δεν υπάρχει ενεργός δίαυλος που να συνδέει το μοναχικό υποδέντρο με το υπόλοιπο δίκτυο. Αν ο κόμβος βρει καινούριο πατέρα ξεκινάει η διαδικασία εισαγωγής του κόμβου στο δίκτυο. Όταν ο κόμβος έχει εισέλθει στο δίκτυο, επειδή εισήλθε από διαφορετικό πατέρα από την πρώτη φορά, η διεύθυνση του θα πρέπει να αλλάξει. Όταν λάβει την νέα διεύθυνση, ξεκινάει η διαδικασία επιδιόρθωσης του υποδέντρου. Ο αρχικός μας κόμβος στέλνει μήνυμα στα παιδιά του πως πρέπει να ενημερώσουν την διεύθυνση. Οι κόμβοι παιδιά άπλα θα αλλάξουν το πρόθεμα που είχαν αποκτήσει από τον κόμβο που μόλις επανασυνδέθηκε στο δίκτυο, δηλαδή, δεν θα χρειαστεί να ακολουθήσουν την διαδικασία όπου ζητάνε διεύθυνση, ορίζεται διεύθυνση και επιβεβαιώνουν αυτή την διεύθυνση. Όταν αλλάξουν διεύθυνση στέλνουν με τη σειρά τους το νέο πρόθεμα στα δικά τους παιδιά κ.ο.κ..



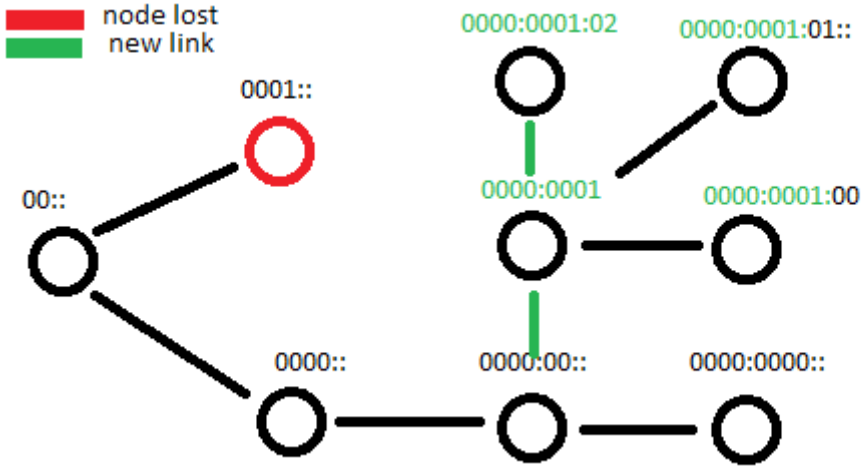
Εικόνα 3.3.1: *Παράδειγμα επιδιόρθωσης δικτύου.*



Εικόνα 3.3.2: *Παράδειγμα επιδιόρθωσης δικτύου.*



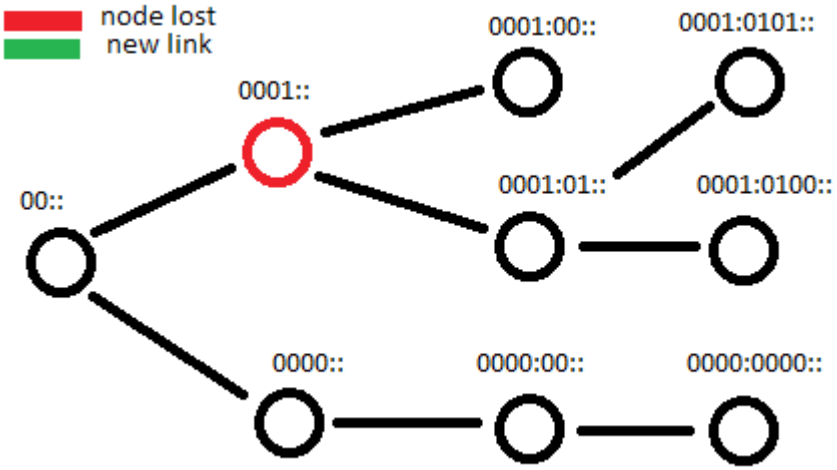
Εικόνα 3.3.3: *Παράδειγμα επιδιόρθωσης δικτύου.*



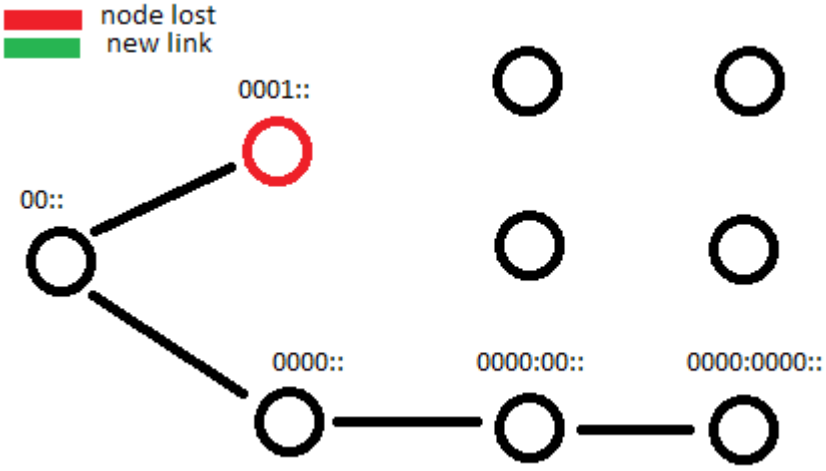
Εικόνα 3.3.4: *Παράδειγμα επιδιόρθωσης δικτύου.*

3.3.3 Διάλυση υποδέντρου

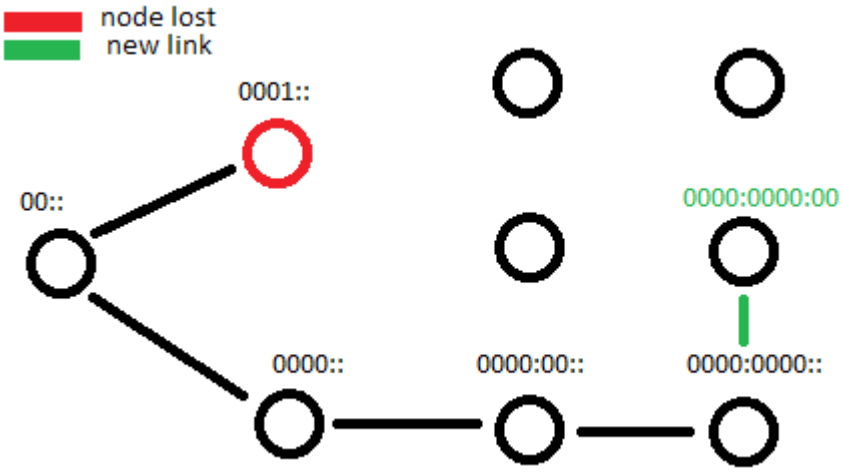
Υπάρχουν δυο ειδικές περιπτώσεις οπού η επιδιόρθωση και επανένταξη του υποδέντρου στο δίκτυο δεν είναι εφικτή. Η πρώτη περίπτωση είναι όταν ο κόμβος όπου έχασε τον πατέρα του (και είναι η ρίζα του υποδικτύου) δεν έχει άλλους κόμβους εντός εμβέλειας εκπομπής και λήψης μηνυμάτων. Η δεύτερη περίπτωση είναι όταν δεν βρίσκει κόμβο με ελεύθερες διευθύνσεις για να δώσει στα παιδιά. Και στις δύο περιπτώσεις δεν θα απαντήσει κανένας κόμβος στο αίτημα για διεύθυνση, ξεπερνώντας το προκαθορισμένο όριο αναμονής. Ο κόμβος – ρίζα του υποδεντρου θα αναγνωρίσει πως βρίσκεται σε ένα τέτοιο σενάριο και θα στείλει εντολή σε όλο το υποδεντρο να διαλυθεί. Διαλύοντας το υποδεντρο, οι κόμβοι εισέρχονται στην αρχική τους κατάσταση όπου αναζητούν πατέρα. Με αυτόν τον τρόπο δίνεται η δυνατότητα σχηματισμού νέου δέντρου όπου θα μπορέσουν τελικά κάποιοι κόμβοι (αν όχι όλοι) να επανασυνδεθούν στο δίκτυο.



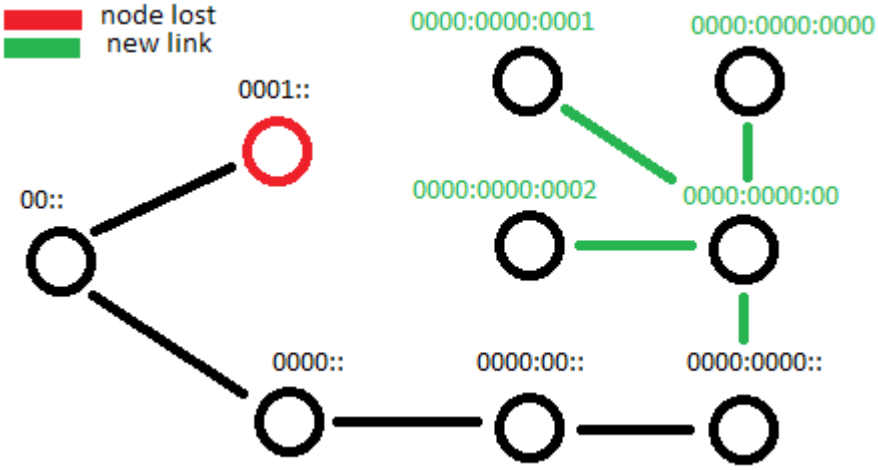
Εικόνα 3.3.5: *Παράδειγμα διάλυσης και επανασύστασης δικτύου.*



Εικόνα 3.3.6: *Παράδειγμα διάλυσης και επανασύστασης δικτύου.*



Εικόνα 3.3.7: *Παράδειγμα διάλυσης και επανασύστασης δικτύου.*



Εικόνα 3.3.8: *Παράδειγμα διάλυσης και επανασύστασης δικτύου.*

3.4 Ομοιότητες και διάφορες με το πρωτόκολλο RPL

3.4.1 Ομοιότητες με το πρότυπο RPL

Η λειτουργία του εντοπισμού γειτόνων, η συνάρτηση σκοπού και η διαφήμιση στοιχείων της οντότητας RPL έχουν παραμείνει ίδιες με το πρότυπο RPL. Ο λόγος που αυτά τα τμήματα αφέθηκαν ίδια είναι επειδή δεν ήταν στα πλαίσια αυτής της εργασίας η βελτιστοποίηση τους. Αν και από τα τρία προαναφερθέν, μόνο η λειτουργία εντοπισμού γειτόνων έχει περιθώρια βελτιστοποίησης και σίγουρα υπάρχει κάποια υλοποίηση η οποία είναι πιο αποδοτική από την δικιά μου, παρόλα αυτά, εγώ ήθελα να βελτιστοποιήσω την χρίση μνήμης και την πολυπλοκότητα της δρομολόγησης, δυο προβλήματα που έχρηζαν άμεση λύση.

3.4.2 Διάφορες

Οι διαφορές του iRPL με το πρότυπο RPL εντοπίζονται στον τρόπο διευθυνσιοδότησης και δρομολόγησης. Με τον αλγόριθμο που ανέπτυξα (που θα περιγραφεί παρακάτω) κατάφερα να μειώσω την χρήση μνήμης σε ένα σταθερό μέγεθος, ανεξάρτητο του πλήθους των συσκευών που βρίσκονται στο δίκτυο, και να φέρω την διαδικασία της αναζήτησης επόμενου γείτονα σε τάξη πολυπλοκότητας του O(1) για οποιοδήποτε μέγεθος του δικτύου.

Κεφάλαιο 4 – Θεωρητική και πειραματική αξιολόγηση

4.1 Εισαγωγή

Αρχικά θα αναλυθούν οι απαιτήσεις σε μνήμη, οι πολυπλοκότητες των αλγορίθμων δρομολόγησης, οι πολυπλοκότητες επικοινωνίας, θα παρουσιαστούν κάποια σενάρια όπου εκτελέστηκαν τα πειράματα όπως και τα αποτελέσματα τους. Τα πειράματα έγιναν και σε εξομοιωτή αλλά και σε πραγματικές συσκευές οι οποίες βρίσκονται στο ινστιτούτο τεχνολογίας υπολογιστών (ΙΤΥ). Τα τεχνικά χαρακτηριστικά των συσκευών, καθώς και η πλατφόρμα του εξομοιωτή που χρησιμοποιήθηκαν θα αναλυθούν παρακάτω.

Στόχος της θεωρητικής και πειραματικής ανάλυσης είναι να αξιολογήσουμε το νέο πρωτόκολλο που σχεδιάστηκε στα πλαίσια της διπλωματικός με το ήδη υπάρχον πρωτόκολλο RPL και να συγκρίνουμε τις αποδόσεις των δυο αυτών πρωτοκόλλων κατά την διάρκεια εκτέλεσης τους.

4.2 Θεωρητική ανάλυση/αξιολόγηση

4.2.1 Ανάλυση απαιτήσεων σε μνήμη

Το πρότυπο RPL χρησιμοποιεί τον κλασσικό τρόπο δρομολόγησης πακέτων, χρησιμοποιώντας πινάκες δρομολόγησης. Οι διαστάσεις των πινάκων εξαρτιούνται άμεσα από την μορφολογία του δικτύου. Κάθε μια γραμμή πινάκα αντιστοιχεί στο πλήθος των γειτόνων κάθε κόμβου, ενώ κάθε μια στήλη στο βάθος του υποδικτύου που βρίσκεται κάτω από τον εκάστοτε κόμβο. Ανάλογα την υλοποίηση το μέγεθος μπορεί να διαφέρει, όμως επειδή στα πλαίσια των πειραμάτων ΔΕΝ υπήρχε η δυνατότητα να χρησιμοποιήσουμε δυναμική μνήμη, ο πίνακας αναγκαστικά θα έπρεπε να ήταν μεγέθους $n \times n$ όπου n οι κόμβοι του δικτύου. Στην βέλτιστη υλοποίηση όπου θα είχαμε στην διάθεση μας δυναμική μνήμη, ο πίνακας θα ήταν $m+n$ όπου m οι γείτονες κάθε συσκευής και n οι κόμβοι του δικτύου. Παρόλα αυτά, αν υπάρχει μια γνώση από πριν για την τοπολογία του δικτύου όπως επίσης και τη μορφή θα έχει το δίκτυο μπορεί εύκολα να αλλάξει ο πίνακας και να έχει μικρότερες διαστάσεις (για παράδειγμα, αν γνωρίζαμε από πριν πως το δέντρο θα έχει μορφή αλυσίδας, ο πίνακας θα είχε μέγεθος $1 \times n$).

Αντιθέτως το iRPL δεν χρειάζεται πίνακες δρομολόγησης, παρά μόνο ένα αλφαριθμητικό. Στην βέλτιστη περίπτωση θα μπορούσε να αναπαρασταθεί με μια μεταβλητή 16 bit, αλλά στην υλοποίηση μου, για λογούς εύκολης προσαρμογής αποφάσισα να το θέσω μεγέθους 32 bytes.

Συγκριτικά: το πρότυπο RPL στην υλοποίηση μου χρειάζεται έναν πίνακα μεγέθους $n^2 \times 16\,bits$ με βέλτιστο πίνακα μεγέθους $(m+(n \times 32)) \times 16\,bits$ (εδώ θεωρούμε πως το μέγεθος ενός δείκτη είναι 32 bits, εξου και ο όρος “32” στην συνάρτηση), ενώ το iRPL έχει σταθερό μέγεθος $256\,bits$. Από άποψη μνήμης το πρότυπο RPL (με την υλοποίηση με πίνακα) είναι πιο αποδοτικό αν το δίκτυο έχει λιγότερες από 16 συσκευές ενώ (με την υλοποίηση με δείκτες) πάντα θα είναι χειρότερο.

4.2.2 Ανάλυση πολυπλοκότητας αλγορίθμων

Για να προωθηθεί ένα πακέτο από έναν κόμβο στον επόμενο, ο κόμβος που το έχει λάβει θα πρέπει να ανατρέξει στις εγγραφές του πίνακα δρομολόγησης. Η πολυπλοκότητα για έναν πίνακα όπως αυτός που υπάρχει στην υλοποίηση του πρότυπου RPL είναι $O(n^2)$. Αν η υλοποίηση είχε γίνει με διπλά συνδεδεμένες εγγραφές, η πολυπλοκότητα θα ήταν $O(m^2)$ με $m<n$

Το iRPL για κάθε δρομολόγηση εκτελεί πάντα τρεις πράξη σύγκρισης δυο αλφαριθμητικών. Η πολυπλοκότητα της δρομολόγησης στο iRPL είναι πάντα $O(1)$.

Μπορούμε εύκολα να καταλάβουμε για ποιον λόγο είναι καλύτερο το iRPL..

4.2.3 Ανάλυση πολυπλοκότητας επικοινωνίας

Η διαφοροποίηση στην πολυπλοκότητα επικοινωνίας δεν βρίσκεται τόσο στην μετάδοση μηνυμάτων, αντιθέτως, βρίσκεται στο πως τα δυο αυτά πρωτόκολλα διαχειρίζονται τα πακέτα DAO.

Στο πρότυπο RPL, όταν ένας κόμβος λάβει ένα μήνυμα DAO, στέλνει επιβεβαίωση στον κόμβο όπου το έστειλε και έπειτα το στέλνει στον πατέρα του. Το βέλτιστο σενάριο είναι όταν όλοι οι κόμβοι έχουν ίδιο πατέρα την ρίζα. Θα σταλούν συνολικά $2 \times n$ πακέτα DAO. Η χειρότερη όμως περίπτωση είναι όταν το δέντρο έχει μορφή αλυσίδας. Κάθε φορά που ένας κόμβος συνδέεται στην αλυσίδα, θα στέλνονται δυο πακέτα από κάθε κόμβο που άνηκε ήδη στην αλυσίδα. Για παράδειγμα, για τον πρώτο κόμβο που θα συνδεθεί στο δίκτυο θα σταλούν 2 μηνύματα, για τον δεύτερο κόμβο που θα συνδεθεί στο δίκτυο θα σταλούν 4 μηνύματα κ.ο.θ.. Αυτό φέρνει την χειρότερη επικοινωνία στα $2 \times n!$ μηνύματα! Άρα εν τέλει, τα μηνύματα DAO για το πρότυπο RPL είναι $2 \times n \leq packet\ count \leq 2 \times n!$.

Το iRPL όμως δεν αναμεταδίδει τα πακέτα DAO μέχρι την ριζά άλλα για την λειτουργιά των πακέτων DAO χρειάζονται 3 πακέτα, φέρνοντας την πολυπλοκότητα επικοινωνίας στα $3 \times n$ πακέτα.

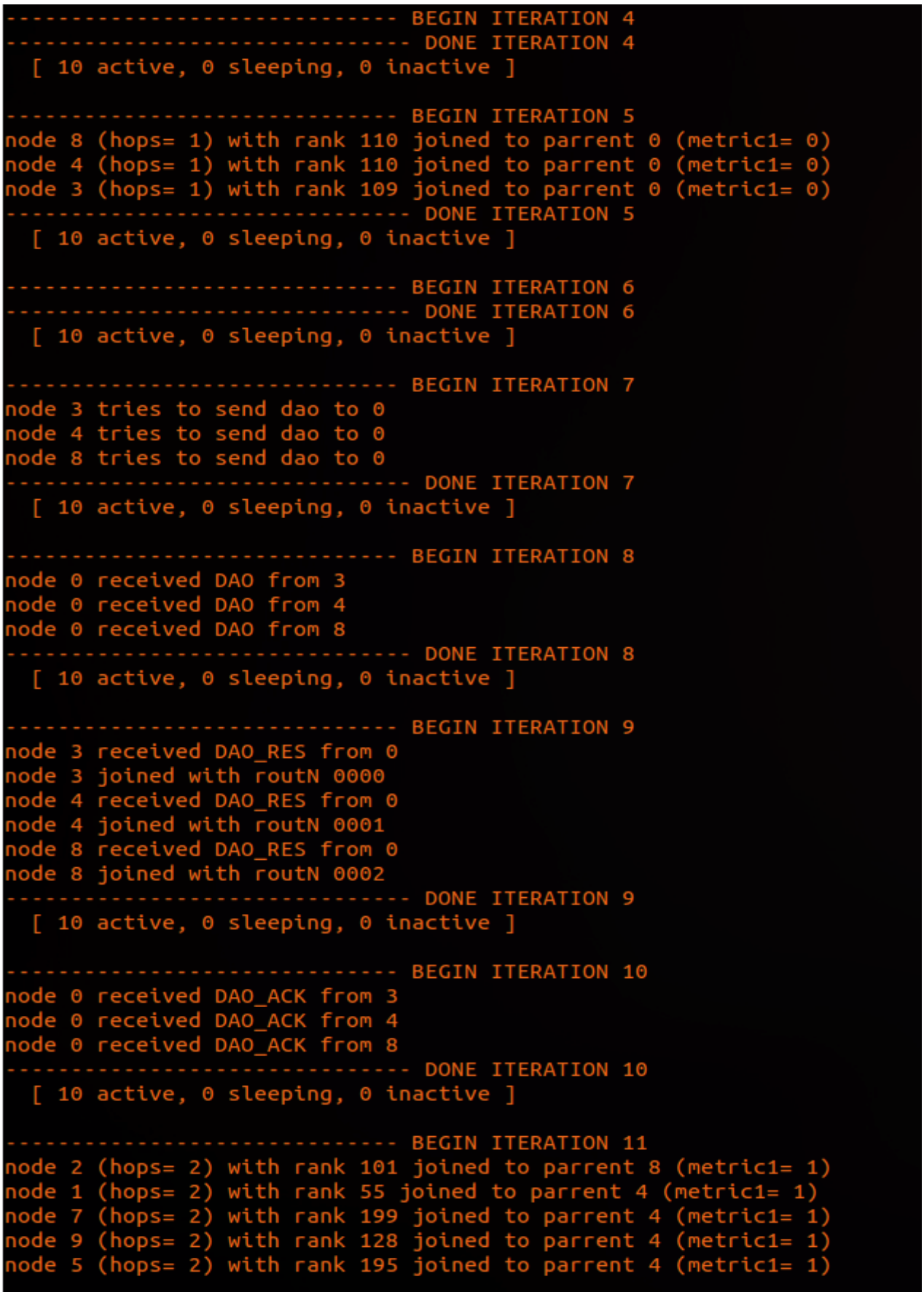
Μπορούμε να υποθέσουμε πως το iRPL θα έχει καλύτερη πολυπλοκότητα επικοινωνίας, ειδικά όσο μεγαλώνει το δίκτυό.

4.3 Πειραματική αξιολόγηση

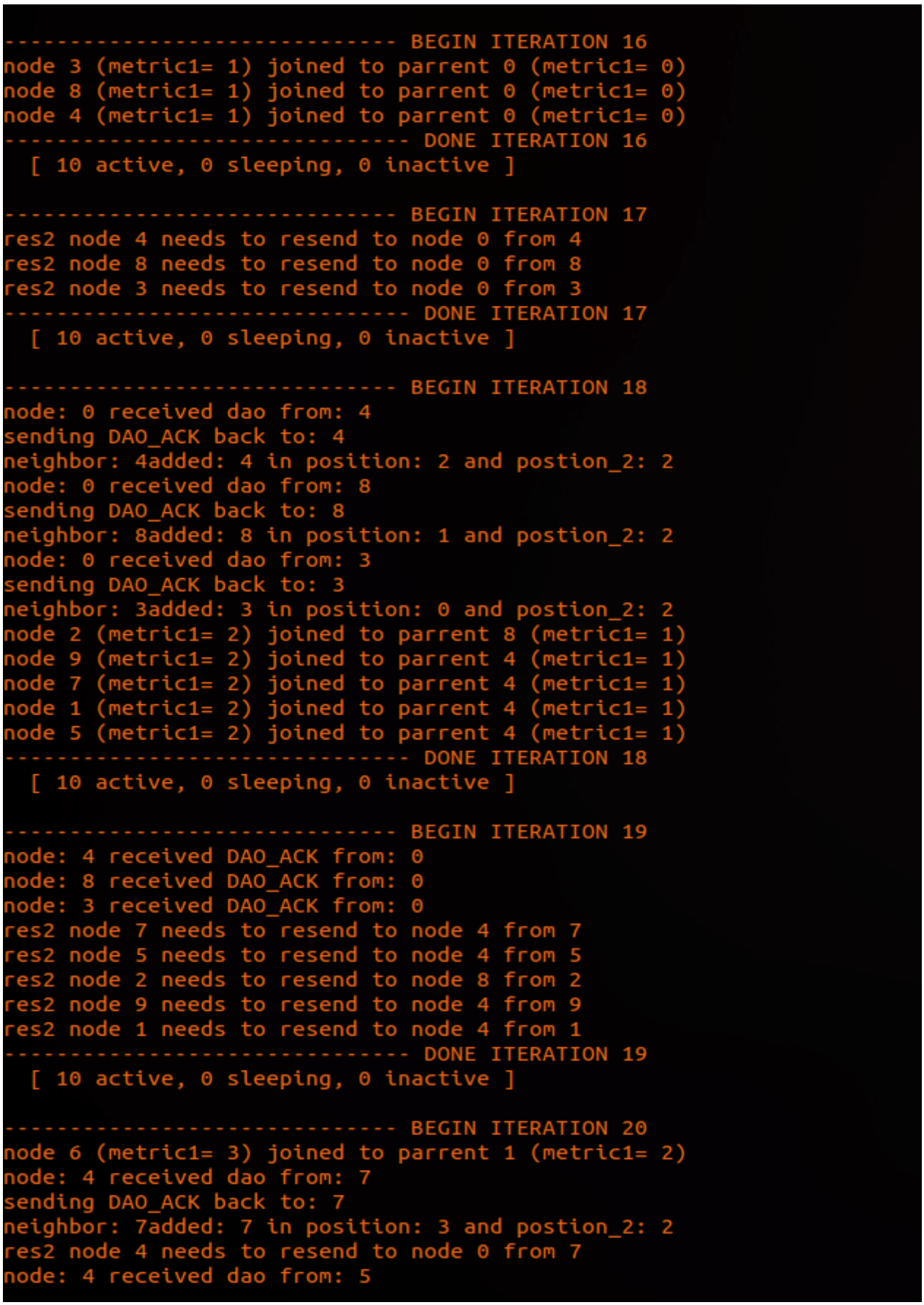
4.3.1 Το περιβάλλον εξομοίωσης Shawn

Το Shawn[4] είναι ένα περιβάλλον προσομοίωσης για ασύρματα δίκτυα αισθητήρων υλοποιημένο σε C++ με μοναδικά χαρακτηριστικά για την ανάπτυξη πρωτοκόλλων και εφαρμογών. Με την αφαίρεση των χαμηλότερων επίπεδων, το Shawn μπορεί να χρησιμοποιηθεί για προσομοίωση τεραστίων δικτύων σε εύλογο χρονικό διάστημα δίνοντας την άνεση στον προγραμματιστή να ασχοληθεί μόνο με την δημιουργία και εκτέλεση του πρωτοκόλλου.

Το Shawn έχει επεκταθεί ώστε να υποστηρίζει εκτέλεση και προσομοίωση κώδικα ο οποίος έχει γραφεί σε Wiselib. Για αυτόν τον λόγο επιλέχτηκε ως περιβάλλον προσομοίωσης για τα πειράματα μου.



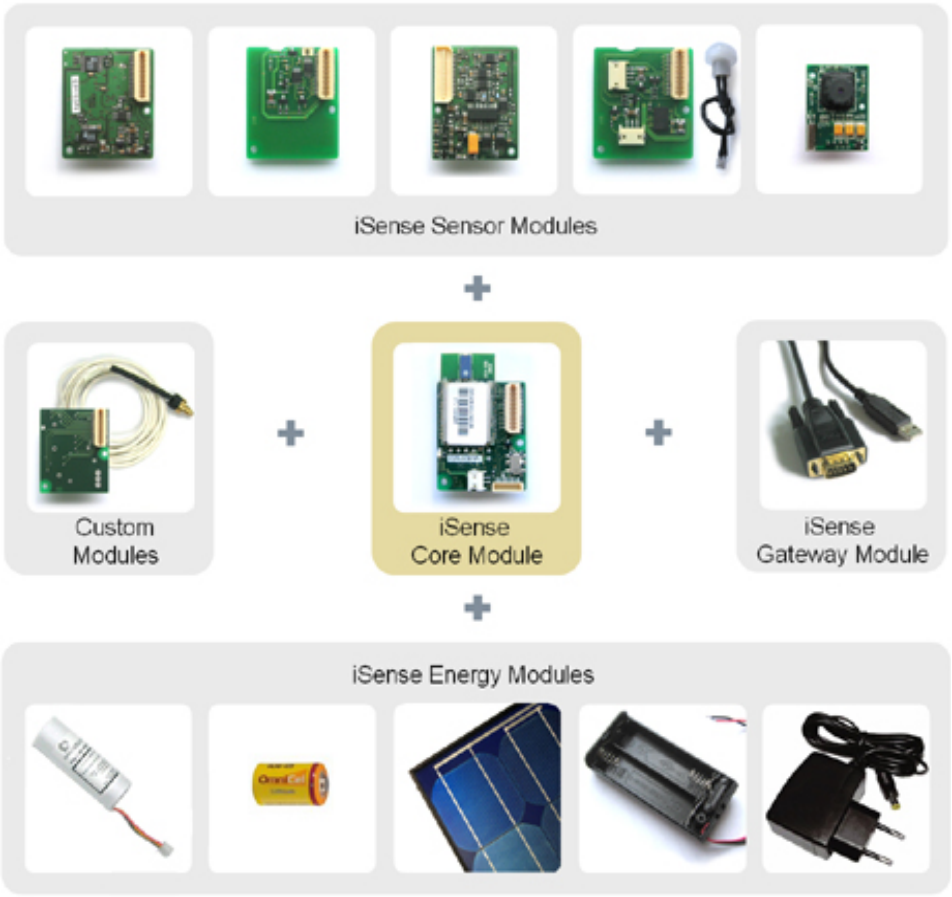
Εικόνα 4.3.1: Εκτέλεση του νέου πρωτοκόλλου στο περιβάλλον shawn



Εικόνα 4.3.2: Εκτέλεση του πρωτοκόλλου RPL στο περιβάλλον shawn

4.3.2 Οι συσκευές iSense

Για τους σκοπούς της διπλωματικής εργασίας, χρησιμοποιήθηκαν οι συσκευές isense[3] της εταιρίας coalesenses. Πιο συγκεκριμένα, στη διάθεση μας είχαμε τις συσκευές του ινστιτούτου τεχνολογίας υπολογιστών. Τα isense core modules παρέχουν την βάση της τμηματικής πλατφόρμας isense για οποιοδήποτε δίκτυο αισθητήρων και των ανάλογων εφαρμογών τους.



Εικόνα 4.3.3: Η βασική μονάδα iSense και διάφορες μονάδες για αναβάθμιση

Τα τεχνικά χαρακτηριστικά τους είναι τα επακόλουθα:

Controller:
32 Bit RISC, 16MHz
96 kB RAM, 128 KB Flash
2.4GHz IEEE 802.15.4
Data rate: 250 kBit/s
Range: up to 400m
Hardware Encryption

High Precision RTC

Voltage regulator

Low power
Communication: ~30 mA
Controller: ~10 mA
Sleep: ~5μA
Extendable via 34 Pin interface
2x UART, 4x AD, 1x DA, 9x GPIO
12C, SPI

Εικόνα 4.3.4: Η βασική μονάδα iSense

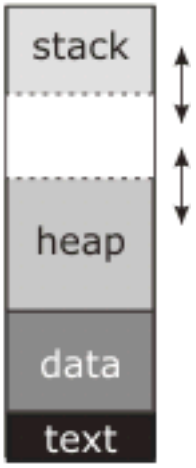
Τα iSense παρέχουν μεγάλη αποδοτικότητα με μικρο ενεργειακό κόστος. Ο ασύρματος

μικροελεγκτής JN5148 παρέχει μεγάλη υπολογιστική δύναμη και προσφέρεται για έναν μεγάλο αριθμό επεκτάσεων.

Επιπλέον είναι εξοπλισμένο με ασύρματη επικοινωνία (IEEE 802.15.4) για ταχύτατες μεταφορές δεδομένων. Παρέχεται κρυπτογράφηση AES και η απόσταση εκπομπής μπορεί να φτάσει μέχρι τα 400 μετρά.

Τέλος, το iSense έχει υψηλής ακριβείας ρολόι.

Το isense core module 1 το οποίο χρησιμοποιήθηκε για αυτή την διπλωματική, έχει μόλις 96 KB RAM βάζοντας ακόμα έναν περιορισμό ως προς την χρησιμοποίηση της μνήμης. Επιπλέον, υπήρχε πάντα ο κίνδυνος σύγκρουσης μεταξύ του stack και του heap. Πάνω σε αυτόν τον περιορισμό θα αναλυθούν κάποια από τα αποτελέσματα των πειραμάτων που έλαβαν χωρά.



Εικόνα 4.3.5: Διάγραμμα μνήμης.

4.3.4 Σενάρια εκτέλεσης

Τα σενάρια που εκτελέστηκαν κατά την διάρκεια των πειραμάτων στον εξομοιωτή είχαν σκοπό να ελέγξουν την σωστή λειτουργία των δυο πρωτοκόλλων, συμπεριλαμβανόμενου των δυο επιπλέον λειτουργιών που υπάρχουν στο iRPL. Κάθε σενάριο αποτελούταν από διαφορετικού μεγέθους δίκτυα, ξεκινώντας από 25 συσκευές και φτάνοντας στις 150 συσκευές. Παρακάτω θα αναλυθούν τα αποτελέσματα.

Στις πραγματικές συσκευές όμως, τα αποτελέσματα φάνηκαν πριν καν ολοκληρωθούν τα πειράματα. Ενώ το iRPL λειτουργούσε πάντα, το πρότυπο RPL είχε ένα μεγάλο πρόβλημα. Το πρόβλημα ήταν πως λόγω των περιορισμών σε μνήμη των συσκευών, αν ο πινάκας είχε μέγεθος μεγαλύτερο από 13×13 (δηλαδή για υποστήριξη 13 συσκευών), το πρωτόκολλο κατέρρεε καθώς δεν υπήρχε διαθέσιμη μνήμη για την λειτουργία του. Βλέπουμε άμεσα πόσο είναι το κόστος σε μνήμη ενός πρωτοκόλλου όπου χρειάζεται πίνακες δρομολόγησης. Μπόρεσα να παραβλέψω το πρόβλημα καθώς ήξερα την μορφολογία του δικτύου και να θέσω έναν μικρότερο πίνακα ώστε να επιβεβαιώσω την σωστή λειτουργία του πρότυπου RPL.

Παρόλα αυτά, ένα πρωτόκολλο για κατανεμημένα συστήματα θα πρέπει να υποστηρίζει παραπάνω από απλά 12 συσκευές καθώς δεν μπορεί ο χρήστης να γνωρίζει

πάντα την τοπολογία του δικτύου ώστε να επωφεληθεί όπως επωφελήθηκα εγώ. Αντιθέτως, το iRPL λειτούργησε χωρίς κανένα πρόβλημα στην εκτέλεση του.

Άμεσα μπορούμε να δούμε πως ένα πολύ σοβαρό πρόβλημα των πρωτοκόλλων δρομολόγησης που κάνουν χρήση πινάκων δρομολόγησης λύνεται με την ιδέα που υπάρχει στο iRPL.

4.3.5 Αποτελέσματα πειραμάτων

Τα αποτελέσματα που θα παρουσιάσουμε είναι από τα πειράματα που εκτελέστηκαν στον εξομοιωτή. Ο λόγος που επιλέξαμε να τα εκτελέσουμε στον εξομοιωτή είναι γιατί δεν είχαμε στην διάθεση μας πολλές συσκευές και επιπλέον είδαμε πως υπήρχε πρόβλημα με το πρότυπο RPL αν υπήρχαν παραπάνω από 12 συσκευές.

Σε κάθε πείραμα, ένας τυχαία επιλεγόμενος κόμβος άρχιζε να στέλνει 100000 πακέτα προς έναν άλλον πάλι τυχαία επιλεγμένο κόμβο. Για κάθε πείραμα η τοπολογία του δικτύου δημιουργούταν τυχαία. Τα αποτελέσματα είναι ο συνολικός χρόνος που απαιτήθηκε μόνο από τον αρχικό κόμβο ώστε να βρεθεί ο επόμενος κόμβος προώθησης. Τα νούμερα των αποτελεσμάτων δεν αντιπροσωπεύουν πραγματικούς χρόνους αλλά τους χρόνους όπως τους μετράει ο εξομοιωτής shawn.

25 κόμβοι	**Χρόνος RPL**	**Χρόνος iRPL**	**50 κόμβοι**	**Χρόνος RPL**	**Χρόνος iRPL**
Πείραμα 1	70000	70000	**Πείραμα 1**	80000	80000
Πείραμα 2	70000	70000	**Πείραμα 2**	70000	70000
Πείραμα 3	70000	70000	**Πείραμα 3**	80000	70000
Πείραμα 4	80000	70000	**Πείραμα 4**	70000	70000
Πείραμα 5	80000	80000	**Πείραμα 5**	70000	70000
Πείραμα 6	70000	60000	**Πείραμα 6**	100000	80000
Πείραμα 7	80000	60000	**Πείραμα 7**	80000	80000
Πείραμα 8	70000	70000	**Πείραμα 8**	70000	60000
Πείραμα 9	80000	90000	**Πείραμα 9**	80000	60000
Πείραμα 10	80000	70000	**Πείραμα 10**	80000	70000
Μέσος όρος	75000	71000	**Μέσος όρος**	78000	71000

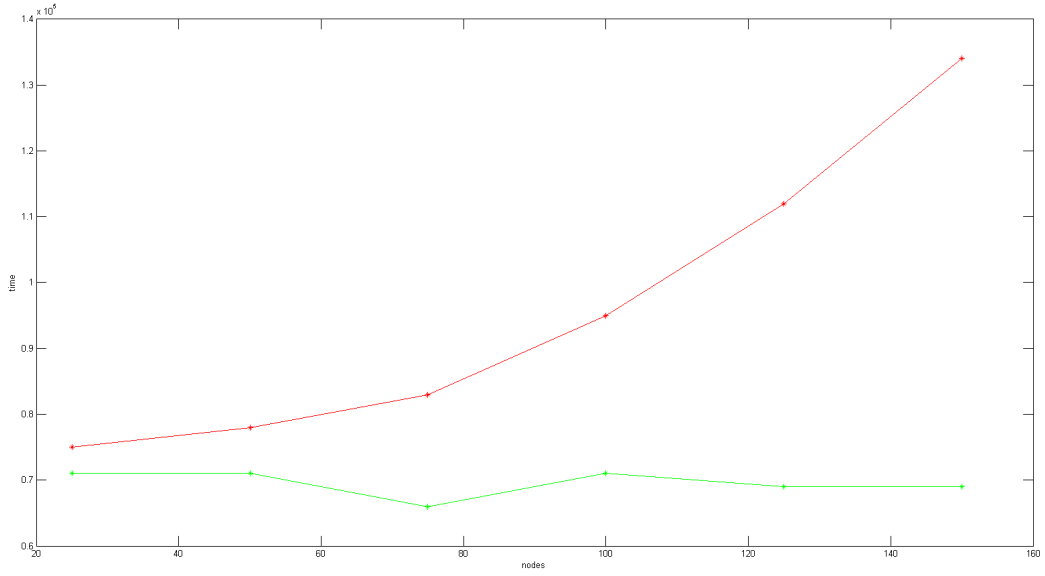
75 κόμβοι	**Χρόνος RPL**	**Χρόνος iRPL**	**100 κόμβοι**	**Χρόνος RPL**	**Χρόνος iRPL**
Πείραμα 1	80000	70000	**Πείραμα 1**	90000	70000
Πείραμα 2	80000	70000	**Πείραμα 2**	90000	70000
Πείραμα 3	70000	60000	**Πείραμα 3**	90000	70000
Πείραμα 4	80000	70000	**Πείραμα 4**	110000	70000
Πείραμα 5	80000	60000	**Πείραμα 5**	90000	80000
Πείραμα 6	80000	60000	**Πείραμα 6**	90000	70000

Πείραμα 7	90000	70000	**Πείραμα 7**	90000	70000
Πείραμα 8	90000	60000	**Πείραμα 8**	100000	80000
Πείραμα 9	100000	70000	**Πείραμα 9**	100000	60000
Πείραμα 10	80000	70000	**Πείραμα 10**	100000	70000
Μέσος όρος	83000	66000	**Μέσος όρος**	95000	71000

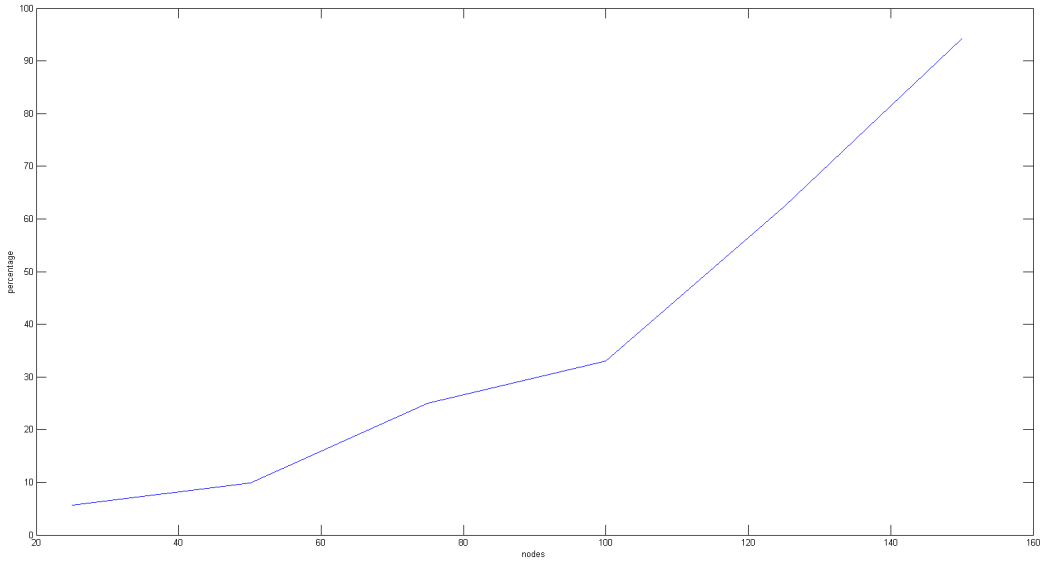
125 κόμβοι	**Χρόνος RPL**	**Χρόνος iRPL**	**150 κόμβοι**	**Χρονος RPL**	**Χρόνος iRPL**
Πείραμα 1	100000	70000	**Πείραμα 1**	130000	70000
Πείραμα 2	120000	60000	**Πείραμα 2**	100000	60000
Πείραμα 3	130000	70000	**Πείραμα 3**	130000	70000
Πείραμα 4	120000	70000	**Πείραμα 4**	150000	70000
Πείραμα 5	120000	70000	**Πείραμα 5**	150000	70000
Πείραμα 6	140000	80000	**Πείραμα 6**	150000	80000
Πείραμα 7	100000	60000	**Πείραμα 7**	130000	60000
Πείραμα 8	100000	70000	**Πείραμα 8**	140000	70000
Πείραμα 9	100000	70000	**Πείραμα 9**	150000	70000
Πείραμα 10	90000	70000	**Πείραμα 10**	110000	70000
Μέσος όρος	112000	69000	**Μέσος όρος**	134000	69000

Ποσοστό βελτιστοποίησης

Κόμβοι	%
25	5.6
50	9.8
75	25
100	33
125	62.3
150	94.2



Εικόνα 4.5.3: Γραφική παράσταση των χρόνων εκτέλεσης των 2 πρωτοκόλλων. Η κόκκινη γραμμή αντιπροσωπεύει το πρωτόκολλο RPL ενώ η πράσινη το νέο πρωτόκολλο



Εικόνα 4.3.6: Γραφική παράσταση της βελτιστοποίησης ως προς τις ενεργές συσκευές στο δίκτυο

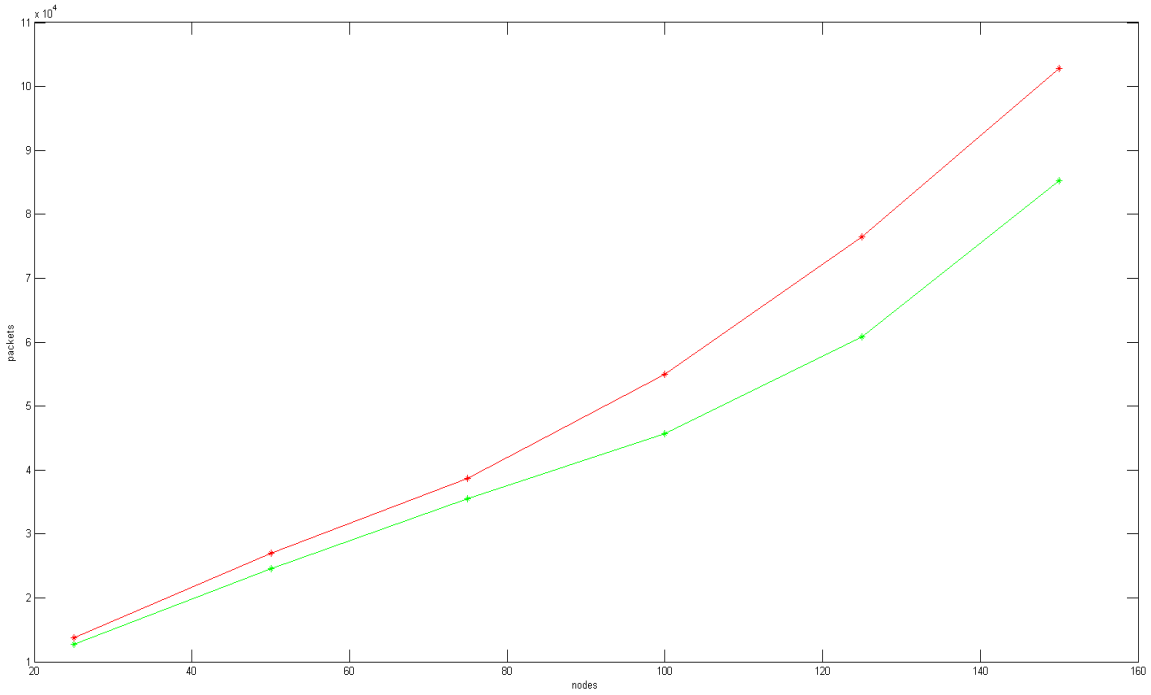
Τα αποτελέσματα που είχαμε από τα πειράματα, βλέπουμε πως ταιριάζουν με την θεωρητική μας ανάλυση. Βλέπουμε πως οι χρόνοι για την διαδικασία της δρομολόγησης του iRPL ήταν σταθεροί για οποιοδήποτε αριθμό κόμβων ενώ οι χρόνοι του πρότυπου RPL μεγάλωναν εκθετικά. Μάλιστα στο σενάριο με τους 150 κόμβους, το ποσοστό βελτίωσης ήταν κοντά στο 100% (94.2%). Αν υποθέσουμε ένα πολύ μεγάλο δίκτυο της τάξης των 10000 συσκευών, το iRPL θα ήταν αρκετές φορές πιο γρήγορο από το συμβατικό RPL με τους πίνακες δρομολόγησης.

Σε κάθε πείραμα είχαμε μια τυχαία τοπολογία κόμβων και σε αυτί την τοπολογία εφαρμόζαμε τα εκάστοτε πρωτοκολλά μέχρι την λειτουργιά μετάδοσης DAO πακέτων. Με την βοήθεια του εξομοιωτή Shawn μπορέσαμε να καταγράψουμε τον συνολικό αριθμό πακέτων που στάλθηκαν κατά την εκτέλεση των δυο αυτών πρωτοκόλλων.

25 κόμβοι	**πακέτα RPL**	**πακέτα IRPL**	**50 κόμβοι**	**πακέτα RPL**	**πακέτα iRPL**
Πείραμα 1	13375	11207	**Πείραμα 1**	24170	24973
Πείραμα 2	12345	11649	**Πείραμα 2**	26876	23605
Πείραμα 3	14617	11924	**Πείραμα 3**	28496	26563
Πείραμα 4	14169	14762	**Πείραμα 4**	27048	25946
Πείραμα 5	13999	14420	**Πείραμα 5**	28462	21709
Μέσος όρος	13701	12792.4	**Μέσος όρος**	27010.4	24559.2

75 κόμβοι	**πακέτα RPL**	**πακέτα IRPL**	**100 κόμβοι**	**πακέτα RPL**	**πακέτα iRPL**
Πείραμα 1	38183	38611	**Πείραμα 1**	52248	47568
Πείραμα 2	38125	31433	**Πείραμα 2**	57582	44504
Πείραμα 3	37715	34904	**Πείραμα 3**	53606	39691
Πείραμα 4	38653	37904	**Πείραμα 4**	53860	47403
Πείραμα 5	40813	35009	**Πείραμα 5**	57354	49309
Μέσος όρος	38697.8	35572.2	**Μέσος όρος**	54930	45695

125 κόμβοι	**πακέτα RPL**	**πακέτα IRPL**	**150 κόμβοι**	**πακέτα RPL**	**πακέτα iRPL**
Πείραμα 1	77419	59893	**Πείραμα 1**	101852	86851
Πείραμα 2	76833	54269	**Πείραμα 2**	99696	85218
Πείραμα 3	73991	63639	**Πείραμα 3**	97860	82299
Πείραμα 4	78645	62049	**Πείραμα 4**	112888	84332
Πείραμα 5	75645	64076	**Πείραμα 5**	102006	87529
Μέσος όρος	76506.6	60785.2	**Μέσος όρος**	102860.6	85245.8



Εικόνα 4.3.7: Γραφική παράσταση των πακέτων κατά την εκτέλεση των 2 πρωτοκόλλων. Η κόκκινη γραμμή αντιπροσωπεύει το πρωτόκολλο RPL ενώ η πράσινη το νέο πρωτόκολλο

Παρόλο που στην θεωρητική ανάλυση είχαμε βρει πως στο βέλτιστο σενάριο το πρότυπο RPL έχει καλύτερη πολυπλοκότητα επικοινωνίας, είδαμε πως στα πειράματα αυτό το σενάριο δεν μπορεί να συμβεί πότε για αυτόν τον λόγο για κάθε διαφορετικό δίκτυο το πρότυπο RPL είναι πάντα χειρότερο.

4.4 Άλλες υλοποιήσεις

4.4.1 ContikiRPL

Λίγο καιρό πριν αρχίσω την υλοποίηση της εργασίας, είχε δημοσιευτεί το αντίστοιχο πρωτόκολλο από την ομάδα της πλατφόρμας Contiki. Η δουλειά τους αν και ολοκληρωμένη λειτουργούσε μόνο στο ContikiOS[7]. Αντιθέτως ο κώδικας που παρουσιάζω, επειδή έχει φτιαχτεί στην πλατφόρμα wiselib, μπορεί να υποστηρίξει και το ContikiOS. Τέλος, το ContikiRPL[8] βασίζεται στο πρότυπο RPL και δεν είναι βελτιστοποιημένο σαν το δικό μου, άρα τα ίδια προβλήματα που υπήρχαν στο πρότυπο RPL υπάρχουν και στο ContikiRPL.

Κεφάλαιο 5 – Τελευταία λόγια / Μελλοντική εργασία

5.1 Τελευταία λογία και συμπεράσματα

Φτάνοντας στο τέλος αυτής της εργασίας, ύστερα από όλη την ανάλυση όπου υπήρξε, μπορούμε με ασφάλεια να πούμε πως αποδείχτηκε ότι υπάρχει μια ακόμα καλύτερη υλοποίηση του πρωτοκόλλου RPL από αυτή που δημιουργήθηκε αρχικά. Καλύτερη ως προς την πολυπλοκότητα δρομολόγησης, πολυπλοκότητα επικοινωνίας και χρησιμοποίησης μνήμης.

5.2 Μελλοντική εργασία

Παρόλο που τελικά φτιάξαμε ένα βελτιωμένο πρωτόκολλο, και ενώ το iRPL είναι λειτουργικό, δεν βρίσκεται στην καλύτερη μορφή που θα μπορούσε να είχε. Αν κάποιος θα ήθελε να ασχοληθεί, μια άμεση αναβάθμιση που θα έπρεπε να γίνει θα ήταν στην αναζήτηση και δημιουργία πινάκα γειτόνων.

Ένα ακόμα μέρος που θα μπορούσε κάποιος να ασχοληθεί, θα ήταν να φτιάξει και τα δυο RPL να δημιουργούν πολλαπλά στιγμιότυπα στο ίδιο δίκτυο. Κατά πάσα πιθανότητα, στο πρότυπο RPL θα δημιουργηθούν ακόμα περισσότερα προβλήματα, ενώ το iRPL θα μπορέσει να κρατήσει την ίδια πολυπλοκότητα στην δρομολόγηση και στην επικοινωνία, όπως και τις ίδιες απαιτήσεις σε μνήμη. Αυτό από μόνο του θα μπορούσε να ήταν μια πολύ καλή ερευνά.

Βιβλιογραφία

[1] Michael Richardson, JP. Vasseur, Adrian Farrel, Rene Struik, Daniel King,
Routing Over Low power and Lossy networks, 2012-2013
https://datatracker.ietf.org/wg/roll/charter/

[2] P. Levis, A. Tavakoli, S. Dawson-Haggerty,
Overview of Existing Routing Protocols for Low Power and Lossy Networks, 2009

[3] T. Winter, Ed. P. Thubert, Ed., A. Brandt, T. Clausen, J. Hui, R. Kelsey,
P. Levis, K. Pister, R. Struik, JP. Vasseur,
RPL: IPv6 Routing Protocol for Low power and Lossy Networks, 2011
http://tools.ietf.org/html/draft-ietf-roll-rpl-19

[4] T. Baumgartner, I. Chatzigiannakis, S. P. Fekete, C. Koninis, A. Kröller and A. Pyrgelis.
Wiselib: A genereic algorithm library for heterogeneous sensor networks. In *Proceedings of the Seventh European Conference on Wireless Sensor Networks (EWSN 2010),*
Pages 162-177, Coimbra, Portugal, February 2010. Springer-Verlag LNCS 5970
https://github.com/ibr-alg/wiselib/wiki

[5] Coalsenses isense module:
http://www.coalesenses.com/index.php/products/solutions/isense-devices/

[6] A. Kröller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer.
Shawn: A new approach to simulating wireless sensor networks. In
Proceeding of the Design, Analysis, and Simulation of Distributed Systems Symposium 2005 (DASD' 05), pages 117-124, Apr. 2005

[7] ContikiOS: http://www.contiki-os.org/

[8] Nicolas Tsiftes, Joakim Eriksson, and Adam Dunkels,
Low-Power Wireless IPv6 Routing with ContikiRPL, 2010.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.167.5802&rep=rep1&type=pdf

ΠΑΡΑΡΤΗΜΑ Α – Κώδικας iRPL

```
/*****************************************************************************
* This file is part of improved-RPL.
*
* improved-RPL is free software: you can redistribute it and/or modify
* it under the terms of the GNU LesserGeneral Public License as published
* by the Free Software Foundation, either version 3 of the License, or
* any later version.
*
* improved-RPL is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public License
* along with improved-RPL. If not, see <http://www.gnu.org/licenses/>.
*****************************************************************************
/


#include "external_interface/external_interface.h"
#include "algorithms/routing/tree/tree_routing.h"
#include <string.h>
#include <stdlib.h>
#include <limits.h>

#define ROOT_NODE 0x0 //0x2028
#define SEND_TO 000200
#define SEND_FROM 6

#define NEIGHBOR_SIZE 256 //256 //65534 is the maximum value. do NOT set this
bigger than 255 never!


#define ROUTN_WIDTH 256 //256
#define ROUTN_LEN 32

#define PARENT_TIMEOUT 15

#define OF_TO_USE 0x00
#define MIN_HIGH_TREE 0x00
#define HOP_WEIGHT 0
#define LQI_WEIGHT 1
#define DATA_SIZE 32

#define NEIGHBOR_DIS 0
#define NEIGHBOR_DIS_ACK 1
#define DIO 2
#define DAO 3
#define DAO_RESPONSE 4
#define DAO_ACK 5
#define UP 0
#define DOWN 1
#define DATA 7
#define REPAIR 8
#define DISBAND 9

typedef wiselib::OSMODEL Os;
```

```
class rpl
{
public:


    struct metrics
    {
        uint16_t metric1;
        uint16_t metric2;
        uint16_t metric3;
        uint16_t metric4;
    };

    typedef metrics metrics_t;

    struct rpl_dio
    {
        uint16_t root;
        uint16_t parent;
        uint8_t type;
        uint8_t hops;
        uint16_t id;
        uint8_t version;
        uint8_t dest;
        uint8_t given;
        uint16_t rank;
        uint8_t instance_id;
        metrics_t dio_metrics;

    };
    typedef rpl_dio rpl_dio_t;

    struct header
    {
        uint8_t type;
        uint16_t to_node;
        uint16_t from_node;
        char to_routN[ROUTN_LEN];
        char from_routN[ROUTN_LEN];
    };
    typedef header header_t;

    struct rpl_dag
    {
        uint8_t of;

        uint8_t version;
        uint8_t id;
        uint8_t instance_id;
        uint8_t joined;
        uint8_t hops;
        uint16_t rank;
        uint16_t root;
        uint16_t parent;
        char routN[ROUTN_LEN];
        char previous_routN[ROUTN_LEN];
        metrics_t dag_metrics;
    };
    typedef rpl_dag rpl_dag_t;

    struct rpl_dao
```

```
    {
        uint8_t instance_id;
        uint8_t dao_sequence;
    };

    typedef rpl_dao rpl_dao_t;

    struct rpl_dao_response
    {
        uint8_t instance_id;
        uint8_t dao_sequence;
    };

    typedef rpl_dao_response rpl_dao_response_t;

    struct rpl_data
    {
        uint8_t instance_id;
        uint8_t direction;
        uint8_t hops;
        //char from[ROUTN_LEN];
        char payload[DATA_SIZE];
    };

    typedef rpl_data rpl_data_t;

//-------------------------------------------------------------------------------
----------------------

    void init( Os::AppMainParameter& value )
    {
        radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet( value );
        timer_ = &wiselib::FacetProvider<Os, Os::Timer>::get_facet( value );
        debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet( value );


        radio_->reg_recv_callback<rpl,
        &rpl::receive_radio_message>( this );
        // timer_->set_timer<rpl, &rpl::>( 5000, this, 0 );



        timer_->set_timer<rpl, &rpl::discover_neighbors>( 2000, this, 0 );

        rpl_dag_structure.root=0xffff;

        rpl_dag_structure.rank=0xffff;

        rpl_dag_structure.dag_metrics.metric1=0xffff;

        rpl_dag_structure.joined=0xff;

        rpl_dag_structure.of=OF_TO_USE;

        memset(rpl_dag_structure.routN, NULL, ROUTN_LEN);

        memset(rpl_dag_structure.previous_routN, NULL, ROUTN_LEN);

        needs_repair_=0;
        dao_flag=0;
        /// na figei...
```

```
        power=1;

        dao_counter=0;
        rpl_dag_structure.joined=0xff;
        for(uint16_t j=0; j<NEIGHBOR_SIZE; j++)
        {
            neighbors[j]=0xffff;
        }


        for (uint8_t i=0; i<=ROUTN_WIDTH-1; i++)
        {
            if(i==ROUTN_WIDTH-1)
                break;

            pending[i][0]=0xffff;
        }


        sequence_counter_=0x00;

        if(radio_->id()==ROOT_NODE)
        {
            rpl_dag_structure.rank=0;
            rpl_dag_structure.root=radio_->id();
            strcpy(rpl_dag_structure.routN, "00");
            rpl_dag_structure.dag_metrics.metric1=0;
            rpl_dag_structure.version=5;
            rpl_dag_structure.joined=0;
            rpl_dag_structure.hops=0;

            debug_->debug("IM ROOT");

        }
        if(radio_->id()==SEND_FROM)
        {
           timer_->set_timer<rpl, &rpl::test_mess>( 60000, this, 0 );
        }
       // if(radio_->id()==4)
       // {
       //     timer_->set_timer<rpl, &rpl::deactivate>( 27000, this, 0 );
       // }
        timer_->set_timer<rpl, &rpl::dio_output>( 2000, this, 0 );
        timer_->set_timer<rpl, &rpl::check_pending>( 4000, this, 0 );

        timer_->set_timer<rpl, &rpl::publish_routn>( 55000, this, 0 );
    }
//-------------------------------------------------------------------------------
---------------------------------------

  void publish_routn( void* )
    {
        debug_->debug("node %x PUBLISHES routN %s\n", radio_->id(),
rpl_dag_structure.routN);
    }
//-------------------------------------------------------------------------------
---------------------------------------
    void deactivate ( void* )
    {
        power=0;
    }
```

```
//-------------------------------------------------------------------------------
-------------------------------------
    void discover_neighbors( void* )
    {
        if(power==1)
        {
            header_t my_header;
            my_header.type=NEIGHBOR_DIS;
            my_header.from_node=radio_->id();
            radio_->send( Os::Radio::BROADCAST_ADDRESS, sizeof(header_t),
( Os::Radio::block_data_t*)&my_header );
            timer_->set_timer<rpl, &rpl::discover_neighbors>( 10000, this, 0 );
        }
    }

//-------------------------------------------------------------------------------
-------------------------------------

    void handle_neighbor_discovery (header_t inbox_header)
    {
        //  debug_->debug("node %x received nd from node %x",radio_->id(),
inbox_header.from_ipv6[0]);
        if (rpl_dag_structure.joined==0 &&
inbox_header.from_node==rpl_dag_structure.parent)
        {
            parent_timeout_=0;
            //debug_->debug("node %d reseting parent_timeout\n",radio_->id());
        }
        header_t header;
        header.type=NEIGHBOR_DIS_ACK;

        handle_neighbor_discovery_ack(inbox_header);

        header.to_node=inbox_header.from_node;
        header.from_node=radio_->id();
        radio_->send( Os::Radio::BROADCAST_ADDRESS, sizeof(header_t),
( Os::Radio::block_data_t*)&header );
    }
//-------------------------------------------------------------------------------
-------------------------------------

    void handle_neighbor_discovery_ack(header_t inbox_header)
    {

        bool used=false;
        uint16_t position=0xffff;
        for (uint16_t i=0; i<NEIGHBOR_SIZE; i++)
        {
            if (position==0xffff && neighbors[i]==0xffff)
            {
                position=i;
            }
            if(neighbors[i]==inbox_header.from_node)
            {
                used=true;
                break;
            }
        }
        if(used==false)
        {
            neighbors[position]=inbox_header.from_node;
```

```
            //debug_->debug("node: %x added %x to list\n", radio_->id(),
inbox_header.from_node);
        }
    }
//-------------------------------------------------------------------------
-------------------------------------
    void dio_output (void *)
    {
        if(power==1)
        {
            if((rpl_dag_structure.joined==0) &&
(strlen(rpl_dag_structure.routN)<ROUTN_LEN))
            {
                rpl_dio_t dio;

                dio.id = radio_->id();
                dio.dio_metrics = rpl_dag_structure.dag_metrics;
                dio.version = rpl_dag_structure.version;
                dio.instance_id = rpl_dag_structure.instance_id;
                dio.root = rpl_dag_structure.root;
                dio.hops = rpl_dag_structure.hops;

                header_t header;
                header.type=DIO;
                header.from_node=radio_->id();

                strncpy(header.from_routN, rpl_dag_structure.routN,
strlen(rpl_dag_structure.routN));
                for (uint16_t i=0; i<=NEIGHBOR_SIZE; i++)
                {
                    if(neighbors[i]==0xffff)
                    {
                        break;
                    }
                    else
                    {
                        header.to_node=neighbors[i];
                    }
                    uint16_t size;
                    size = sizeof(header_t)+sizeof(rpl_dio_t);
                    unsigned char message[size];
                    memcpy(message, &header, sizeof(header_t));
                    memcpy(message+sizeof(header_t),&dio, sizeof(rpl_dio_t));
                    // debug_->debug("metric1: %x, version: %x",
dio.dio_metrics.metric1, dio.version);

                    radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);
                }
            }

            timer_->set_timer<rpl, &rpl::dio_output>( 2000, this, 0 );

        }
    }

//-------------------------------------------------------------------------
-------------------------------------
    void dio_input (header_t header, rpl_dio_t input_dio, uint8_t LQI)
    {

        if((rpl_dag_structure.joined!=0)) /* auto tha figei */ //&&
```

```
(strncmp(header.from_routN, rpl_dag_structure.previous_routN,
strlen(rpl_dag_structure.previous_routN))!=0))
        {
            //debug_->debug("RESULTTTTTTTT: %d",strncmp(header.from_routN,
rpl_dag_structure.previous_routN, strlen(rpl_dag_structure.previous_routN)));

            if((strlen(rpl_dag_structure.previous_routN)>0) )
            {
                if((strncmp(header.from_routN,
rpl_dag_structure.previous_routN, strlen(rpl_dag_structure.previous_routN)-
2)==0))
                {
                    //do nothing
                }
                else
                {
                    {

                        uint16_t i;
                        for (i=0; i<=NEIGHBOR_SIZE; i++)
                        {
                            if(i==NEIGHBOR_SIZE || i==0xffff)
                            {
                                break;
                            }

                            if(neighbors[i]==header.from_node)
                            {
                                break;
                            }
                        }

                        if(i!=NEIGHBOR_SIZE)
                        {

                            switch(rpl_dag_structure.of)
                            {
                            case MIN_HIGH_TREE:

temp_rank=min_high_tree_OF(input_dio.dio_metrics, LQI);
                                // debug_->debug("node %d temp_rank:
%d\n",radio_->id(),temp_rank);
                                break;
                            }

                            if(temp_rank<rpl_dag_structure.rank)
                            {
                                rpl_dag_structure.rank = temp_rank;
                                rpl_dag_structure.hops = input_dio.hops + 1;
                                rpl_dag_structure.parent = input_dio.id;
                                rpl_dag_structure.version = input_dio.version;
                                rpl_dag_structure.dag_metrics.metric1 =
input_dio.dio_metrics.metric1+1;
                                rpl_dag_structure.root = input_dio.root;
                                if(radio_->id()!=ROOT_NODE && dao_flag!=1)
                                {
                                    timer_->set_timer<rpl,
&rpl::call_dao_output>( 2000, this, 0 );
                                    dao_flag=1;
                                }
                                debug_->debug("node %x (hops= %x) with rank %d
```

```
rejoined to parrent %x ,%s, %s\n", radio_-
>id(),rpl_dag_structure.hops,rpl_dag_structure.rank, rpl_dag_structure.parent,
rpl_dag_structure.previous_routN ,header.from_routN);
                                dao_counter=0;

                            }
                        }
                    }
                }
            }
            else
            {

                uint16_t i;
                for (i=0; i<=NEIGHBOR_SIZE; i++)
                {
                    if(i==NEIGHBOR_SIZE || i==0xffff)
                    {
                        break;
                    }

                    if(neighbors[i]==header.from_node)
                    {
                        break;
                    }
                }

                if(i!=NEIGHBOR_SIZE)
                {

                    switch(rpl_dag_structure.of)
                    {
                    case MIN_HIGH_TREE:
                        temp_rank=min_high_tree_OF(input_dio.dio_metrics, LQI);
                        // debug_->debug("node %d temp_rank: %d\n",radio_-
>id(),temp_rank);
                        break;
                    }

                    if(temp_rank<rpl_dag_structure.rank)
                    {

                        rpl_dag_structure.rank = temp_rank;
                        rpl_dag_structure.hops = input_dio.hops + 1;
                        rpl_dag_structure.parent = input_dio.id;
                        rpl_dag_structure.version = input_dio.version;
                        rpl_dag_structure.dag_metrics.metric1 =
input_dio.dio_metrics.metric1+1;
                        rpl_dag_structure.root = input_dio.root;
                        if(radio_->id()!=ROOT_NODE && dao_flag!=1)
                        {
                            timer_->set_timer<rpl, &rpl::call_dao_output>(
2000, this, 0 );
                            dao_flag=1;
                        }
                        debug_->debug("node %x (hops= %x) with rank %d joined
to parrent %x (metric1= %x)\n", radio_-
>id(),rpl_dag_structure.hops,rpl_dag_structure.rank, rpl_dag_structure.parent,
input_dio.dio_metrics.metric1);
                        dao_counter=0;
```

```
                    }
                }
            }
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------

    uint16_t min_high_tree_OF(metrics_t input_metrics, uint8_t LQI)
    {

        if (input_metrics.metric1==0xffff)
            return input_metrics.metric1;
        else
        {
            uint16_t temp=(input_metrics.metric1+1)*HOP_WEIGHT + LQI *
LQI_WEIGHT;
            return temp;
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------
    void dao_output(rpl_dag_t input_dag)
    {
        if(rpl_dag_structure.joined!=0)
        {
            debug_->debug("node %x tries to send dao to %x\n",radio_->id(),
rpl_dag_structure.parent);
            header_t header;
            rpl_dao_t dao;

            header.type=DAO;
            header.from_node = radio_->id();
            header.to_node = input_dag.parent;
            dao.instance_id = input_dag.instance_id;
            //dao.dao_sequence = sequence_counter_;

            uint16_t size;
            size = sizeof(header_t)+sizeof(rpl_dao_t);
            unsigned char message[size];
            memcpy(message, &header, sizeof(header_t));
            memcpy(message+sizeof(header_t),&dao, sizeof(rpl_dao_t));
            radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);

            timer_->set_timer<rpl, &rpl::call_dao_output>( 4000, this, 0 );
        }

    }

//-------------------------------------------------------------------------------
-------------------------------------
    void call_dao_output(void *)
    {
        if (dao_counter>=0 && dao_counter<=3)
        {
            dao_output(rpl_dag_structure);
            dao_counter++;
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------
```

```
    void dao_response(uint16_t from)
    {

        if(rpl_dag_structure.joined==0)
        {
            int16_t temp = RoutingNumGenerator(from);

            if(temp!=-1)
            {
                header_t header;
                rpl_dao_response_t dao_response;
                header.type=DAO_RESPONSE;
                header.to_node=from;
                header.from_node=radio_->id();

                dao_response.dao_sequence=temp;
                dao_response.instance_id=rpl_dag_structure.instance_id;
                char next[2];
                char hex[]= {"0123456789abcdef"};
                uint8_t i, j;
                i=temp/16;
                j=temp%16;
                sprintf(next,"%c%c",hex[i],hex[j]);
                strcpy(header.to_routN,rpl_dag_structure.routN);
                strcat(header.to_routN,next);

                uint16_t size;
                size = sizeof(header_t)+sizeof(rpl_dao_response_t);
                unsigned char message[size];
                memcpy(message, &header, sizeof(header_t));
                memcpy(message+sizeof(header_t),&dao_response,
sizeof(rpl_dao_response_t));
                radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);
                // debug_->debug("process %d with %s received connection
question from %d and replied %s\n",radio_->id(), RoutN_, from, mess->Routing);
            }

        }

    }
//-------------------------------------------------------------------------------
---------------------------------------
    void handle_dao_response (header_t inbox_header, rpl_dao_response_t
inbox_dao_res)
    {
        if(inbox_header.from_node==rpl_dag_structure.parent)
        {
            strcpy(rpl_dag_structure.routN, inbox_header.to_routN);
            rpl_dag_structure.joined=0;

            debug_->debug("node %x joined with routN %s\n", radio_->id(),
rpl_dag_structure.routN);

            header_t header;
            rpl_dao_response_t dao_res;
            header.type=DAO_ACK;
            header.to_node=inbox_header.from_node;
            header.from_node=radio_->id();

            dao_res.instance_id=rpl_dag_structure.instance_id;
            dao_res.dao_sequence=inbox_dao_res.dao_sequence;
```

```
            uint16_t size;
            size = sizeof(header_t)+sizeof(rpl_dao_response_t);
            unsigned char message[size];
            memcpy(message, &header, sizeof(header_t));
            memcpy(message+sizeof(header_t),&dao_res,
sizeof(rpl_dao_response_t));
            radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);
            if(needs_repair_==1)
            {
                debug_->debug("node %d rejoined\n", radio_->id());
                send_repair();
            }
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------
    void handle_dao_ack (header_t inbox_header, rpl_dao_response dao_ack)
    {
        if(pending[dao_ack.dao_sequence][0]==inbox_header.from_node)
        {
            pending[dao_ack.dao_sequence][0]=0xffff;
            given[dao_ack.dao_sequence]=true;
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------
    void check_pending ( void* )
    {
        //debug_->debug("test\n");
        //repair procedure

        if((parent_timeout_<=PARENT_TIMEOUT) && (rpl_dag_structure.joined==0))
        {

            if ((rpl_dag_structure.joined==0) &&(radio_->id() != ROOT_NODE))
            {
                parent_timeout_++;
               // debug_->debug("node %d increasing parent timeout at %d \n",
radio_->id(), parent_timeout_);
            }
        }
        else if((parent_timeout_>PARENT_TIMEOUT) &&
(rpl_dag_structure.joined==0))
        {
            debug_->debug("node %d lost its parent\n", radio_->id());
            rpl_dag_structure.root=0xffff;

            rpl_dag_structure.rank=0xffff;

            rpl_dag_structure.dag_metrics.metric1=0xffff;

            rpl_dag_structure.joined=0xff;
            strcpy(rpl_dag_structure.previous_routN, rpl_dag_structure.routN);
            // debug_->debug("QUICK DEBUG: %s\n", rpl_dag_structure.routN);

            memset(rpl_dag_structure.routN, NULL, ROUTN_LEN);
            needs_repair_=1;
            ///TODO: Repair();
        }
```

```
        for(uint8_t i=0; i<= ROUTN_WIDTH-1; i++)
        {
            if(i==ROUTN_WIDTH-1)
            {
                break;
            }

            if (pending[i][0] != 0xffff)
            {
                if(pending[i][1]>6) //this is for timout of waiting for ack
                {
                    pending[i][1]=0xff;
                    pending[i][0]=0xffff;
                    //                debug_->debug("i1: %x", i);
                }
                else if(pending[i][1]>=4)
                {
                    dao_response(pending[i][0]);
                }

                if (pending[i][1] < 0xff)
                {
                    pending[i][1]++;
                }

            }


        }
        timer_->set_timer<rpl, &rpl::check_pending>( 4000, this, 0 );

    }
//-------------------------------------------------------------------------------
---------------------------------------
    void data_output ( char dest[ROUTN_LEN], char from[ROUTN_LEN], char
input_data[DATA_SIZE])
    {
        //debug_->debug("%x will try to send\n",radio_->id());

        header_t header;
        rpl_data_t rpl_data;

        header.type=DATA;

        if(strncmp(rpl_dag_structure.routN, dest,
strlen(rpl_dag_structure.routN))==0)
        {
            //debug_->debug("%s : %s :
%d\n",rpl_dag_structure.routN,dest,strlen(rpl_dag_structure.routN));
            //debug_->debug("%d\n",strncmp(rpl_dag_structure.routN, dest,
strlen(rpl_dag_structure.routN)));
            //debug_->debug("node %x tries to send DOWN\n",radio_->id());
            rpl_data.direction=DOWN;
        }
        else
        {
            //debug_->debug("node %x tries to send UP\n",radio_->id());
            rpl_data.direction=UP;
        }
        rpl_data.hops=rpl_dag_structure.hops;
        strcpy(header.to_routN,dest);
```

```
        strcpy(header.from_routN,from);

        rpl_data.instance_id=rpl_dag_structure.instance_id;
        //strcpy(rpl_data.from, from);
        strcpy(rpl_data.payload,input_data);


        uint16_t size;
        size = sizeof(header_t)+sizeof(rpl_data_t);
        unsigned char message[size];
        memcpy(message, &header, sizeof(header_t));
        memcpy(message+sizeof(header_t),&rpl_data, sizeof(rpl_data_t));
        radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);

    }
//-------------------------------------------------------------------------------
---------------------------------------
    void data_input(header_t inbox_header, rpl_data_t input_data)
    {
        //debug_->debug("node %d received DATA:1\n",radio_->id());
        //debug_->debug("node %d: debug 1: direction: %d, from: %s, routN %s,
routN len: %d, hops: %d\n",radio_-
>id(),input_data.direction,inbox_header.from_routN,rpl_dag_structure.routN,strl
en(rpl_dag_structure.routN),input_data.hops );
        if((input_data.direction==UP) && (strncmp(rpl_dag_structure.routN,
inbox_header.from_routN,strlen(rpl_dag_structure.routN))==0) &&
(input_data.hops==rpl_dag_structure.hops+1))
            //if ((strncmp(inbox_header.from_routN, inbox_header.to_routN,
strlen(rpl_dag_structure.routN)+2)!=0) &&
(strlen(inbox_header.from_routN)==strlen(rpl_dag_structure.routN)+2))
        {
            // debug_->debug("node %d received DATA:2\n",radio_->id());
            if(strcmp(inbox_header.to_routN,rpl_dag_structure.routN)==0)
            {
                //debug_->debug("received DATA to %s, from: %s, data: %s\n",
rpl_dag_structure.routN,inbox_header.from_routN, input_data.payload);
                //debug_->debug("received DATA to %s, data: %s\n",
rpl_dag_structure.routN, input_data.payload);
            }
            else //if(strncmp(inbox_header.from_routN, rpl_dag_structure.routN,
strlen(rpl_dag_structure.routN))==0)
            {
                // debug_->debug("resending from
%s\n",rpl_dag_structure.routN);
                data_output(inbox_header.to_routN, inbox_header.from_routN,
input_data.payload);
            }
        }
        else if((input_data.direction==DOWN) &&
(strncmp(rpl_dag_structure.routN, inbox_header.to_routN,
strlen(rpl_dag_structure.routN))==0) &&
(input_data.hops==rpl_dag_structure.hops-1))
            //else if((strncmp(inbox_header.from_routN,
rpl_dag_structure.routN, strlen(rpl_dag_structure.routN)-2)==0) &&
(strlen(inbox_header.from_routN)==strlen(rpl_dag_structure.routN)-2))
        {
            if(strcmp(inbox_header.to_routN,rpl_dag_structure.routN)==0)
            {
                //debug_->debug("received DATA to %s, from: %s, data: %s\n",
rpl_dag_structure.routN,inbox_header.from_routN, input_data.payload);
                //debug_->debug("received DATA to %s, data: %s\n",
```

```
rpl_dag_structure.routN, input_data.payload);
            }
            else //if(strncmp(inbox_header.to_routN, rpl_dag_structure.routN,
strlen(rpl_dag_structure.routN))==0)
            {
                // debug_->debug("resending from
%s\n",rpl_dag_structure.routN);
                data_output(inbox_header.to_routN, inbox_header.from_routN
,input_data.payload);
            }
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------
    void send_repair()
    {
        if(strlen(rpl_dag_structure.routN)==ROUTN_LEN)
        {
            //SEND disband message
            send_disband();
        }
        else
        {
            //generate new routN for each child.
            for(uint8_t i=0; i<ROUTN_WIDTH; i++)
            {
                if(i==0xff)
                {
                    break;
                }
                if (given[i]==true)
                {
                    //send message
                    header_t header;
                    header.type=REPAIR;
                    header.from_node=radio_->id();

                    uint8_t j, k;
                    char next[2];
                    char hex[]= {"0123456789abcdef"};

                    j=i/16;
                    k=i%16;

                    sprintf(next,"%c%c",hex[j],hex[k]);
                    strcpy(header.from_routN, rpl_dag_structure.routN);
                    strcat(header.from_routN,next);
                    strcpy(header.to_routN,rpl_dag_structure.previous_routN);
                    strcat(header.to_routN,next);

                    uint16_t size;
                    size = sizeof(header_t);
                    unsigned char message[size];
                    memcpy(message, &header, sizeof(header_t));

                    radio_->send( Os::Radio::BROADCAST_ADDRESS,
sizeof(header_t), message);
                }
            }
            debug_->debug("node %d send repair\n", radio_->id());
        }
```

```
    }

//-------------------------------------------------------------------------------
---------------------------------------

    void handle_repair(header_t inbox_header)
    {

        if(strncmp(rpl_dag_structure.routN, inbox_header.to_routN,
strlen(rpl_dag_structure.routN))==0 &&
(inbox_header.from_node==rpl_dag_structure.parent))
        {
            //change prefix
            strcpy(rpl_dag_structure.previous_routN, rpl_dag_structure.routN);
            memset(rpl_dag_structure.routN, NULL, ROUTN_LEN);
            strcpy(rpl_dag_structure.routN, inbox_header.from_routN);
            debug_->debug("node %d repaired\n", radio_->id());
            debug_->debug("node %d received repair from %d, with to_routN: %s,
from_routN: %s\n", radio_->id(), inbox_header.from_node, inbox_header.to_routN,
inbox_header.from_routN);
            send_repair();
        }
    }

//-------------------------------------------------------------------------------
---------------------------------------
    void send_disband ()
    {
        for(uint8_t i=0; i<ROUTN_WIDTH; i++)
        {
            if(i==0xff)
            {
                break;
            }
            if (given[i]==true)
            {
                header_t header;
                header.type=DISBAND;
                header.from_node=radio_->id();

                uint8_t j, k;
                char next[2];
                char hex[]= {"0123456789abcdef"};

                j=i/16;
                k=i%16;

                sprintf(next,"%c%c",hex[j],hex[k]);
                strcpy(header.from_routN, rpl_dag_structure.routN);
                strcpy(header.to_routN,rpl_dag_structure.routN);
                strcat(header.to_routN,next);

                uint16_t size;
                size = sizeof(header_t);
                unsigned char message[size];
                memcpy(message, &header, sizeof(header_t));

                radio_->send( Os::Radio::BROADCAST_ADDRESS, sizeof(header_t),
message);
            }
        }
```

```
        debug_->debug("node %d send disband\n", radio_->id());

    }
//-------------------------------------------------------------------------------
-------------------------------------
    void handle_disband ( header_t inbox_header )
    {
        if(strncmp(rpl_dag_structure.routN, inbox_header.to_routN,
strlen(rpl_dag_structure.routN))==0 &&
(inbox_header.from_node==rpl_dag_structure.parent))
        {
            rpl_dag_structure.root=0xffff;
            rpl_dag_structure.rank=0xffff;
            rpl_dag_structure.dag_metrics.metric1=0xffff;
            rpl_dag_structure.joined=0xff;
            rpl_dag_structure.of=OF_TO_USE;
            memset(rpl_dag_structure.routN, NULL, ROUTN_LEN);
            memset(rpl_dag_structure.previous_routN, NULL, ROUTN_LEN);

            send_disband();
        }

    }
//-------------------------------------------------------------------------------
-------------------------------------
    void test_mess( void* )
    {
        for(message_counter=0;message_counter<100000;message_counter++){
        data_output("0001000000",rpl_dag_structure.routN,"test_data");
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------
//-------------------------------------------------------------------------------
-------------------------------------
    void receive_radio_message( Os::Radio::node_id_t from, Os::Radio::size_t
len, Os::Radio::block_data_t *buf,  const Os::Radio::ExtendedData& exdata )
    {
        header_t header;
        memcpy(&header, buf, sizeof(header_t));

        switch(header.type)
        {
        case NEIGHBOR_DIS:
            handle_neighbor_discovery(header);
            // free(header);
            //TODO: complete this
            break;
        case NEIGHBOR_DIS_ACK:
            if(header.to_node==radio_->id())
            {
                handle_neighbor_discovery_ack(header);
            }
            break;
        case DIO:
            if(header.to_node==radio_->id())
            {
                rpl_dio_t temp_dio;
                memcpy(&temp_dio, buf+sizeof(header_t), sizeof(rpl_dio_t));
                dio_input(header, temp_dio, exdata.link_metric());
            }
```

```
            break;
        case DAO:
            if(header.to_node==radio_->id())
            {
                debug_->debug("node %x received DAO from %x\n", radio_-
>id(),from);
                //rpl_dao_t temp_dao;
                //memcpy(&temp_dao, buf+sizeof(header_t), sizeof(rpl_dao_t));
                dao_response(header.from_node);
            }
            break;
        case DAO_RESPONSE:
            if(header.to_node==radio_->id())
            {
                debug_->debug("node %x received DAO_RES from %x\n", radio_-
>id(),from);
                rpl_dao_response_t temp_dao_res;
                memcpy(&temp_dao_res, buf+sizeof(header_t),
sizeof(rpl_dao_response_t));
                handle_dao_response(header, temp_dao_res);
            }
            break;
        case DAO_ACK:
            if(header.to_node==radio_->id())
            {
                debug_->debug("node %x received DAO_ACK from %x\n", radio_-
>id(),from);
                rpl_dao_response_t temp_dao_ack;
                memcpy(&temp_dao_ack, buf+sizeof(header_t),
sizeof(rpl_dao_response_t));
                handle_dao_ack(header,temp_dao_ack);
            }
            break;
        case DATA:
            //debug_->debug("node %x received DATA from %x\n", radio_-
>id(),from);
            rpl_data_t temp_data;
            memcpy(&temp_data, buf+sizeof(header_t), sizeof(rpl_data_t));
            data_input(header,temp_data);
            break;
        case REPAIR:
            handle_repair(header);
            break;
        case DISBAND:
            handle_disband(header);
            break;
        }
    }
//-------------------------------------------------------------------------------
---------------------------------------

    int16_t RoutingNumGenerator (uint16_t from)
    {

        for(uint8_t i=0; i<=ROUTN_WIDTH-1; i++)
        {
            if(i==ROUTN_WIDTH-1)
                break;

            if (pending[i][0]==from)
                return(i);
```

```
        }

        for (uint8_t i=0; i<=ROUTN_WIDTH-1; i++)
        {
            if(i==ROUTN_WIDTH-1)
            {
                break;
            }
            if(given[i]==false && pending[i][0]==0xffff)
            {
                pending[i][0]=from;
                return(i);

            }
        }
        return (-1);

    }
//-------------------------------------------------------------------------------
-------------------------------------

//-------------------------------------------------------------------------------
-------------------------------------
private:
    Os::Radio::self_pointer_t radio_;
    Os::Timer::self_pointer_t timer_;
    Os::Debug::self_pointer_t debug_;
    rpl_dag_t rpl_dag_structure;
    uint16_t neighbors[NEIGHBOR_SIZE];
    uint8_t sequence_counter_;
    uint8_t parent_timeout_;
    uint16_t temp_rank;
    uint8_t dao_flag;
    uint32_t message_counter;
    uint8_t needs_repair_;
    uint8_t power;
    uint8_t dao_counter;
    bool given[ROUTN_WIDTH];
    uint16_t pending[ROUTN_WIDTH][2];
};
// -------------------------------------------------------------------------
wiselib::WiselibApplication<Os, rpl> rpl_app;
// -------------------------------------------------------------------------
void application_main( Os::AppMainParameter& value )
{
    rpl_app.init( value );
}
```

ΠΑΡΑΡΤΗΜΑ Β – Κώδικας πρότυπου RPL

```
/*****************************************************************************
* This file is part of improved-RPL.
*
* improved-RPL is free software: you can redistribute it and/or modify
* it under the terms of the GNU LesserGeneral Public License as published
* by the Free Software Foundation, either version 3 of the License, or
* any later version.
*
* improved-RPL is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public License
* along with improved-RPL. If not, see <http://www.gnu.org/licenses/>.
*****************************************************************************
/

#include "external_interface/external_interface.h"
#include "algorithms/routing/tree/tree_routing.h"

#define ROOT_NODE 0x296
#define SEND_TO 0xca3
#define SEND_FROM 0x1cde

#define NEIGHBOR_SIZE 40 //65534 is the maximum value. do NOT set this bigger
than 255 never!
#define NEIGHBOR_WIDTH 40
#define SENT_QUEUE 16      //254 is the maximum value. do NOT set this bigger
than 255 never!

#define OF_TO_USE 0x00
#define MIN_HIGH_TREE 0x00
#define DATA_SIZE 32

#define NEIGHBOR_DIS 0
#define NEIGHBOR_DIS_ACK 1
#define DIO 2
#define DAO 3
#define DAO_ACK 4
#define DATA 5

typedef wiselib::OSMODEL Os;

class rpl
{
public:



    struct metrics
    {
        uint16_t metric1;
        uint16_t metric2;
        uint16_t metric3;
```

```
    uint16_t metric4;
};

typedef metrics metrics_t;

struct rpl_dio
{
    uint16_t root;
    uint16_t parent;
    uint8_t type;
    uint8_t hops;
    uint16_t id;
    uint8_t version;
    uint8_t dest;
    uint8_t given;
    uint16_t rank;
    uint8_t instance_id;
    char from[32];
    char Routing[32];
    metrics_t dio_metrics;

};
typedef rpl_dio rpl_dio_t;

struct header
{
    uint8_t type;
    uint16_t to_node;
    uint16_t from_node;
   // uint16_t to_ipv6[8];
  //  uint16_t from_ipv6[8];
};
typedef header header_t;

struct rpl_dag
{
    uint8_t of;

    uint8_t version;
    uint8_t id;
    uint8_t instance_id;
    uint8_t joined;
    uint8_t hops;
    uint16_t rank;
            uint16_t root;
    uint16_t parent;
    metrics_t dag_metrics;
};
typedef rpl_dag rpl_dag_t;


struct rpl_dao
{
    uint16_t to;
    uint16_t from;
    uint8_t instance_id;
    uint8_t dao_sequence;
};

    typedef rpl_dao rpl_dao_t;
```

```
struct rpl_dao_ack
{
    uint16_t to;
    uint16_t from;
    uint8_t instance_id;
    uint8_t dao_sequence;
};

typedef rpl_dao_ack rpl_dao_ack_t;

struct rpl_data
{
    uint16_t to;
    uint16_t from;
    uint8_t instance_id;
    unsigned char payload[DATA_SIZE];
};

typedef rpl_data rpl_data_t;

void init( Os::AppMainParameter& value )
{
    radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet( value );
    timer_ = &wiselib::FacetProvider<Os, Os::Timer>::get_facet( value );
    debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet( value );



    radio_->reg_recv_callback<rpl,
    &rpl::receive_radio_message>( this );

    timer_->set_timer<rpl, &rpl::discover_neighbors>( 2000, this, 0 );

    rpl_dag_structure.root=0xffff;

    rpl_dag_structure.rank=0xffff;

    rpl_dag_structure.dag_metrics.metric1=0xffff;

    rpl_dag_structure.of=OF_TO_USE;

   for(uint16_t j=0;j<NEIGHBOR_SIZE; j++)
   {
        for (uint16_t i=0;i<NEIGHBOR_WIDTH; i++)
        {
            // debug_->debug("node: %x value: %x",radio_->id(), i);
            neighbors[j][i]=0xffff;


        }
   }
     //debug_->debug("%x",rpl_dag_structure.of);
    for (uint8_t i=0;i<=SENT_QUEUE;i++)
    {
        sent_timer_[i]=0xff;
        if(i==0xff)
        {
            break;
        }
    }
```

```
        sequence_counter_=0x00;

        if(radio_->id()==ROOT_NODE)
        {
            rpl_dag_structure.rank=0;
            rpl_dag_structure.root=radio_->id();

            rpl_dag_structure.dag_metrics.metric1=0;
            rpl_dag_structure.version=5;

            debug_->debug("IM ROOT");

        }
        if(radio_->id()==SEND_FROM)
        {
           timer_->set_timer<rpl, &rpl::send>( 60000, this, 0 );
        }

                    timer_->set_timer<rpl,
        &rpl::dio_output>( 15000, this, 0 );

           timer_->set_timer<rpl, &rpl::check_sent_timer>( 4000, this, 0 );

    }
//-------------------------------------------------------------------------------
---------------------------------------
  //  void discover_neighbors( void* )
//-------------------------------------------------------------------------------
---------------------------------------
    void discover_neighbors( void* )
    {
        header_t my_header;
        my_header.type=NEIGHBOR_DIS;
        my_header.from_node=radio_->id();
        radio_->send( Os::Radio::BROADCAST_ADDRESS, sizeof(header_t),
( Os::Radio::block_data_t*)&my_header );
       // timer_->set_timer<rpl, &rpl::discover_neighbors>( 2000, this, 0 );
    }

//-------------------------------------------------------------------------------
---------------------------------------

    void handle_neighbor_discovery (header_t inbox_header)
    {
      //  debug_->debug("node %x received nd from node %x",radio_->id(),
inbox_header.from_ipv6[0]);
        header_t header;
        header.type=NEIGHBOR_DIS_ACK;
       // header.to_ipv6[0]=inbox_header.from_ipv6[0];
       // header.from_ipv6[0]=radio_->id();

       // test
       handle_neighbor_discovery_ack(inbox_header);

        header.to_node=inbox_header.from_node;
        header.from_node=radio_->id();
        radio_->send( Os::Radio::BROADCAST_ADDRESS, sizeof(header_t),
( Os::Radio::block_data_t*)&header );
    }
//-------------------------------------------------------------------------------
```

```
--------------------------------------

    void handle_neighbor_discovery_ack(header_t inbox_header)
    {
        bool used=false;
        uint16_t position=0xffff;
        for (uint16_t i=0; i<NEIGHBOR_SIZE; i++)
        {
            if (position==0xffff && neighbors[i][0]==0xffff)
            {
                position=i;
            }
            if(neighbors[i][0]==inbox_header.from_node)
            {
                used=true;
                break;
            }
        }
        if(used==false)
        {
            neighbors[position][0]=inbox_header.from_node;
            debug_->debug("node: %x added %x to list\n", radio_->id(),
inbox_header.from_node);
        }
    }
//------------------------------------------------------------------------------
--------------------------------------
void dio_output (void *)
{
    rpl_dio_t dio;

    dio.id = radio_->id();
    dio.dio_metrics = rpl_dag_structure.dag_metrics;
    dio.version = rpl_dag_structure.version;
    dio.instance_id = rpl_dag_structure.instance_id;
    dio.root = rpl_dag_structure.root;

    header_t header;
    header.type=DIO;
    header.from_node=radio_->id();
    for (uint16_t i=0;i<=NEIGHBOR_SIZE;i++)
        {
        if(neighbors[i][0]==0xffff)
        {
            break;
        }
        else
        {
            header.to_node=neighbors[i][0];
        }
        uint16_t size;
        size = sizeof(header_t)+sizeof(rpl_dio_t);
        unsigned char message[size];
        memcpy(message, &header, sizeof(header_t));
        memcpy(message+sizeof(header_t),&dio, sizeof(rpl_dio_t));
   // debug_->debug("metric1: %x, version: %x", dio.dio_metrics.metric1,
dio.version);

        radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);
   }
```

```
    timer_->set_timer<rpl, &rpl::dio_output>( 2000, this, 0 );

}

//-------------------------------------------------------------------------------
---------------------------------------
void dio_input (header_t header, rpl_dio_t input_dio)
{
    uint16_t i;
          for (i=0;i<=NEIGHBOR_SIZE;i++)
          {
              if(i==NEIGHBOR_SIZE || i==0xffff)
              {
                  break;
              }

              if(neighbors[i][0]==header.from_node)
              {
                  break;
              }
          }

          if(i!=NEIGHBOR_SIZE)
          {

              switch(rpl_dag_structure.of)
            {
                case MIN_HIGH_TREE:
                temp_rank=min_high_tree_OF(input_dio.dio_metrics);
                // debug_->debug("node %d temp_rank: %d\n",radio_-
>id(),temp_rank);
                break;
            }

            if(temp_rank<rpl_dag_structure.rank)
            {
                rpl_dag_structure.rank = temp_rank;
                rpl_dag_structure.hops = input_dio.hops + 1;
                rpl_dag_structure.parent = input_dio.id;
                rpl_dag_structure.version = input_dio.version;
                rpl_dag_structure.dag_metrics.metric1 =
input_dio.dio_metrics.metric1+1;
                rpl_dag_structure.root = input_dio.root;
                if(radio_->id()!=ROOT_NODE){
                timer_->set_timer<rpl, &rpl::send_dao>( 1000, this, 0 );
                }
                debug_->debug("node %x (metric1= %x) joined to parrent %x
(metric1= %x)\n", radio_->id(),rpl_dag_structure.dag_metrics.metric1,
rpl_dag_structure.parent, input_dio.dio_metrics.metric1);
            }
          }
}
//-------------------------------------------------------------------------------
---------------------------------------
void dao_output (rpl_dag_t input_dag, uint16_t from)
{


   for (sequence_counter_=0;sequence_counter_<=SENT_QUEUE;sequence_counter_++)
   {
              if(sequence_counter_==0xff || sequence_counter_==SENT_QUEUE)
```

```
                {
                    break;
                }
        if(sent_node_[sequence_counter_]==from && sent_node_[sequence_counter_]!
=0xffff)
        {
            debug_->debug("res1 node %x needs to resend to node %x from %x\n",
radio_->id(), input_dag.parent, from);
            break;

        }

    }

    if(sequence_counter_==SENT_QUEUE)
    {
         for
(sequence_counter_=0;sequence_counter_<=SENT_QUEUE;sequence_counter_++)
            {
                if(sequence_counter_==SENT_QUEUE || sequence_counter_==0xff)
                {
                    break;
                }
                if(sent_timer_[sequence_counter_]==0xff)
                {

                    break;
                }

            }
    }
     if (sequence_counter_!=SENT_QUEUE)
     {
         debug_->debug("res2 node %x needs to resend to node %x from %x\n",
radio_->id(), input_dag.parent, from);

        header_t header;
        rpl_dao_t dao;

        header.type=DAO;
        header.from_node = radio_->id();
        header.to_node = input_dag.parent;
       // debug_->debug("parent: %x", sequence_counter_  );

        dao.to=input_dag.root;
        dao.from=from;


        dao.instance_id = input_dag.instance_id;
        dao.dao_sequence = sequence_counter_;
        sent_timer_[sequence_counter_]=0;
        sent_node_[sequence_counter_]=from;

        uint16_t size;
        size = sizeof(header_t)+sizeof(rpl_dao_t);
        unsigned char message[size];
        memcpy(message, &header, sizeof(header_t));
        memcpy(message+sizeof(header_t),&dao, sizeof(rpl_dao_t));
        radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);
```

```
    }
}
//-------------------------------------------------------------------------------
-------------------------------------
void dao_input(header_t header, rpl_dao_t input_dao)
{
   // if(input_dao.to==radio_->id())                      //einai gia tin
parousa diergasia to dao?
  //  {
        //handle dao o.O
 //   }
 //   else
  //  {
  if (input_dao.instance_id == rpl_dag_structure.instance_id)
  {
        send_dao_ack(header.from_node, input_dao.dao_sequence);
        debug_->debug("sending DAO_ACK back to: %x\n",header.from_node);
        uint16_t position;

        for (position=0;position<=NEIGHBOR_SIZE; position++)  // briskei ton
geitona pou to esteile
        {
            if(neighbors[position][0]==header.from_node)
            {
                //position=i;
                neighbors[position][1]=header.from_node;
                debug_->debug("neighbor: %x",neighbors[position][0]);
                break;
            }

            if(position==NEIGHBOR_SIZE)
            {
                break;
            }
        }

        if (position!=NEIGHBOR_SIZE)                        //psaxnei na brei
an to i diergasia pou proorizete to dao uparxei sto rout table. an oxi, tin
prosthetei.
        {
            uint16_t position_2;
            uint16_t temp=0xffff;
            for (position_2=2;position_2<=NEIGHBOR_SIZE; position_2++)
            {
                if(neighbors[position][position_2]==0xffff && temp==0xffff)
                {
                    //neighbors[position][position_2]=input_dao.from;
                   // break;
                   temp=position_2;
                }
                else if (neighbors[position][position_2]==input_dao.from)
                {
                    //debug_->debug("neighbors[position][position_2]:
%x",neighbors[position][position_2]);
                     break;
                }

                if(position_2==NEIGHBOR_SIZE)
                {
```

```
                    break;
                }
            }

            if (position_2==NEIGHBOR_SIZE)
            {
                 neighbors[position][temp]=input_dao.from;
                 debug_->debug("added: %x in position: %x and postion_2:
%x\n",neighbors[position][temp],position,temp);
            }
        }
        if (radio_->id()!=input_dao.to)
        {
            dao_output(rpl_dag_structure, input_dao.from);
        }
  }
   // }
}
//-------------------------------------------------------------------------------
-------------------------------------
void send_dao_ack (uint16_t dest, uint8_t sequence)
{
    header_t header;
    rpl_dao_ack_t dao_ack;
    header.type = DAO_ACK;
    header.to_node = dest;
    header.from_node = radio_->id();
    dao_ack.dao_sequence = sequence;
    dao_ack.instance_id = rpl_dag_structure.instance_id;

    uint16_t size;
    size = sizeof(header_t)+sizeof(rpl_dao_t);
    unsigned char message[size];
    memcpy(message, &header, sizeof(header_t));
    memcpy(message+sizeof(header_t),&dao_ack, sizeof(rpl_dao_ack_t));
    radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);
}

//-------------------------------------------------------------------------------
-------------------------------------
void handle_dao_ack (header_t header, rpl_dao_ack_t dao_ack)
{
    if(sent_timer_[dao_ack.dao_sequence]!=0xff)
    {
        sent_timer_[dao_ack.dao_sequence]=0xff;
        sent_node_[dao_ack.dao_sequence]=0xffff;
    }
}
//-------------------------------------------------------------------------------
-------------------------------------
void clear_sent_timer (uint8_t sequence)
{
    sent_timer_[sequence_counter_]=0xff;
}
//-------------------------------------------------------------------------------
-------------------------------------
void check_sent_timer ( void* )
{


    for(uint8_t i=0; i<= SENT_QUEUE; i++)
```

```
    {
            if(i==SENT_QUEUE)
            {
                break;
            }

        if (sent_timer_[i] != 0xff)
        {
            if(sent_timer_[i]>6) //this is for timout of waiting for ack
            {
                sent_timer_[i]=0xff;
                sent_node_[i]=0xffff;
            }
            else if(sent_timer_[i]>=4)
            {
                dao_output(rpl_dag_structure, sent_node_[i]);
            }

            if (sent_timer_[i] < 0xff)
            {
                sent_timer_[i]++;
            }

        }


    }
    timer_->set_timer<rpl, &rpl::check_sent_timer>( 4000, this, 0 );
}
//-------------------------------------------------------------------------------
---------------------------------------
void send_dao( void* )
{
    dao_output(rpl_dag_structure, radio_->id());
}

//-------------------------------------------------------------------------------
---------------------------------------
void data_output ( uint16_t dest, uint16_t from, unsigned char
input_data[DATA_SIZE])
{
    debug_->debug("%x will try to send\n",radio_->id());
    if (dest!=radio_->id())
    {

    header_t header;
    rpl_data_t data;

    header.type=DATA;
    data.to = dest;
    data.instance_id=rpl_dag_structure.instance_id;
    if(from==radio_->id())
    {
        data.from=radio_->id();
    }
    else
    data.from=from;

    header.from_node = radio_->id();
    header.to_node=0xffff; //flag (o Thanasis tha me misei gia ta tromera
sxolia m)
```

```
    for (uint16_t i=0; i<=NEIGHBOR_SIZE; i++)
    {
       // if(i==0xffff)
       // break;
       if(header.to_node!=0xffff)
       break;

       if (neighbors[i][0]==0xffff || dest== rpl_dag_structure.parent)
       {
           if(radio_->id()!=rpl_dag_structure.root)
           {
                header.to_node=rpl_dag_structure.parent;
                break;
           }
       }
       else
       {
           for (uint16_t j=1;j<=NEIGHBOR_SIZE; j++)
           {
                if(neighbors[i][j]==0xffff)
                break;

                if(neighbors[i][j]==dest)
                {
                    header.to_node=neighbors[i][0];
                    break;
                }

           }
       }
    }


    if(header.to_node!=0xffff)
    {
        debug_->debug("data.to: %x, data.from: %x, header.to: %x, header.from:
%x\n", data.to, data.from, header.to_node, header.from_node);
        memcpy(data.payload, input_data, DATA_SIZE);
        uint16_t size;
        size = sizeof(header_t)+sizeof(rpl_data_t);
        unsigned char message[size];
        memcpy(message, &header, sizeof(header_t));
        memcpy(message+sizeof(header_t),&data, sizeof(rpl_data_t));
        radio_->send( Os::Radio::BROADCAST_ADDRESS, size, message);
    }
    else
    debug_->debug("found none\n");
    }

}
//-------------------------------------------------------------------------------
---------------------------------------
void data_input(rpl_data_t data)
{
    if(data.instance_id==rpl_dag_structure.instance_id)
    {
        if(data.to==radio_->id())
        {
            //handle payload
```

```
            debug_->debug("node %x received DATA from %x\n", radio_->id(),
data.from);

        }
        else
        {
            data_output(data.to, data.from, data.payload);
        }
    }
}
//-------------------------------------------------------------------------
---------------------------------------
void send ( void* )
{
    data_output(SEND_TO, radio_->id(),(unsigned char *)"test_data");
}
//-------------------------------------------------------------------------
---------------------------------------

uint16_t min_high_tree_OF(metrics_t input_metrics)
{

    if (input_metrics.metric1==0xffff)
    return input_metrics.metric1;
    else
    return input_metrics.metric1+1;
}
//-------------------------------------------------------------------------
---------------------------------------
    void receive_radio_message( Os::Radio::node_id_t from, Os::Radio::size_t
len, Os::Radio::block_data_t *buf, const Os::Radio::ExtendedData& exdata )
    {
        //debug_->debug("LQI: %d", 255 - exdata.link_metric());
        header_t header;
        memcpy(&header, buf, sizeof(header_t));

        switch(header.type)
        {
        case NEIGHBOR_DIS:
            handle_neighbor_discovery(header);
            break;
        case NEIGHBOR_DIS_ACK:
            if(header.to_node==radio_->id())
            {
                handle_neighbor_discovery_ack(header);
            }
            break;
        case DIO:
            if(header.to_node==radio_->id())
            {
                rpl_dio_t temp_dio;
                memcpy(&temp_dio, buf+sizeof(header_t), sizeof(rpl_dio_t));
                dio_input(header, temp_dio);
            }
            break;
        case DAO:
            if(header.to_node==radio_->id())
            {
                debug_->debug("node: %x received dao from: %x\n",radio_-
>id(),header.from_node);
                rpl_dao_t temp_dao;
```

```
                memcpy(&temp_dao, buf+sizeof(header_t), sizeof(rpl_dao_t));
                dao_input(header, temp_dao);
            }
            break;
        case DAO_ACK:
            if(header.to_node==radio_->id())
            {
                debug_->debug("node: %x received DAO_ACK from: %x\n",radio_-
>id(),header.from_node);
                rpl_dao_ack_t temp_dao_ack;
                memcpy(&temp_dao_ack, buf+sizeof(header_t),
sizeof(rpl_dao_ack_t));
                handle_dao_ack(header, temp_dao_ack);
            }
            break;
        case DATA:
            if(header.to_node==radio_->id())
            {

                rpl_data_t temp_data;
                memcpy(&temp_data, buf+sizeof(header_t), sizeof(rpl_data_t));
                data_input(temp_data);
            }
        }
    }
//-------------------------------------------------------------------------------
-------------------------------------

//-------------------------------------------------------------------------------
-------------------------------------
private:
        Os::Radio::self_pointer_t radio_;
        Os::Timer::self_pointer_t timer_;
        Os::Debug::self_pointer_t debug_;

        uint16_t neighbors[NEIGHBOR_SIZE][NEIGHBOR_WIDTH];
        uint8_t hops_;
        uint16_t temp_rank;
        rpl_dag_t rpl_dag_structure;
        uint8_t sequence_counter_;
        uint8_t sent_timer_[SENT_QUEUE];
        uint16_t sent_node_[SENT_QUEUE];
    };
// -------------------------------------------------------------------------
    wiselib::WiselibApplication<Os, rpl> rpl_app;
// -------------------------------------------------------------------------
    void application_main( Os::AppMainParameter& value )
    {
        rpl_app.init( value );
    }
```

ΠΑΡΑΡΤΗΜΑ Γ – shawn configuration

```
prepare_world edge_model=list comm_model=disk_graph range=2
processors=wiselib_shawn_standalone
load_world file=test3.xml snapshot=0
simulation max_iterations=40
```

ΠΑΡΑΡΤΗΜΑ Δ – Τοπολογία παράδειγμα

```
<?xml version="1.0" ?>
<scenario>
<snapshot id="0">
    <node id="v0-RI8IaB-C">
        <location x="3.23764" y="1.37588" z="0" />
    </node>
    <node id="v1-RI8IaB-D">
        <location x="1.11281" y="2.1695" z="0" />
    </node>
    <node id="v2-RI8IaB-E">
        <location x="1.14647" y="0.502253" z="0" />
    </node>
    <node id="v3-RI8IaB-F">
        <location x="3.08735" y="1.57775" z="0" />
    </node>
    <node id="v4-RI8IaB-G">
        <location x="2.2398" y="2.32436" z="0" />
    </node>
    <node id="v5-RI8IaB-H">
        <location x="0.600142" y="2.02329" z="0" />
    </node>
    <node id="v6-RI8IaB-I">
        <location x="0.319845" y="3.23768" z="0" />
    </node>
    <node id="v7-RI8IaB-J">
        <location x="0.725475" y="2.43928" z="0" />
    </node>
    <node id="v8-RI8IaB-K">
        <location x="1.92218" y="0.137881" z="0" />
    </node>
    <node id="v9-RI8IaB-L">
        <location x="0.818847" y="2.68828" z="0" />
    </node>
</snapshot>
</scenario>
```

ΠΑΡΑΡΤΗΜΑ Ε – makefile

```
# ----------------------------------------
# Environment variable WISELIB_PATH needed
# ----------------------------------------

all: shawn
# all: scw_msb
# all: contiki_msb
# all: contiki_micaz
# all: isense
# all: tinyos-tossim
# all: tinyos-micaz

export APP_SRC=improved-rpl.cpp
export BIN_OUT=improved-rpl

include ../../applications/Makefile
```