Πανεπιστήμιο Θεσσαλίας

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων

Διπλωματική Εργασία

Δημιουργία Scatternet σε Δίκτυα Bluetooth με θεωρία κοινωνικών Δικτύων

(Scatternet Formation in Bluetooth Networks with Social Networks Theory)

Συγγραφεύς: **Γιομπλιάκης Βασίλειος**

1ος Επιβλέπων καθηγητής: *Κατσαρός Δημήτριος*

2ος Επιβλέπων καθηγητής: *Μποζάνης Παναγιώτης*

ΕΠΙΤΡΟΠΗ ΕΞΕΤΑΣΗΣ

1ος Εξεταστής – Επιβλέπων Καθηγητής

Δρ. Κατσαρός Δημήτριος
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων
Πανεπιστήμιο Θεσσαλίας

2ος Εξεταστής

Δρ. Μποζάνης Παναγιώτης
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων
Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες.

Ολοκληρώνοντας την παρούσα διπλωματική μου εργασία, θα ήθελα να αναγνωρίσω, την πραγματικά πολύτιμη βοήθεια που μου προσφέρθηκε, και να πω, ένα μεγάλο **«ευχαριστώ»** στον επιβλέποντα καθηγητή μου, Δρ. Κατσαρό Δημήτριο. Επίσης οφείλω ένα «ευχαριστώ» στον δεύτερο επιβλέποντα καθηγητή μου, Μποζάνη Παναγιώτη για την άψογη συνεργασία.

Ένα μεγάλο ευχαριστώ, οφείλω επίσης σε όλους τους ανθρώπους, συγγενής και φίλους, που με βοήθησαν υλικά και ηθικά, για την περάτωση αυτής της εργασίας.

Τέλος θα ήθελα να ευχαριστήσω, τους φίλους μου Μάμαλη Δημήτριο, Τριαντόπουλο Ευστάθιο, και Μπαράτη Νικόλαο, για τον πολύτιμο χρόνο τους που μου αφιέρωσαν, και για την βοήθεια που μου παρείχαν κατά την διάρκεια ολοκλήρωσης της διπλωματικής μου εργασίας.

Περιεχόμενα

1. Εισαγωγή

Η εισαγωγή των wireless Pas (WPANs) στην τεχνολογία, έχει φέρει επανάσταση στις ασύρματες τεχνολογίες. Τα WPANs είναι ικανά σε μια μικρή απόσταση (από κάποια εκατοστά μέχρι κάποια μέτρα) να μεταφέρουν και να ανταλλάξουν πληροφορίες με παρόμοιες συσκευές. Τα WPANs, μπορούν να χρησιμοποιηθούν μέχρι και για την αντικατάσταση των καλωδίων, από τα computers στα περιφερειακά τους, ή για να γίνεται μια σωστή διανομή πληροφοριών μεταξύ συσκευών, ή και για την κατασκευή ενός δικτύου αισθητήρων. Το καλύτερο παράδειγμα WPANs, είναι η τεχνολογία Bluetooth, που στις μέρες μας εμφανίζεται σε πολλές ηλεκτρονικές συσκευές όπως κινητά τηλέφωνα, PDAs, ασύρματα hands-free, ή και ασύρματα πληκτρολόγια.

1.1 Τεχνολογία Bluetooth (τι είναι το Bluetooth)

Το Bluetooth (ή συντομογραφικά BT), έχει γίνει μία από τις πιο σημαντικές μεθόδους μετάδοσης για τις εταιριες τηλεπικοινωνιών τα τελευταία χρόνια. Το Bluetooth είναι ένα χαμηλού κόστους, και μικρής ακτίνας μετάδοσης επικοινωνιακό μέσο. Εισάχθηκε σαν ιδέα στο Ericsson Laboratories, το 1994. Οι μηχανικοί έβλεπαν στην ανάγκη για μια ασύρματη τεχνολογία μετάδοσης, που θα ήταν φτηνή, δυνατή, ευέλικτη, και χαμηλού κόστους ενέργειας. Η βασική ιδέα αντικατάστασης των καλωδίων με περιθώρια εξέλιξης, έφερε πολύ γρήγορα, πάνω από 2500 εταιρίες αντιμέτωπες με το Bluetooth SIG. Το Bluetooth επιλέχτηκε, να εξυπηρετεί την κύρια «γραμμή» τις IEEE 802.15.1 standard για τα WPANs τα οποία μπορούσαν να υποστηρίξουν σύγχρονη επικοινωνία, όπως φωνή αλλά και ασύγχρονη, όπως μετάδοση δεδομένων.

1.2 Ιστορία και εφαρμογές

Στο περιβάλλον του WPANs, η τεχνολογία Bluetooth εισέρχεται τον Μάιο του 1998, και από τότε το Bluetooth SIG, οδήγησε στην επέκταση του Bluetooth, συμπεριλαμβάνοντας πρωτόκολλα και εφαρμογές. Το Bluetooth SIG είναι ένα group αποτελούμενο από «leaders» στον τομέα των τηλεπικοινωνιών και των υπολογιστών. Εταιρίες όπως η 3Com, Ericsson, IBM, Intel, Microsoft, Motorola, Nokia, Toshiba. Η ασύρματη τεχνολογία Bluetooth έχει κατοχυρωθεί σαν standard προδιαγραφή, για τις χαμηλού κόστους, και μικρής ακτίνας συνδέσεις, μεταξύ φορητών PCs, κινητών τηλεφώνων, και άλλων φορητών συσκευών. Σκοπός του Bluetooth είναι να παρέχει στους χρήστες του τη δυνατότητα σύνδεσης, αρκετών υπολογιστικών και τηλεπικοινωνιακών συσκευών, χωρίς τη χρήση καλωδίων. Επιτυγχάνει γρήγορη, ad-hoc σύνδεση, μεταξύ των συσκευών. Λόγω του οτι το Bluetooth μπορεί να χρησιμοποιηθεί σε πολλές εφαρμογές, ενδεχομένως, μπορεί να αντικαταστήσει και πολλές ενσύρματες ζεύξεις, μέσο ενός μικρού radio link. Οι εφαρμογές που εισήχθησαν με τα WPANs, είναι πολλές, και τώρα με το Bluetooth περισσότερες. Το Bluetooth SIG προτείνει πέντε εφαρμογές, που παρέχουν καλή εικόνα για τις ικανότητες του Bluetooth. Το three-in-one phone, μια Internet bridge, μία interactive conference, ένα hands-free, και ένα automatic synchronizer.

Το three-in one phone, είναι ένα τηλέφωνο, που μπορεί να λειτουργήσει ως φορητό τηλέφωνο σπιτιού, σαν κινητό τηλέφωνο, και σαν walkie-talkie με άλλη συσκευή Bluetooth που βρίσκεται στην εμβελειά της. Το παράδειγμα του internet bridge, επιτρέπει σε έναν φορητό-κινητό υπολογιστή, την σύνδεσή του, με μία άλλη συσκευή που βρίσκεται στο internet, μέσο του Bluetooth. Το παράδειγμα του interactive conference, επιτρέπει, την ανταλλαγή δεδομένων, μεταξύ πολλών υπολογιστών, στη διάρκεια μια υγιής «σύσκεψης» των συσκευών αυτών. Το Bluetooth hands-free, μπορεί να συνδεθεί με οποιαδήποτε άλλη συσκευή Bluetooth, και να παράγει, φωνή και ήχο. Έτσι είναι σαν να λειτουργεί, σαν ασύρματο τηλέφωνο, σαν κινητό και σαν music player. Η automatic synchronizer, είναι μια εφαρμογή που επιτρέπει συσκευές όπως, desktop computers, laptops, PDAs, και κινητά τηλέφωνα , να συγχρονίζονται μεταξύ τους και να μπορούν να ανταλλάσουν πληροφορίες, και να προσαρμόζονται, παρόλο που είναι διαφορετικές συσκευές. Παρακάτω παρουσιάζουμε κάποιες άλλες εφαρμογές, που μπορούν τα Bluetooth δίκτυα να βρουν χώρα

- Ασύρματα περιφερειακά για home PC, μικρής ακτίνας.
- Game controllers
- Επαγγελματικά σημειωματάρια, PDAs, cell phones
- Σπορ, όπως αισθητήρες παλμών
- Στρατός

Το Bluetooth παρέχει μεγάλη δυνατότητα στη μετακίνηση και στο συγχρονισμό πληροφοριών σε μια τοπική ζεύξη. Οι δυνατότητες των Bluetooth εφαρμογών είναι τεράστιες, διότι, η επικοινωνία είναι πολύ μεγαλύτερη και συχνότερη, με ανθρώπους που βρίσκονται κοντά μας σε σύγκριση με ανθρωπους που είναι πιο μακριά. Είναι ένα φυσικό φαινόμενο στην ανθρώπινη κοινωνία.

1.3 Τεχνική επισκόπηση του Bluetooth

Τα χαρακτηριστικά του Bluetooth, περιγράφουν radio-συσκευές, που έχουν σχεδιαστεί ώστε να δουλεύουν σε μια μικρή σχετικά ακτίνα μετάδοσης, περίπου δέκα(10) μέτρα, ή και περιστασιακά και στα εκατό(100) μέτρα. Η μετάδοση γίνεται πάνω σε ένα radio-link που μεταφέρει ήχο ή δεδομένα, με μια μέγιστη χωρητικότητα ανά link-κανάλι 720kbps. Ο πραγματικός σκοπός αυτων των links, ήταν να αντικαταστήσουν τα καλώδια, μεταξύ των συσκευών. Στόχος ήταν τα «χαρακτηριστικά» να περιέγραφαν μία συσκευή που είναι, απλή, αντοχής, ανθεκτική, καταναλώνει μικρή ενέργεια, και δόθηκε έμφαση, στο να είναι μια συσκευή αρκετά φτηνή στην παραγωγή της.

Η ραδιοσυχνότητα που λειτουργεί, ανήκει στην ISM band στα 2.4 έως 2.48 GHz, χρησιμοποιώντας ένα φάσμα συχνοτήτων (frequency hopping spread spectrum, FHSS), μέχρι και 1600 hops/seconds. Το σήμα «αναπηδά» (κάνει hop), ανάμεσα από 79 συχνότητες με διαστήματα στο 1 MHz, προσπαθώντας έτσι να εξαλείψει τις παρεμβολές από άλλες συσκευές. Αυτό είναι πολύ σημαντικό αν σκεφτούμε ότι τις συχνότητες αυτές τις μοιράζονται πολλές ηλεκτρονικές συσκευές. Παρ όλα αυτα, ακόμα και αυτά τα μέτρα δεν είναι ικανά να κρατήσουν το Bluetooth ανεπηρέαστο από εξωτερικές παρεμβολές, και από παρεμβολές που δημιουργούνται από τις δικές του συσκευές.

Στο Bluetooth, το RF output είναι 0dBm (1 mW) σε ακτίνα 10 μέτρων, και -30dBm μέχρι +20dBm (100 mW) σε μεγαλύτερη ακτίνα μετάδοσης. Τα χαρακτηριστικά του Bluetooth χωρίζονται σε δύο μέρη:

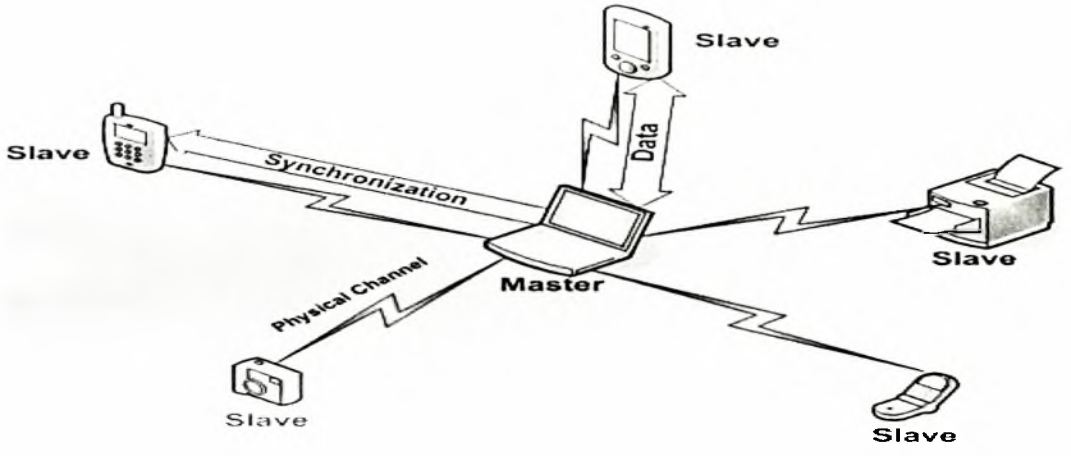
- πυρήνας (Core). Το τμήμα αυτό καθορίζει βασικές λειτουργίες, όπως είναι η μετάδοση, το base band, το link-manager, service discovery protocol, επίπεδο μεταφοράς(transport layer), επικοινωνιακή λειτουργία με διαφορετικά επικοινωνιακά πρωτόκολλα.
- Το προφίλ (Profile). Το τμήμα του προφίλ, χαρακτηρίζει τα πρωτόκολλα και τις διαδικασίες που απαιτούνται για διαφορετικού είδους Bluetooth εφαρμογών.

Τα χαρακτηριστικά του Bluetooth, καλύπτουν χαρακτηριστικά από το φυσικό και πληροφοριακό επίπεδο από το link επικοινωνίας. Πρέπει να σημειωθεί ότι ο στενός διαχωρισμός ανάμεσα στα διαφορετικά επίπεδα του τυπικού πρωτοκόλλου, που ορίστηκε από το Open System Interconnection (OSI) model, χάνει την σπουδαιότητά του στις ασύρματες υλοποιήσεις.

1.4 Ad-Hoc Δίκτυα

Όταν ενα ζεύγος ή μικρή ομάδα Bluetooth συσκευών, έρχονται σε ακτίνα μετάδοσης μεταξύ τους, μπορούν να δημιουργήσουν ένα ad-hoc δίκτυο χωρίς την απαίτηση κάποιας υποδομής. Οι συσκευές, προστίθενται ή αφαιρούνται από το δίκτυο δυναμικά. Έτσι μπορούν να συνδεθούν ή και να αποσυνδεθούν από ένα δίκτυο στο μέλλον, χωρίς να διακόψουν τη λειτουργία των άλλων συσκευών. Στο Bluetooth, η συσκευή που αναλαμβάνει την πρωτοβουλία να αρχίσει την επικοινωνία με κάποια άλλη συσκευή, αναλαμβάνει το ρόλο του master, και η άλλη συσκευή το ρόλο του slave.

- **Pico-net**



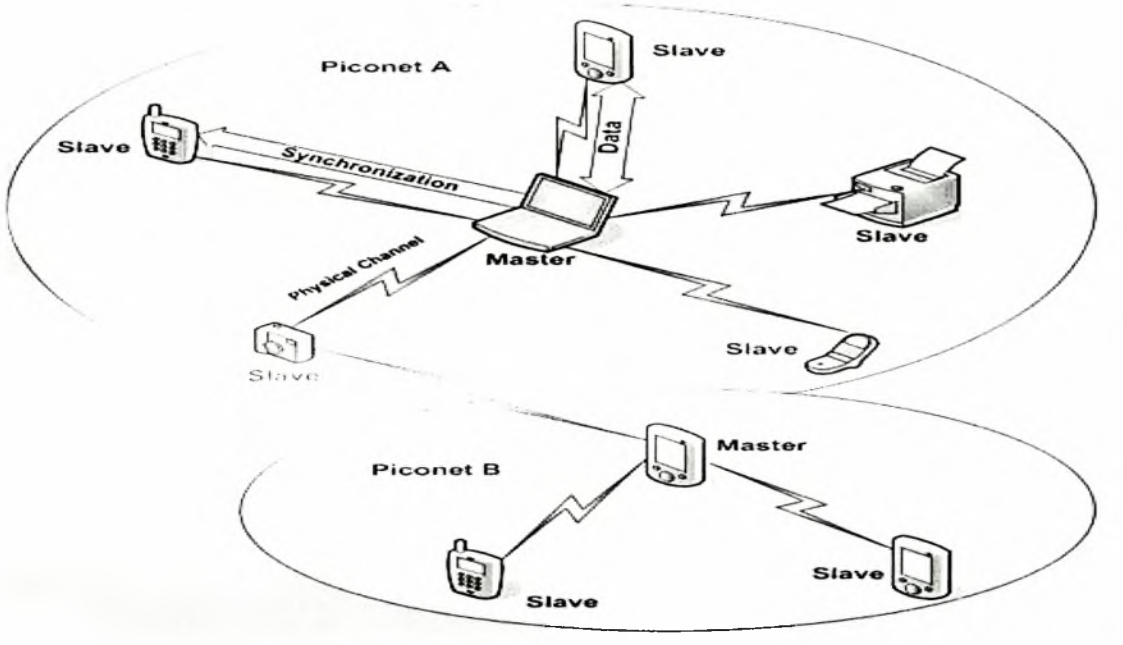
Η βασική αρχιτεκτονική μονάδα του Bluetooth είναι το Pico net. Αποτελούμενο από μία master συσκευή και μέχρι επτά, «ενεργούς», slaves κόμβους, οι οποίοι επικοινωνούν μεταξύ τους, αποκλείστηκα μέσο του master. Στην παραπάνω εικόνα δίνεται ένα παράδειγμα από ένα Pico net.

Όλοι οι κόμβοι σε ένα Pico net, μοιράζονται τις ίδιες συχνότητες επικοινωνίας χρησιμοποιώντας ένα slotted time division duplex σχήμα με ένα maximum bandwidth 1 Mbps. Το Pico net σε ένα Bluetooth δίκτυο, βασίζεται στη λειτουργία του master. Η master συσκευή, έχει υπό τον έλεγχό της ένα μικρό κανάλι, και όλες οι slaves συσκευές που ανήκουν στο Pico net λειτουργούν σε αυτό το κανάλι. Για να γίνει μια συσκευή master, αιτείται μια σύνδεση με μία άλλη συσκευή. Εάν η συσκευή δεχτεί τη πρόταση, η συσκευή που έκανε την πρόσκληση γίνεται master γι' αυτήν την ζεύξη, και η άλλη γίνεται slave. Από τη στιγμή που όλη η κίνηση στο piconet περνάει δια-μέσω του master, ο master έχει πλήρη έλεγχο της επικοινωνίας στο δικό του piconet. Σύμφωνα με το αυστηρό TDD σχήμα, μια slave συσκευή, έχει την άδεια να μεταδώσει σε ένα slot υπό τους ακόλουθους κανόνες:

- Όταν ο master κάνει ερώτηση στον slave εάν επιθυμεί να στείλει κάποιο μήνυμα.
- Όταν ο master στείλει ένα broadcast πακέτο σε ένα προηγούμενο slot.
- Όταν το slave έχει κάνει ήδη κράτηση για το συγκεκριμένο slot.

Εκτός των ενεργά slaves, μπορούν να συμμετάσχουν σε ένα piconet και σαν parked nodes, που απλα «ακουν» και δεν συμμετέχουν. Όταν επιθυμήσουν να εισέλθουν κανονικά, τότε κάποιο από τα ενεργά nodes αποχωρεί και τη θέση του παίρνει το καινούριο. Με αυτή τη μέθοδο, θεωρητικά μπορούν μέχρι 255 συσκευές να συνδέονται σε ένα piconet. Σε περίπτωση που το ενεργό master εγκαταλείψει το piconet τότε ένα από τα slaves αναλαμβάνει το ρόλο του. Στη διάρκεια του piconet formation, ο master δεσμεύει ένα "active member address" σε όλες τις συσκευές, και χρησιμοποιούνται αυτές η διευθύνσεις για την επικοινωνία πάνω στο piconet. Επίσης, κάθε Piconet χρησιμοποιεί ένα διαφορετικό Frequency Hopping Sequence, με σκοπό να μειωθούν οι παρεμβολές από άλλα piconet.

- Scatternet



Με σκοπό τη μείωση των κόμβων σε ένα δίκτυο, μια scatternet αρχιτεκτονική δικτύου, αποτελούμενη από μερικά piconet προτάθηκε και παρουσιάζεται ένα παράδειγμα στο παραπάνω σχήμα. Εμφανίζεται ένα scatternet αποτελούμενο από δύο piconets. Το scatternet μπορεί να

αποτελείτε από παραπάνω από ένα piconet, έτσι κάποιες συσκευές δρουν σαν γέφυρες επικοινωνίας μεταξύ των Piconets για τη μετάδοση των πακέτων. Στη περίπτωση αυτή τα piconet έχουν το δικό τους μοναδικό αναγνωριστικό (FHS). Μια συσκευή που είναι γέφυρα επικοινωνίας, είναι μέλος σε παραπάνω από ένα piconet, ωστόσο μπορεί να είναι ενεργή μόνο σε ένα από αυτά στιγμιαία. Μια συσκευή μπορεί να είναι slave σε διαφορετικά piconets αλλά μόνο σε ένα μπορεί να δρα ως master. Δεδομένου ότι τα διαφορετικά piconets έχουν διαφορετικά αναγνωριστικά (FHS) οι κόμβοι «γέφυρες» επιλέγουν την ταυτότητα του master στη διάρκεια τις επικοινωνίας μεταξύ των piconet, με σκοπό να συγχρονίσει την επικοινωνία μεταξύ των διαφορετικών piconets.

2. Επισκόπηση Τεχνικών Δημιουργίας Scatternet-Piconet

2.1 BlueNet

Ένα από τα πρωτόκολλα που υπάρχουν για το Bluetooth scatternet formation, είναι και το Bluenet. Το πρωτόκολλο αυτό, παράγει scatternets, αποτελούμενα από piconets με πεπερασμένο αριθμό k slaves.

Έπειτα από την ολοκλήρωση της φάσης, του εντοπισμού των κόμβων ενός δικτύου, κάθε κόμβος εισέρχεται σε μια κατάσταση, που είναι είτε page mode, είτε page scan mode, με πιθανότητα P0 (phase 0).

Όταν ένας κόμβος καταφέρνει με επιτυχία να δεσμεύσει ένα slave, προχωράει στη φάση 1, και προσπαθεί να αποκτήσει μέχρι k γειτονικούς κόμβους, ως slaves. Οι κόμβοι οι οποίοι, δεσμεύθηκαν ως slaves, επίσης προχωρούν στη φάση 1, μαζί με τον master τους. Οι slaves που βρίσκονται στη φάση 1, εξακολουθούν να παραμένουν σε page scan mode. Οι masters που πήγαν στη φάση 1, πρέπει να γνωρίζουν τους γειτονικούς τους masters, καθώς και τους masters των γειτονικών τους slaves, που ανήκουν σε άλλα piconets, καθώς και κόμβους που συνάντησαν στην φάση 0, όπου ήταν σε page scan mode, και τους ανακήρυξαν ως slaves. Για να περιγράψουμε αυτή τη διαδικασία, ένα master κάνει page, όλους τους γείτονές του, με ένα round-robin τρόπο, και έπειτα αποφασίζει, ποιο από αυτά θα πάει σε page scan mode και ποιο σε page mode, τυχαία με πιθανότητα P. Από την εναλλαγή, μεταξύ page mode και page scan mode, ο κάθε master μπορεί να συνδεθεί μέχρι k slaves μεταξύ των γειτόνων του, κι να αποκτήσει τις απαιτούμενες πληροφορίες, για τους υπόλοιπους γείτονες.

Όταν ένας κόμβος είναι ανεπιτυχώς στη διαδικασία να αποκτήσει έστω ένα slave, η έχει λάβει πρόσκληση από το piconet κάποιου άλλου κόμβου, τότε παραμένει στη εκτέλεση της φάσης 0, προσπαθώντας να αποφασίσει εάν θα είναι σε page mode ή σε page scan mode, με πιθανότητα P0. Αυτό συμβαίνει, μέχρι να πληροφορηθεί, πως όλοι οι γειτονικοί κόμβοι, έχουν γίνει μέλος κάποιου piconet.

Σε περίπτωση που ένας κόμβος στη φάση 0, μείνει μόνος του, τότε μεταβαίνει στη φάση 2. Π.χ. πηγαίνει σε page mode και προσπαθεί, να διασυνδεθεί με κάποιο από τα piconets, ζητώντας slaves από κάποια piconets (μέχρι k). Με την ολοκλήρωση αυτής της διαδικασίας, ένας κόμβος που είναι στη φάση 2, ολοκληρώνει την εκτέλεση του πρωτοκόλλου.

Ένα master στη φάση 1, αφού έχει επικοινωνήσει με όλους τους γειτονικούς του κόμβους, και έχει αποκτήσει μέχρι k slaves στο piconet του, πηγαίνει στη φάση 3, που είναι η φάση διασύνδεσης του piconet. Στη φάση αυτή, τα slaves από τα piconet, που επιλέχθεισαν στη φάση 1, (εναλλάσσοντας page mode και page scan mode), προσπαθούν να στήσουν ένα link μεταξύ των άλλων γειτονικών salves που επιλέχθεισαν στη φάση 1 σύμφωνα με την απόφαση των master τους. Οι masters επιλέγουν ένα από αυτά τα slaves να πάει σε page mode και δίνουν εντολή σε όλα τα άλλα να πάνε σε page scan mode. Κάθε τέτοιο slave, αρχίζει να κάνει page όλους τους γείτονές του,

με έναν round-robin τρόπο, όπου διαρκεί κάθε page περίπου χρόνο ίσο με Tpage. Για κάθε επιτυχημένο page, το slave ρωτάει το master του, αν είναι χρήσιμο να εγκαταστήσουν μια διασύνδεση με αυτόν τον κόμβο. Όταν έχουν ολοκληρωθεί όλες οι διαδικασίες paging των γειτόνων, τότε οι masters δίνουν εντολή στα slaves να πάνε σε page scan mode, και επιλέγουν κάποιο άλλο slave που θα πάει σε page mode. Όταν απομένει μόνο ένα slave, με τους δικούς του γειτονικούς slaves, τότε παίρνει εντολή, να εναλλαχτεί μεταξύ page mode και page scan mode, ώστε να εγκαταστήσει την απαιτούμενη διασύνδεση. Όλη η διαδικασία τερματίζει, μόλις ολοκληρωθούν οι απαιτούμενες διασυνδέσεις μεταξύ των διαφορετικών piconets. Η συνεκτικότητα στο παραγόμενο scatternet, δεν είναι εγγυημένη. Δεν συνδέονται όλα τα Bluenets, ακόμα και είναι συνδεδεμένα στη φάση ανακάλυψης των κόμβων. Ένα απλό παράδειγμα φαίνεται στο παρακάτω σχήμα.

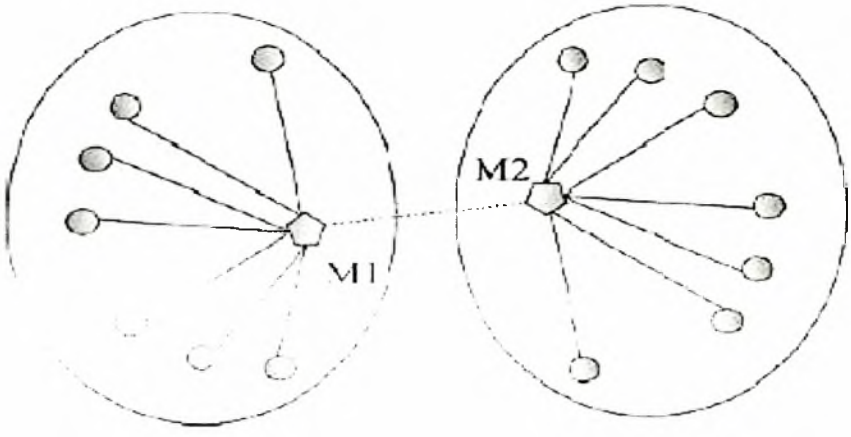
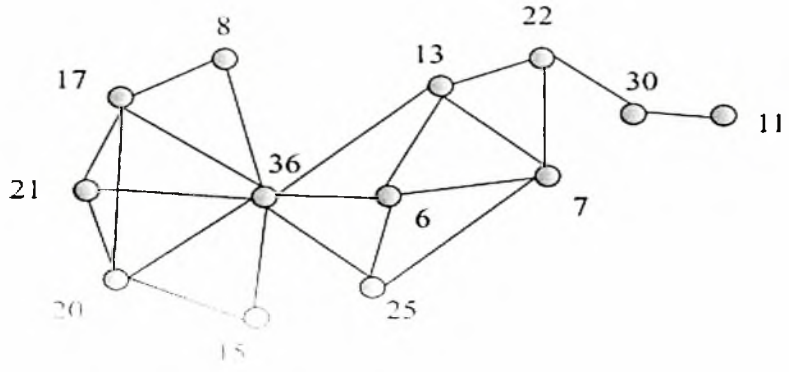


Figure 3. A disconnected BlueNet.

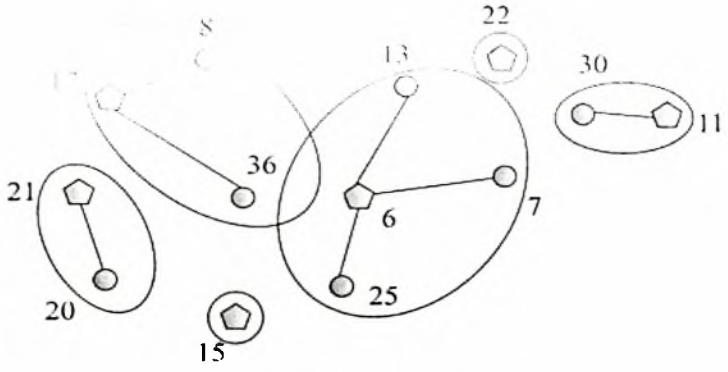
Υποθέτουμε ότι οι κόμβοι M1 και M2, ανακαλύφθηκαν μεταξύ τους, και είναι οι μόνοι δύο κόμβοι που πηγαίνουν σε page mode στη φάση 0. Υποθέτουμε, ότι επιτυχώς προσκαλούν τους 7 αριστερά και 7 δεξιά κόμβους αντίστοιχα, να γίνουν μέλη των piconet τους. Σ' αυτή τη φάση οι κόμβοι και από τα δύο piconet πηγαίνουν στη φάση 3. Παρόλα αυτά δεν υπάρχει τρόπος να διασυνδεθούν τα δύο piconet. Είτε μέσο κάποιας πύλης διασύνδεσης(κανένα από τα slaves του Piconet M1 δεν είναι γείτονας με κάποιο slave του M2, και το αντίθετο), είτε απευθείας μεταξύ τους, αφού κάθε ένα από αυτά έχει ήδη 7 slaves.

Σύμφωνα με το πρωτόκολλο, οι slaves θα πρέπει να αποτρέπονται από την εγκατάσταση μιας διασύνδεσης μέσα στο ίδιο τους το piconet. Επιπλέων διαφορετικές διασυνδέσεις, μεταξύ των ίδιων piconets πρέπει να αποφεύγονται. Οι απαιτήσεις σχεδίασης του scatternet επιτυγχάνονται καλύτερα, όταν ο master πληροφορεί τους slaves για την ταυτότητα του κάθε καινούριου slave κόμβου που συμμετάσχει στο piconet. Αυτό επιτυγχάνεται χρησιμοποιώντας LMP πακέτα, από το master στα slave του. Για τα Masters(slaves) για να επιτύχουν την μετάδοση(παραλαβή) αυτή, πρέπει τα slaves που ανήκουν σ' αυτό το piconet, να πάνε σε sniff mode. Σταματώντας έτσι περιοδικά τις δραστηριότητες του page και του page scan, ή τις ενέργειες διασύνδεσης, και στρέφονται στη συχνότητα μετάδοσης του piconet με σκοπό να «ακούσουν» τα μηνύματα που στελνει ο master. Για να αποφύγουμε διπλό-συνδεδεμένα piconets, ένα slave κάθε στιγμή, δημιουργεί ή γίνεται μέλος ενός προσωρινού Piconet που περιέχει ένα slave από διαφορετικό

Piconet. Έτσι επικοινωνεί μετά, με το master του, για να τσεκάρει αν υπάρχει ήδη αυτή η σύνδεση. Εάν ναι, το master πληροφορεί τους slave του, για την καινούρια διασύνδεση με το καινούριο piconet, στο πρώτο επερχόμενο sniff παράθυρο.



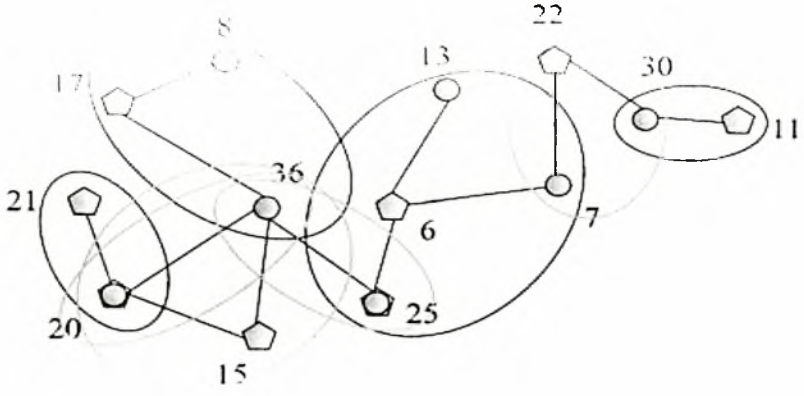
Το BlueNet, περιγράφεται με το ακόλουθο παράδειγμα. Ξεκινάμε με το δίκτυο που φαίνεται στην παραπάνω εικόνα, θεωρώντας ότι μόλις έχουμε ολοκληρώσει, τη διαδικασία εντοπισμού κόμβων. Θεωρούμε ότι ο αριθμός που αντιπροσωπεύει κάθε κόμβο, είναι το προσωπικό του ID. Στην αρχική φάση 0, οι κόμβοι 6, 11, 17, 21 και 22, τυχαία αποφασίζουν να κάνουν page τους γείτονές τους. Όλοι οι υπόλοιποι κόμβοι, πηγαίνουν σε page scan mode. Όταν ο κόμβος 16, αποκτήσει επιτυχώς τον κόμβο 13 σαν slave, τότε θα προχωρήσει στη φάση 1, μαζί με τον κόμβο 13. Το ίδιο συμβαίνει και με τους κόμβους 11 και 30 που προχωράνε στη φάση 1(ο κόμβος 11 είναι ο master και ο 30 ο slave του). Παρομοίως οι κόμβοι17 και 21, αποκτούν τους 36 και 20 σαν slaves αντίστοιχα. Και οι τέσσερις κόμβοι, πηγαίνουν στη φάση 1. Ο κόμβος 6 επεκτείνει το piconet του, αποκτώντας τους κόμβους 7 και 25 σαν slaves. Ο κόμβος 8, γίνεται μέλος του piconet 17. Με το που κάνει ο κόμβος 22 επιτυχώς page τους 7. 13 και 30, συνειδητοποιεί, ότι αυτοί οι κόμβοι συμμετέχουν ήδη σε κάποιο άλλο piconet. Όντας σαν μοναδικός master, ο κόμβος 22 προχωράει στη φάση 2. Ο κόμβος 15, περιμένει για κάποιο page από κάποιον γειτονικό κόμβο. Από τη στιγμή που δεν φτάνει κάποιο page, μετά από ένα χρονικό διάστημα αποφασίζει τυχαία, αν θα παραμείνει σε page scan mode ή αν θα μεταβεί σε page mode. Όταν κάποια στιγμή πάει σε page mode, και κάνει page τους γείτονες του, 20 και 36, θα καταλάβει ότι αυτοί ανήκουν σε κάποιο άλλο master, οπότε και θα προχωρήσει στη φάση 2. Στο παρακάτω σχήμα φαίνεται το «original piconet», που δημιουργήθηκε από τα Piconets, στις φάσεις 1 και 2.



Οι κόμβοι 15 και 22, που είναι στη φάση 2, κάνουν page όλους τους γείτονές τους, με σκοπό να συνδεθούν με κάποιο γειτονικό piconet. Ο κόμβος 22 για παράδειγμα κάνει page τους κόμβους 7, 13 και 30, μέχρι να επικοινωνήσει με όλους. Στην επικοινωνία με τον κόμβο 30, ο

κόμβος 22 ζητάει να επιτευχθεί μια σύνδεση μεταξύ του piconet 22 και 11. Ο κόμβος 30 ρωτάει τον master του, και ο κόμβος 11 αποδέχεται να γίνει ο 30, μια slave πύλη διασύνδεσης δύο piconets. Παρόμοια ο 22 κάνει την ίδια αίτηση στον 7, και ο 6 αποδέχεται τον slave 7 σαν πύλη διασύνδεσης με το άλλο Piconet. Όταν ο 22 κάνει το ίδιο με τον γείτονά του 13, ο 13 ζητάει από τον master του, τον 6, την άδεια να συμμετάσχει στο piconet του 22, αλλά σε αυτήν την περίπτωση ο 6 δεν συμφωνεί με το να μοιραστεί τον κόμβο 13, γιατί έχει ήδη μια άλλη διασύνδεση με τον Piconet αυτό. Παρόμοια ο κόμβος 15, επεκτείνει τον piconet του προσθέτοντας τους κόμβους 20 και 36, εγκαθιστώντας έτσι και διασύνδεση με το Piconet 21 και 17. Ο κόμβος 15 και 22 τερματίζουν την εκτέλεση του πρωτοκόλλου.

Αφού έχουν επικοινωνήσει με όλους τους γείτονές τους, οι masters στη φάση 1 πηγαίνουν στη φάση 3 μαζί με τα slave τους. Στη φάση αυτή, γειτονικά piconets είναι συνδεδεμένα με πύλες διασύνδεσης. Ο κόμβος 21 οδηγεί τον μοναδικό του slave, σε εναλλαγές page και page scan mode, με σκοπό να επικοινωνήσει με όλα τα slaves των γειτονικών piconets. Οι κόμβοι 36 και 20 εγκαθιστούν τελικά ένα Link διασύνδεσης, ομοίως και οι 17 με 21. Ο κόμβος 20, γίνεται ο master, από ένα extra piconet που δημιουργήθηκε, συμπεριλαμβάνοντας τον κόμβο 36 σαν slave. Το piconet 25, το οποίο επίσης συμπεριλαμβάνει τον 36 σαν slave, διασυνδέεται με τα piconet 6 και 17. Ο κόμβος 13, λαμβάνει μήνυμα, ότι ο 36 συνδέθηκε με το piconet του 6 μέσο του 25. Έτσι δεν επικοινωνεί με τον κόμβο 36. Το αποτέλεσμα του BlueNet περιγράφεται στην παρακάτω εικόνα. Με αχνές γραμμές, είναι τα extra piconets που δημιουργήθηκαν.



2.2 Bluemesh

Στην ενότητα αυτή περιγράφουμε το BlueMesh, ένα πρωτόκολλο για Scatternet formation, όπου στα δίκτυα αυτά έχουμε ένα πεπερασμένο αριθμό slaves. Η συνεκτικότητα και η σωστή διασύνδεση των master του δικτύου, επιτυγχάνεται με την εγκατάσταση ενός interpiconet μεταξύ οποιονδήποτε δύο master που βρίσκονται το πολύ 3-hop μακριά. Το BlueMesh δεν χρειάζεται πληροφορίες που αφορούν την θέση των κόμβων.

Το BlueMesh προχωράει σε επιτυχημένες επαναλήψεις. Κάθε επανάληψη εκτελείται από τον κόμβο του δικτύου, που δεν έχει τερματίσει ακόμα, την εκτέλεση του πρωτοκόλλου σε κάποια

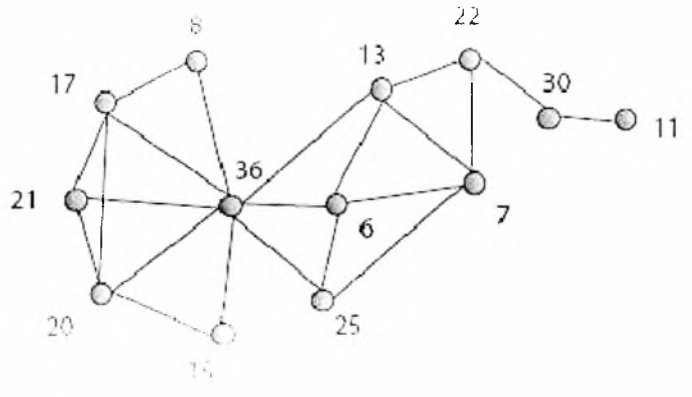
προηγούμενη επανάληψη. Ας ονομάσουμε $Gi = (Vi, Ei)$ την τοπολογία του δικτύου, στην επανάληψη $i, i > 1$. G1 είναι η τοπολογία του δικτύου μετά την φάση ανακάλυψης κόμβων στον γραφο. Κάθε ένα από τα $Gi, i > 1$, είναι υπο-γράφος του G1 που εκτείνει τον κόμβο V1 που δεν είχε τελειώσει την εκτέλεσή του Bluemesh σε κάποια προηγούμενη επανάληψη. Σε κάθε επανάληψη i, σχηματίζονται piconets από τους κόμβους του τοπικού γράφου Gi. Η υπερσύνδεση δύο piconet, επιτυγχάνεται είτε με την χρήση μιας πύλης slave, σαν πύλη διασύνδεσης, είτε με την χρήση ενός ζεύγους slaves σαν πύλη διασύνδεσης, όπου το ένα ανήκει στο ένα Piconet και το άλλο, στο άλλο piconet. Τα slaves , τα οποία θα λειτουργήσουν σαν πύλες διασύνδεσης, επιλέγονται στην εκάστοτε επανάληψη, ώστε να είναι σίγουρο ότι τα piconets θα συνδέονται. Τα masters με τη σειρά τους τότε, προχωρούν στο να επιλέξουν τις πύλες διασύνδεσης, για τα piconet που δεν έχουν ακόμα συνδεθεί.

Οι ενδιάμεσοι κόμβοι, είναι και οι κόμβοι που θα «προχωρήσουν» και στην επόμενη επαναληψη. Όλοι οι masters, οι slaves που δεν εξελίχθησαν σε ενδιάμεσες πύλες διασύνδεσης, καθώς και οι πύλες που ήταν slaves, στο σημείο αυτό, ολοκληρώνουν την εκτέλεση του πρωτοκόλλου BlueMesh. Το BlueMesh τερματίζει μόλις ολοκληρώσουν την εκτέλεση του πρωτοκόλλου, όλοι οι κόμβοι.

Η συνάρτηση του BlueMesh περιγράφεται στο παρακάτω παράδειγμα. Υποθέτουμε ότι κάθε BT συσκευή, γνωρίζει την ταυτότητά της, το βάρος της, καθώς και το βάρος και την ταυτότητα κάθε 1-hop ή 2-hop γείτονά της. Οι πληροφορίες για τους 2-hop γείτονες επιτυγχάνονται με την ανακάλυψη κόμβων, εκτελώντας το πρωτόκολλο «packing order» όπου οι πληροφορίες ανταλλάσσονται, μέσο των προσωρινών piconets, ανάμεσα από τους κόμβους v και u που είναι οι λιστες των γειτόνων τους, N(v) και N(u).

Κάθε επανάληψη του πρωτοκόλλου παρουσιάζεται τοπικά σε κάθε κόμβο v, και αποτελείται από δύο μέρη: role selection(για piconet formation) και getaway selection.

Η επιλογή ρόλου(role selection) εκτελείται από κάθε κόμβο στις αρχές κάθε επανάληψης i (στη περίπτωση της πρώτης επανάληψη, η επιλογή του ρόλου, παρουσιάζεται αφού έχει ολοκληρωθεί η διαδικασία ανακάλυψης της 2-hop γειτονιάς). Βασιζόμενο στο βάρος του και στο βάρος των 1-hop γειτόνων του, έναν κόμβος αποφασίζει αν θα είναι init στο Gi. Μόνο οι init κόμβοι μεταβαίνουν σε page mode. Όλοι οι υπόλοιποι πηγαίνουν σε page scan mode.



Ας υποθέσουμε πως έχουμε το δίκτυο της παραπάνω εικόνας. Ξεκινώντας με τους κόμβους που έχουν το μεγαλύτερο βάρος στη γειτονία τους, οι κόμβοι 30 και 36 είναι οι init κόμβοι. Γίνονται masters και πηγαίνουν σε page mode. Όλοι οι υπόλοιποι κόμβοι πηγαίνουν σε page scan mode. Ο κόμβος 30 που έχει λιγότερους από εφτά γείτονες, τους επιλέγει όλους (τους κόμβους 22 και 11) σαν slave του και τους κάνει page περιμένοντας την απάντηση τους. Έτσι το piconet 30, αποτελείται από τους κόμβους, 30, 22, 11. Ο κόμβος 36 όμως, έχει οκτώ γείτονες, και γνωρίζουμε ότι δεν μπορούμε να έχουμε πάνω από εφτά σε ένα master. Έτσι ο 36 θα πρέπει να επιλέξει εφτά κόμβους για slaves από τους οκτώ. Η επιλογή των slave σε έναν master κόμβο v, παρουσιάζεται στη εκτέλεση της παρακάτω διαδικασίας, όπου S(v) είναι το σύνολο των επιλεγμένων γειτόνων, και C(v) οι μικρότεροι γείτονες του v, και οι μεγαλύτεροι γείτονες του v που είναι slaves.

```
COMPUTES(v)
1  S(v) _ 0/
2  U _ C(v)
3  while U _ 0/
4     do x_bigger in U(v)
5        S(v) _S(v) _ {x}
6        U_U \ N(x)
7  S(v) _S(v) _ GET(7 – |S(v)|, C(v) \ S(v))
```

Κάθε master v, επιλέγει σαν slaves τους γείτονες του C(v) όπου «καλύπτουν» όλους τους άλλους γείτονες στη πρόταση, εάν ένας κόμβος u δεν επιλέχθηκε από τον v σαν slave, τουλάχιστο ένας από τους γείτονες του u θα επιλέχθηκε από τον v. Για να είμαστε καλυμμένοι, επιλέγουμε συνήθως μέχρι πέντε slaves.

Με τη διαδικασια ComputeS επιτυγχάνουμε πολύ καλή προσέγγιση του S(v). Ένας από τους γειτονες του v στο C(v) (π.χ. αυτός με το μεγαλύτερο βάρος) επιλέγεται σαν slave, έστω ο κόμβος χ. Η διαδικασία έπειτα εκτελείται σε όλους τους κόμβους C(v) \{x}, που δεν καλύπτονται από τον χ. ο κανόνας αυτός επιτρέπει στον v να επιλέξει μέχρι πέντε από τους γείτονές του στο C(v), όπου μπορούν να παρατηρηθούν όλοι οι γείτονες.

Η συνάρτηση GET(m, W), επιστρέφει ένα πακέτο κόμβων m από ένα πακέτο W (για παράδειγμα τους m μικρότερους). Χρησιμοποιείται από την COMPUTES, για την επιλογή των slaves που δεν υπερβαίνουν τους εφτά σε αριθμό.

Εκτελώντας έτσι το COMPUTES(36) , ο κόμβος 36, επιλέγει αρχικά πέντε κόμβους, τους 8, 13, 15, 21 και 25. Μέσω αυτών, ο 36 μπορεί να δει και τους υπόλοιπους γείτονες του. Έπειτα επιλέγοντας τους κόμβους 17 και 20, φτάνει στο όριο των εφτά slaves κόμβων (διαδικασία GET, line 7). Στο σημείο αυτό ο 36, κάνει page όλους τους γείτονές του, και πληροφορεί τον 6 ότι δεν θα συμμετέχει στο piconet του, και προσκαλεί όλους τους υπόλοιπους κόμβους. Όταν ένας κόμβος δέχεται πρόταση να συμμετάσχει σε ένα Piconet την δέχεται ακόμα και αν είναι μέλος άλλου Piconet. Έτσι το piconet 36, αποτελείται από οκτώ κόμβους. Τον master 36 και τους slaves 8, 13, 15, 17, 20, 21 και 25. Στο Piconet 30 ο slave 22 έχει δεχτεί page από όλους τους μεγαλύτερους γείτονές του. Έτσι, πηγαίνει σε page mode, και αρχίζει να κάνει page έναν-έναν τους γείτονές του από το C(22), συγκεκριμένα τις συσκευές 7 και 13, γνωστοποιώντας μάλιστα και τον «ρόλο» του. Ομοίως η συσκευή 25 που έχει δεχτεί page από την 36, κάνει page την 6 και τη 7, που ανήκουν στο C(25). Η συσκευή 13 έχει λαβει page από όλους τους μεγαλύτερούς της γείτονες, και είναι τώρα έτοιμη, να τους γνωστοποιησει το ρολο της(slave του master 36), στους μικρότερους της γείτονες 6 και 7. Η

συσκευή 7 που ξέρει πλέων πως όλοι οι μεγαλύτεροι της γείτονες(κόμβοι 13, 22, 25) είναι slave, αποφασίζει και γίνεται master, κάνοντας page και τους τέσσερις γείτονές της, όπου και τους προσκαλεί στο Piconet του, και αυτοί δέχονται. Έτσι το piconet 7, αποτελείται από τον master του και τους slave του 6, 13, 22, 25. Αυτήν τη στιγμή το δίκτυο έχει χωριστεί σε τρία piconet, όπου πρέπει να διασυνδεθούν. Αυτό σηματοδοτεί, την έναρξη της φάσης επιλογής πύλης διασύνδεσης, κάθε κομματιού ξεχωριστά. Στη φάση αυτή, όλοι οι slaves, επικοινωνούν με τους master τους, και πληροφορούν για τον ρόλο των γειτόνων τους, τους master των γειτόνων τους, και αν τους επιλεξε κάποιον γειτονικός master σαν slave. Η πληροφορία αυτή, διαχέεται στο δίκτυο σαν ένα κύμα, στη φάση που γίνεται η επιλογή των ρόλων κάθε κόμβου. Με βάση αυτές τις πληροφορίες, κάθε master αποφασίζει πιο slave θα ορίσει σαν κόμβο-πύλη διασύνδεσης δυο Piconets, με σκοπό τη δημιουργία ενός scatternet. Σε περίπτωση που δύο master έχουν επιλέξει ένα κοινό ζευγάρι από slaves, τότε επιλέγεται το μεγαλύτερο, και γίνεται η πύλη διασύνδεσης δύο piconets. Αυτός είναι και ο πιο ενδεδειγμένος τρόπος διασύνδεσης δύο piconet. Όταν δεν μπορέσει να επιλεγεί πύλη διασύνδεσης, τότε, ξαναγίνεται η επιλογή με βάση το βάρος του κάθε κόμβου (π.χ. έτσι ώστε το άθροισμά του βάρους τους να μεγιστοποιείτε, ή το ελάχιστο του βάρους να μεγιστοποιείτε, ή κάποιος άλλος κανόνας). Με το πέρας αυτής της φάσης, τα slaves διασύνδεσης, μαζί με τα master και τα slaves μη-διασύνδεσης, ολοκληρώνουν την εκτέλεση του πρωτοκόλλου BlueMesh. Οι πύλες διασύνδεσης, προβαινουν σε i+1 επαναλήψεις με σκοπό τη δημιουργία καινούριων piconets που τα διασύνδεσανε, και από δω και πέρα, παρέχουν διασύνδεση μεταξύ των piconets που διασύνδεσανε στην επανάληψη i. Πηγαίνοντας πίσω στο παράδειγμά μας, οι κόμβοι 22 και 25 είναι κοινοί slaves από τα piconets 7 και 30, και 7 και 36, έτσι επιλέγονται σαν πύλες διασύνδεσης τέτοιων piconets. Τα piconets 30 και 36, μπορούν να διασυνδεθούν μόνο μέσο των πυλών 13 και 22. Αυτοί οι δύο κόμβοι, είναι οι μοναδικοί κόμβοι, που θα προχωρήσουν στην επόμενη επανάληψη του πρωτοκόλλου. Όλοι οι υπόλοιποι, τερματίζουν την εκτέλεση του BlueMesh, αυτήν την στιγμή. Κάποιοι από αυτούς τους κόμβους είναι masters (σε ένα μικρό piconet), κάποιοι είναι slaves σε κάποιο Piconet(όπως είναι ο 6), και κάποιοι έχουν διάφορους ρόλους (όπως είναι οι πύλες διασύνδεσης 13, 22, 25). Ο κόμβος 22 τώρα από ξέρει ότι είναι init, πηγαίνει σε page mode, και [illegible] τον γειτονικό του κόμβο 13. Έτσι το piconet (δύο κόμβων), 22 σχηματίζεται, παρέχοντας τη διασύνδεση μεταξύ του master 30 και 36. Το αποτέλεσμα του scatternet που δημιουργείται, μετά την εκτέλεση του BlueMesh, παρουσιάζεται στη παρακάτω εικόνα. Τα master εμφανίζονται σαν πεντάγωνα, τα piconets που δημιουργήθηκαν στην πρώτη επανάληψη, σαν άστρα, κα τα piconets που δημιουργήθηκαν στη δεύτερη επανάληψη, σαν οβάλ σχήματα.

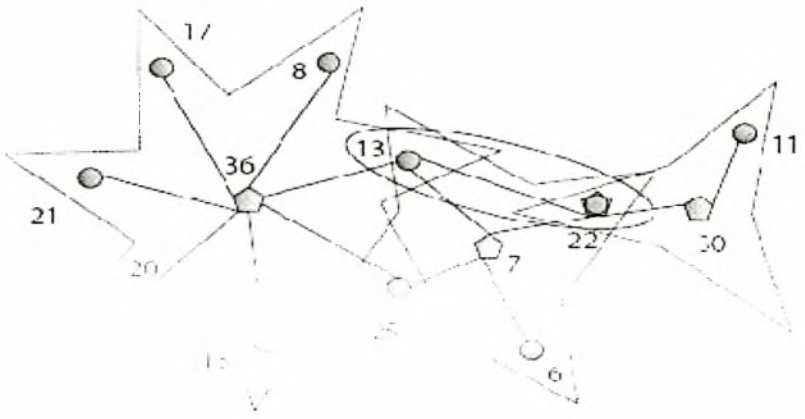


Figure 4.5. A BlueMesh scatternet.

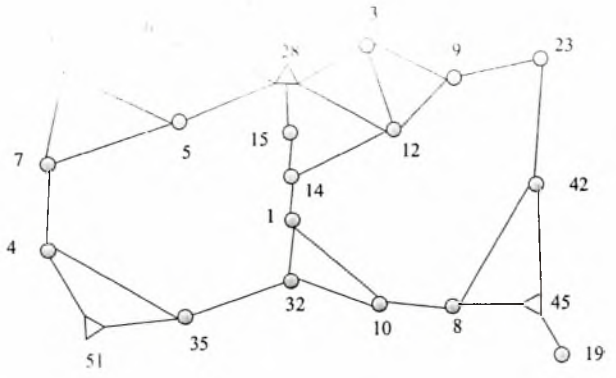
2.3 Bluestar

Για τη δημιουργία ενός piconet απαιτούνται δύο φάσεις. Η πρώτη φάση έχει να κάνει με την ανακάλυψη των γειτονικών κόμβων. Πρέπει να ανταλλαχτεί κάποια πληροφορία ανάμεσα σε ένα υποψήφιο master και σε ένα υποψήφιο slave ώστε να δημιουργηθεί ένα σύνδεσμος μεταξύ αυτών των δύο κόμβων.

Εδώ θα εισάγουμε τις έννοιες *inquiry mode* και *inquiry scan mode.* Για τη διαδικασία της ανακάλυψης των γειτονικών κόμβων, απαιτείται οι δύο συσκευές να βρίσκονται σε αντίθετες καταστάσεις. Συγκεκριμένα η συσκευή που κάνει την αναζήτηση πρέπει να είναι σε inquiry mode, και η συσκευή που πρόκειται να ανακαλυφθεί, σε inquiry scan mode. Ο inquirer μεταδίδει inquiry ID πακέτα στους γειτονικούς κόμβους, ζητώντας τους να τα επιβεβαιώσουν με τη σειρά τους, ώστε να υπάρξει ένας σύνδεσμος επικοινωνίας. Για να είναι ταχεία η ανακάλυψη των κόμβων, τα ID πακέτα που στέλνονται θα είναι όσο το δυνατόν μικρότερου μεγέθους. Όταν ένας κόμβος, που βρίσκεται σε inquiry scan mode, παραλάβει ένα ID πακέτο, υπολογίζει έναν χρόνο επιστροφής. Μόλις περάσει αυτός ο χρόνος, εάν η συσκευή σε inquiry scan mode λάβει ένα ID πακέτο τότε θα απαντήσει με ένα FHS (Frequency Hop Synchronization) πακέτο που συμπεριλαμβάνει το αναγνωριστικό της και πληροφορίες συγχρονισμού, όπως το Bluetooth ρολόι της. Η διαδικασία αυτή οδηγεί σε μια ασύγχρονη «αναγνώριση» των γειτόνων, διότι η συσκευή που έλαβε το ID πακέτο δεν γνωρίζει πληροφορίες για τον inquirer. Έπειτα από μια επιτυχημένη απάντηση της συσκευής που είναι σε inquiry scan mode, επέρχεται μια «χειραψία» μεταξύ των δύο συσκευών ώστε πλέων να είναι γνωστά τα και των δύο τα στοιχεία. Έπειτα ο inquirer προσκαλεί τον άλλο κόμβο να γίνει μέλος του δικού του piconet.

Η πρόσκληση αυτή γίνεται μέσο μιας διαδικασίας *paging*. Για να συνάψουν ένα link επικοινωνίας οι δύο κόμβοι, πρέπει να είναι σε αντίθετα mode, ο ένας σε page scan mode και ο άλλος σε mage mode. Εξ ορισμού master είναι αυτός που είναι σε page mode. Στην ερευνά μας δεν θα μελετήσουμε περεταίρω τη διαδικασία του paging.

Σκοπός της δικιάς μας έρευνας, είναι η εύρεση ενός αλγορίθμου, ώστε να έχουμε τη βέλτιστη επιλογή μεταξύ master και slaves σε ένα σύνολο από Bluetooth κόμβους. Σε μια γεωγραφική περιοχή, μπορεί να υπάρχει ένας μεγάλος αριθμός κόμβων. Όπως προαναφέραμε για να σχηματιστεί ένα δίκτυο από αυτούς τους κόμβους, πρέπει να σεβαστούμε κάποιους περιορισμούς. Ένας από αυτούς είναι, ότι κάθε κόμβος που είναι master δεν μπορεί να έχει πάνω από εφτά slaves. Γνωρίζοντας ότι το δίκτυο που θα σχηματιστεί πρέπει να είναι συνεκτικό, (δηλαδή να μην υπάρχει κόμβος του δικτύου που είναι αποκομμένος από τους υπόλοιπους), καταλαβαίνουμε ότι η επιλογή των master και των slaves, είναι καθοριστική για τη λειτουργικότητα του δικτύου.



Έχοντας έτοιμο λοιπόν το δίκτυο με τους Bluetooth κόμβους, όπως παρουσιάζεται στην παραπάνω εικόνα, θα αρχίσουμε να μελετάμε την μέθοδο (Scatternet formation 2: Bluestar formation) επιλογής master και slave στο δίκτυο μας.

Στη μέθοδο αυτή, περιγράφεται ένα κατανεμημένο πρωτόκολλο κατηγοριοποίησης-ομαδοποίησης των Bluetooth κόμβων σε piconets. Το ζητούμενο είναι, κάθε piconet να αποτελείται από ένα master και έναν περιορισμένο αριθμό από slaves. Έτσι κάθε piconet μοιάζει σαν ένας αστέρας (η τοπολογία του). Γι' αυτό το λόγο η φάση αυτή, του πρωτοκόλλου, ονομάζεται «bluestar formation face».

Βασισμένοι στις πληροφορίες της γεννήτριας τυχαίων γραφημάτων, κάθε κόμβος έχει ένα μοναδικό ID, και ένα «βάρος». Για την ανταλλαγή των πληροφοριών συγχρονισμού το πρωτόκολλο θα εφαρμοστεί σε κάθε κόμβο τοπικά. Ο κανόνας που ακολουθείτε από κάθε συσκευή ξεχωριστά, είναι ο εξής: Η απόφαση μιας συσκευής ***v***, να είναι master ή slave εξαρτάται από την απόφαση που πάρθηκε από γειτονικές συσκευές με μεγαλύτερο «βάρος»(μεγαλύτερο βάρος της ***v***). Συγκεκριμένα η ***v*** γίνεται slave του πρώτου γειτονικού master (που έχει και μεγαλύτερο βάρος), που θα εκτελέσει τη λειτουργία του paging πάνω της, δηλαδή του πρώτου master που θα την προσκαλέσει να γίνει μέλος του δικού του piconet. Σε περίπτωση που κανένας μεγαλύτερος γείτονας δεν προσκαλέσει την ***v*** στο δικό του piconet, τότε η ***v*** ανακηρύσσεται από μόνη της ως master.

Με το που αποφασίσει μια συσκευή το ρόλο της (master ή slave), τον ανακοινώνει στους αμέσως μικρότερους γείτονές της, ώστε να μπορέσουν κι αυτοί να πάρουν τη δική τους απόφαση, για το ρόλο που θα έχουν.

Αρχικά ονομάζουμε τους κόμβους με τα μεγαλύτερα βάρη στη γειτονία τους ως *init*. Δεδομένου ότι τα βάρη είναι αριθμοί μεγαλύτεροι του μηδενός, σίγουρα θα υπάρχει ένας init κόμβος στο δίκτυό μας. Οι init κόμβοι, είναι οι συσκευές που ξεκινούν τη διαδικασία του bluestar formation. Οι κόμβοι αυτοί, είναι και αυτοί που εξαρχής παίρνουν το ρόλο του

master και πηγαίνουν σε page mode. Οι υπόλοιποι κόμβοι βρίσκονται και κατάσταση page scan mode αναμένοντας τη διαδικασία ανακάλυψης κόμβων.

Στο παραπάνω σχήμα, οι init κόμβοι εμφανίζονται σαν τρίγωνα, ενώ οι υπόλοιποι κύκλοι. Η ακμή ανάμεσα σε δύο κόμβους συμβολίζει ότι, οι κόμβοι αυτοί βρίσκονται σε εμβέλεια μετάδοσης. Σ' αυτή τη φάση, μια συσκευή, μένει σε page scan mode, μέχρι να λάβει ένα page από όλους τους μεγαλύτερους γείτονές της. Εάν έστω και ένας μεγαλύτερος γείτονας είναι master τότε η συσκευή γίνεται μέλος από το piconet του πρώτου master που της έκανε page. Σε διαφορετική περίπτωση, η συσκευή αυτή αυτοανακηρύσσεται ως master. Τέλος, μετά από τα προηγούμενα βήματα, η συσκευή πηγαίνει σε page scan mode.

Ο παρακάτω ψευδοκώδικας, εκτελείτε από κάθε συσκευή αρχικά, για να το διαχωρισμό των ρόλων κάθε κόμβου.

```
INITOPERATIONS(v)
1 if (INIT(v))
2    then PAGEMODE
3         for each smaller u
4           do PAGE(u, v, 'master', v)
5              EXIT
6    else PAGESCANMODE
```

Η διαδικασία INIT(v) αποφασίζει εάν ένας κόμβος είναι init ή όχι. Μόνο οι init κόμβοι πηγαίνουν αρχικά σε page mode και αρχίζουν να κάνουν page έναν-έναν, όλους τους μικρότερους γειτονικούς κόμβους τους.

Στην συνάρτηση PAGE(u, v, 'master', v), η πρώτη παράμετρος u, συμβολίζει τη συσκευή που λαμβάνει το page. Η δεύτερη παράμετρος v, είναι η συσκευή που κάνει page στην u. Η τρίτη παράμετρος συμβολίζει το ρόλο της συσκευής v. Θα είναι δηλαδή master ή slave. Και η τελευταία παράμετρος, για μια συσκευή v που είναι slave, συμβολίζει, τον master που έχει. Στη περίπτωση που η συσκευή v είναι master, τότε στη θέση εκείνη, αναγράφεται απλά η ίδια η συσκευή, αφού δεν έχει κάποιον αλλο master. Ολες οι non init συσκευές πηγαίνουν σε page scan mode.

Ο παρακάτω ψευδοκώδικας εκτελείται από μια non init συσκευή v που λαμβάνει ένα page.

```
ONRECEIVINGPAGE(v, u, role of u, t)
1 RECORDROLE(u)
2 if (weight[v] < weight[u])
3   then if (role[u] == 'master')
4     then if (role[v] == 'none')
5       then master[v] = u
6            role[v] = 'slave'
7       else inform u on master[v]
8     if (some bigger neighbor z has to page)
9       then WAIT PAGE(v, z, r, w)
10    else PAGEMODE
11       if (for all bigger u: role[u] == 'slave')
12          then role[v] = 'master'
13               for each smaller z
14                   do PAGE(z, v, 'master', v)
15               for each bigger slave w
16                   do PAGE(w, v, 'master', v)
17               EXIT
18          else for each slave neighbor z
```

```
19              do k = master[v]
20                PAGE(z, v, 'slave', k)
21                PAGESCANMODE
22          else if (some smaller neighbor z has to page)
23              then WAIT PAGE(v, z, r, w)
24              else EXIT
```

Η διαδικασία καταγραφής του ρόλου της συσκευής u στη γραμμή 1, συμπεριλαμβάνει και όλες τις πληροφορίες συγχρονισμού, διευθυνσιοδότησης κλπ. που ενεργοποιεί την επικοινωνία με την v αν χρειαστεί.

Σε περίπτωση που η u είναι slave τότε καταγράφεται η ταυτότητα της συσκευής που έχει ως master. Λαμβάνοντας ένα page ένας κόμβος v από έναν κόμβο u, ελέγχει κατά πόσο είναι, ένας μεγαλύτερος κόμβος ο u, ή όχι. Εάν ο u είναι μεγαλύτερος από τον v, τότε ελέγχεται αν ο u είναι master. Εάν ισχύει αυτό και ο v δεν ανήκει σε κάποιο piconet, τότε γίνεται μέλος στο piconet του u. Σε περίπτωση όμως που το v ήδη ανήκει σε κάποιο piconet, τότε απλώς ενημερώνει τον u, πού ανήκει, και με ποιον master. Τότε ο κόμβος v εξετάζει εάν όλοι οι μεγαλύτεροι γείτονές του, του έχουν κάνει page. Εάν όχι, τότε περιμένει page και από άλλους κόμβους, οπότε και τερματίζει την εκτέλεση του κώδικα.

Όταν αισίως δεχτεί page από όλους τους μεγαλύτερους γείτονες, τότε ξέρει σίγουρα αν έχει γίνει μέλος κάποιου piconet, με master έναν από του μεγαλύτερους γείτονές του. Σ' αυτή τη περίπτωση, γίνεται μέλος του piconet, από το πρώτο master που του έκανε page. Σε διαφορετική περίπτωση, το v θα γίνει από μόνο του master. Σε κάθε περίπτωση μεταπίπτει σε page mode και ανακοινώνει την απόφασή του πρώτα σε όλους τους μικρότερους γείτονες και έπειτα σε μεγαλύτερους που είναι slaves.

Στο σημείο αυτό, ένας κόμβος v που είναι master, τερματίζει την εκτέλεση του κώδικα. Αν ο v είναι slave, τότε μεταπίπτει σε page mode, και περιμένει pages από όλους τους μικρότερους γείτονες. Υπάρχει περίπτωση ένας μικρότερος γειτονικός κόμβος του slave v, να μην έχει αποφασίσει ακόμα το ρόλο του. Με το που πάρει την απόφαση ένας κόμβος για το ρόλο του, κάνει page στους μεγαλύτερους γείτονές του, και τους ενημερώνει για το αν είναι master ή slave (αν είναι slave στέλνει και το ID του master του).

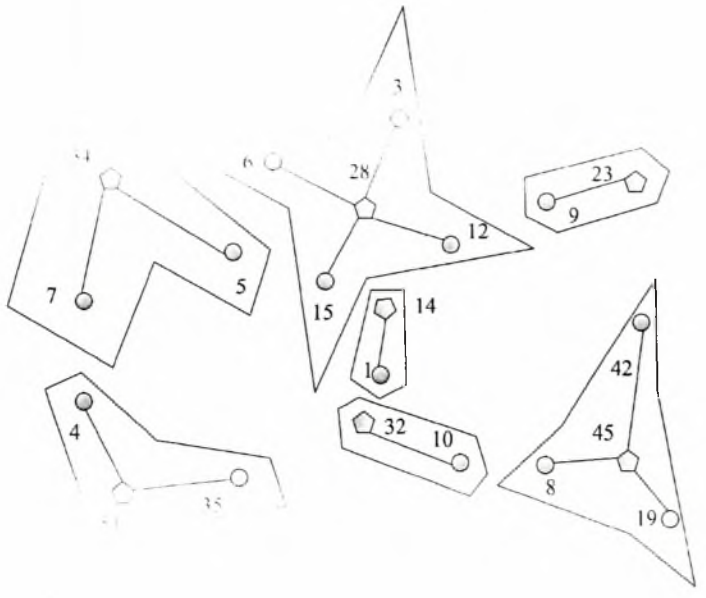
Αυτή η κίνηση, αφορά μια μετέπειτα φάση αυτού του πρωτοκόλλου. Σημειώνουμε ότι το εξωτερικό else, εκτελείται από έναν slave κόμβο, που έχει κάνει page όλους τους γείτονές του, και ο master έχει μια πλήρη εικόνα για τον ρόλο των γειτόνων του και τα ID των master τους. Βλέποντας το επόμενο παράδειγμα καταλαβαίνουμε καλύτερα τη λειτουργία τις συνάρτησης bluestar formation. Στη προηγούμενη εικόνα του δικτύου, κάθε ακμή παρίστανε ότι οι δύο κόμβοι βρίσκονται σε ακτίνα μετάδοσης, οπότε θεωρούνται «γειτονικοί», και κάθε τρίγωνος κόμβος ήταν init. Εδώ, αντί για βάρη θα χρησιμοποιήσουμε το αναγνωριστικό τους ID. Έτσι έχουν και όλες οι συσκευές διαφορετικό «βάρος».

Στην αρχή της φάσης του bluestar formation όλοι οι κόμβοι εκτελούν τη συνάρτηση: INITOPERATIONS. Έτσι οι συσκευές με τα μεγαλύτερα βάρη στη γειτονιά τους, θα είναι και οι init κόμβοι. Στο παράδειγμά μας init κόμβοι είναι οι : 51, 45, 34 και 28. Αυτοί με τη σειρά τους πηγαίνουν σε page mode και αρχίζουν να κάνουν page έναν-έναν τους γείτονές τους. Οι υπόλοιποι κόμβοι βρίσκονται σε page scan mode. Η συσκευή 51, επιτυχώς θα κάνει page τις συσκευές 4 και 35, οι οποίες και θα γίνουν slaves στο piconet του 51(είναι ο master, γι' αυτό και αναφέρεται ως piconet του 51). Το piconet του 45 αποτελείται από τον 45 κόμβο, που είναι και master και από τους 8, 19, 42, οι οποίοι είναι οι slave του.

Ο master 34 επιτυχώς κάνει page τους 5, 7 οι οποίοι θα γίνουν οι δύο slave, του δικού του piconet. Ο κόμβος 6, που είναι γείτονας του master 34, είναι μέλος του piconet 28. Διότι τον έκανε page πριν τον κάνει ο 34. Οι συσκευές 3, 12, 15, είναι επίσης μέλη του piconet 28. Στο σημείο αυτό, οι τέσσερεις init συσκευές τερματίζουν την εκτέλεση αυτής της φάσης του πρωτοκόλλου. Στο piconet 45, το slave 42 έχει δεχτεί page, από όλους τους μεγαλύτερούς του, γείτονες. Πηγαίνει σε page mode και αρχίζει να κάνει page όλους τους μικρότερους του, γείτονες. Συγκεκριμένα τον κόμβο 8 και 23. Λαμβάνοντας ένα page από τη συσκευή 42, που είναι slave του master 45, η συσκευή 23 αποφασίζει να ανακηρύξει τον εαυτό της ως master (αφού όλοι οι μεγαλύτεροι γείτονές της, είναι slaves). Έτσι κάνει page τους μικρότερους της γείτονες, και συγκεκριμένα την συσκευή 9, η οποία γίνεται και slave του piconet 23.

Παρομοίως η συσκευή 14, αφού δέχτηκε page από την 15, η οποία είναι slave του piconet 28, αποφασίζει να αυτοανακηρυχθεί ως master και να κάνει το δικό της piconet. Έτσι κάνει page τους κόμβους 1, 12(ο οποίος είναι όμως ήδη slave του piconet 28). Τελικά, μόνο ο 1 θα γίνει slave στο piconet της 14. Το piconet με master τον κόμβο 32, κατασκευάζεται αντίστοιχα, αφού δέχτηκε μήνυμα από τον 35, ότι είναι μέλος του piconet 51. Από όλους τους μικρότερους γείτονες του 32 (ο 1 και ο 10), μόνο ο 10 θα γίνει slave του piconet 32, αφού ο 1 είναι ήδη μέλος του piconet 14.

Τελικά από τις 21 συσκευές του δικτύου μας, εφτά έγιναν master (τέσσερις εκ των οποίων ήταν init) και όλοι οι υπόλοιποι έγιναν slaves σε έναν από αυτούς τους master. Τα αποτελέσματα τις φάσης bluestar formation, παρουσιάζονται στην παρακάτω εικόνα.



Ο κώδικας που υλοποιήθηκε σε γλώσσα προγραμματισμού C, για την υλοποίηση του παραπάνω πρωτοκόλλου παρουσιάζεται παρακάτω. Τα αποτελέσματα που προκύπτουν από το πρόγραμμα αυτό, είναι ο ρόλος κάθε κόμβου (master ή slave), ενός δικτύου.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define n 10

struct node{
        int two_hop_neib[10];
        int has_page[10];
        int init;      //1=init  0=NONinit
        char mode[10];  //page h pagescan
        char role[10];  // einai slave h master
        int master;     //edw 8a valoume poio exei master, an einai auto master tote vazoume ton eauto tou.
        }nodes[n];

        call_page(nodes, a);
        call_two_hop_neib(nodes, a);

return 0;
}

void call_arey_file(){

        }

        FILE  *in_file;

        in_file=fopen("connected_nodes.txt","r");
        if(in_file == NULL) {
  printf("Cannot open file.\n");

        fclose(in_file);
}

void call_initial(struct node nodes[], int a[n][n]){
```

```
	for(i=0; i<10; i++){
		for(j=0; j<10; j++){
			nodes[i].has_page[j]=0;
			nodes[i].two_hop_neib[j]=0;
		}
	}

		for(i=0; i<10; i++){
				nodes[i].init=0;			//8etw ta prwta 8 ws non init
				[illegible] "pagescan");	//pagescan mode



		for(j=n-1; j>=0; j--){
			if((a[i][j]==0)&&(i!=j)&&(num_init<2)&&(nodes[i].init!=1)&&(nodes[j].init!=1)){
				nodes[i].init=1;	//8etw ta teleutaia 2 (kai megalitera) ws init nodes
				strcpy(nodes[i].mode, "page");	//page mode
				strcpy(nodes[i].role, "master");
				nodes[i].master=i;
				num_init++;
				nodes[j].init=1;
				strcpy(nodes[j].mode, "page");	//page mode
				strcpy(nodes[j].role, "master");
				[illegible]



		}
	}
}

void call_page(struct node nodes[], int a[n][n]){
	int v, u;

	for(v=0; v<10; v++){

								[illegible] gia init node, kai mikroterous geitones


		}
	}
}

void call_two_hop_neib(struct node nodes[], int a[n][n]){
	int i, j;
	for(i=0; i<n; i++){
		printf("o komvos: %d  einai  %s  kai exei Master ton %d \n", i, &nodes[i].role, nodes[i].master);



			if(nodes[i].two_hop_neib[j]==1){printf("%d ", j);}
		}
```

```
void page(int v, int u, struct node nodes[], int a[n][n]){
        int all_nei_slave=0;
        int i, j, d;
        printf(" to %d kanei page sto %d\n", u, v);
                if(!strcmp(nodes[u].role, "master")){
                        if((!strcmp(nodes[v].role, "none"))&&(u>v)){
                                nodes[v].master=u;
                                strcpy(nodes[v].role, "slave");}

                                /*elegxos an kapoios geitonas z has to page)*/
                                if((a[v][i]==1)&&(i>v)){
                                        if(/*(!strcmp(nodes[i].mode, "page"))&&*/(nodes[i].has_page[v]==0)){
                                                return;
                                        }
                                }
                        }
                }
                strcpy(nodes[v].mode, "page");

                                        = !strcmp(nodes[u].role, "none")&&(strcmp(nodes[i].role,

                                                all_nei_slave=1;
                                        }
                                        else{
                                                all_nei_slave=0;
                                                break;
                                        }
                                }

                        }


                                for(i=0; i<n; i++){  //for each smaller z  page(z, v, 'master', v);
                                                if((a[u][i]==1)&&(i<u)){
                                                        strcpy(nodes[u].role, "master");
                                                        nodes[u].master=u;
                                                        printf("kainourio master1\n");
                                                        nodes[u].has_page[i]=1;
                                                        page(i, u, nodes, a);}
                                        }

                                return;


                                for(i=0; i<n; i++){ //for each slave(vazw egw kai "none") neighbor z
printf("%d, %d, slave , nodes[v].master);
```

```
                                                                    for(d=n; d>=0; d--){
        if((a[j][d]==1)&&(strcmp(nodes[d].role, "master"))){
        nodes[j].has_page[d]=1;
        page(d, j, nodes, a);
                                                                                }
                                                                    }
                }
        }
}

void two_hop(int v, struct node nodes[], int a[n][n]){
        int i, j;
        for(i=0; i<n; i++){
                if(a[v][i]==1){

}
```

3. Το BCSF πρωτόκολλο (Betweeness Centrality Scatternet Formation)

3.1 Τι είναι και πώς υπολογίζεται ο NI ;

Όλοι οι broadcasting αλγόριθμοι που έχουν προταθεί μέχρι τώρα, παρουσιάζουν κάποιες αδυναμίες. Μερικοί από αυτούς βασίζονται στο node ID, με σκοπό την ελάττωση των περιττών broadcasts, ή την εύρεση προτεραιοτήτων. Αυτές οι προσεγγίσεις, έχουν το μειονέκτημα ότι δεν μπορούν να εντοπισουν όλες τις πιθανές εξαιρέσεις, για το λόγο ότι βασίζονται πάνω στο node ID. Σαν αποτελεσμα, να εχουμε πολλές άσκοπες αναμεταδώσεις. Άλλες μέθοδοι, βασίζονται στη γνώση «τοπικής» πληροφορίας, όπως είναι η γνώση της κ-hop γειτονιάς (κ>2). Υπάρχουν και οι μέθοδοι, που είναι εξαιρετικά ακριβές, παράγοντας ένα κόστος, της τάξης του O(f^2) ή O(f^3), όπου f είναι ο μέγιστος βαθμός ενός κόμβου, σε ένα ad-hoc δίκτυο. Ωστόσο, και αυτό το κόστος δεν είναι απαγορευτικά μεγάλο, γίνεται όμως μεγάλο, όταν αυξημένη κινητικότητα στο δίκτυο, αλλάζει τις υπάρχουσες, τοπικά υπολογισμένες, πληροφορίες. Τέλος υπάρχουν και μέθοδοι, που δεν εκμεταλλεύονται σωστά, τις δοθείσες πληροφορίες, όπως, η χρήση του βαθμού ενός κόμβου, για την προτεραιότητα του, στον υπολογισμό του βαθμού κυριαρχίας του, μπορεί να μην αποτελεί την καλύτερη τοπική απόφαση. Εδώ προτείνεται, ένα καινούριο πρωτόκολλο broadcasting για ad-hoc δίκτυα, όπου:

- Είναι τοπικό, άρα και κατανεμημένο. Μπορεί να εξάγει πληροφορίες για την 1-hop, 2-hop, k-hop γειτονία. Παρέχοντας διαφορετικά εναλλαγές στην αποδοτικότητα, σε σχέση με το κόστος επικοινωνίας. Για χάρη ευκολίας, εδώ παρουσιάζεται μέχρι και για την 2-hop γειτονιά.
- Εισάγεται μια νέα μέθοδος υπολογισμού της σημαντικότητας του κόμβου.
- Υπολογίζεται ο βαθμός σημαντικότητας του κόμβου, σε χρόνο ανάλογο του αριθμού των κόμβων του δικτύου, και του αριθμού των ακμών του κόμβου αυτού, με το υπόλοιπο δίκτυο, και άσχετα με τον βαθμό του κάθε κόμβου.
- Υπολογίζει την επόμενη «απόφαση» του δικτύου, πράγμα που είναι χρήσιμο για σύγχρονα δίκτυα μεγάλης κινητικότητας.

Ένα ασύρματο ad-hoc δίκτυο περιγράφεται ως: G(V, E). Μια ακμή e=(u, v), υφίσταται, αν και μόνο αν το u βρίσκεται σε ακτίνα μετάδοσης με το v, και το αντίστροφο. Όλα τα links στο γράφο είναι διπλής κατεύθυνσης. Το δίκτυο θεωρείται ότι είναι σε μια συνδεδεμένη κατάσταση. Το σύνολο τωβ γειτόνων του v κόμβου, παριστάνονται ως εξής: N1(v), $N1(v) = \{u : (v,u) \in E\}$. Το σύνολο των 2-hop κόμβων του v, είναι οι κόμβοι, που είναι γείτονες των γειτονικών κόμβων του v, είναι γείτονες δηλαδή του N1(v), και παριστάνεται σαν N2(v), $N2(v) = \{W : (u,w) \in E$, όπου w διάφορο του v και το w δεν υπάρχει στο NI και $(v, u) \in E\}$. Ο συνδυασμός και των δύο, μας δίνει το N12(v).

Ορισμός 1. Μια τοπική όψη του δικτύου εμφανίζεται σαν LNv, του δικτύου G(V, E). Ένας κόμβος v που ανήκει στο V, είναι το υποδίκτυο του G μαζί με την 2-hop γειτονιά του.

Υπολογίζουμε ένα μονοπάτι από το u E V στο w E V, σαν εναλλακτικό σετ από διανύσματα και ακμές, ξεκινώντας από το u και καταλήγοντας στο w, έτσι ώστε να σχηματίζεται η διαδρομή με ένα

διάνυσμα. Το μήκος του μονοπατιού, είναι ο αριθμός των ακμών. Συμβολίζουμε την απόσταση μεταξύ του u και w σαν d(u, w), και είναι η μικρότερη διαδρομή που μπορούμε να έχουμε στο δικτυο μας. Εξ ορισμού, d(v, v)=0. Και d(v, w)=d(w, v) στο ίδιο δίκτυο πάντα. Να σημειώσουμε ότι η απόσταση δεν εξαρτάτε από την καθυστέρηση σε κάθε ακμή, απλά εξάγεται από τον αριθμό των hops.

Υπολογίζοντας την σημαντικότητα του κόμβου

Εδώ παρουσιάζεται, μια καινούρια αντιμετώπιση της σημαντικότητας του κόμβου, αποφεύγοντας τα πισωγυρίσματα. Θεωρούμε, σ uw=σ wu, την μικρότερη δυνατή διαδρομή, από το u E V στο w E V (εξορισμού, σ uu=0). Θεωρούμε ότι σ uw(v) την μικρότερη δυνατή διαδρομή, από το u στο w, αλλά διαμέσου του v. Έπειτα υπολογίζουμε τον πίνακα σημαντικότητας κόμβων (node importance index), για κάθε κόμβο ξεχωριστά. Για τον κόμβο v, είναι ο NI(v).

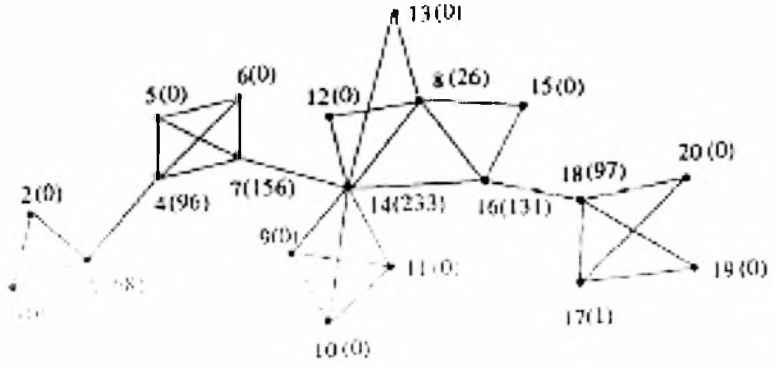
Ορισμός 2. Ο πινακας σημαντικότητας κόμβου για έναν κόμβο v, είναι ίσος με:

$$NI(v) = \sum_{u \neq v \neq w \in V} \frac{\sigma_{uw}(v)}{\sigma_{uw}}.$$

Μεγάλοι αριθμοί, αυτού του πίνακα, υποδηλώνουνε, ότι ο κόμβος v, μπορεί να προσπελάσει μεγάλο αριθμό άλλων κόμβων, που περνάν από το ίδιο μονοπάτι, ή ότι είναι η «γέφυρα» που περνάν, πολλά άλλα, συντομότερα μονοπάτια.

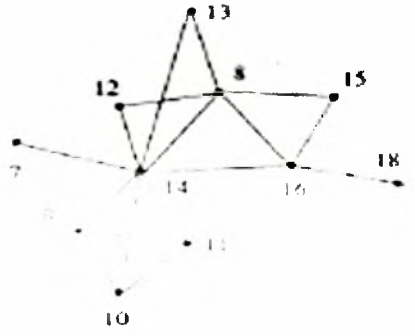
3.2 Παράδειγμα σε ένα γράφημα

Στο παρακάτω σχήμα, μπορούμε να δούμε τον NI πίνακα, κάθε κόμβου, δίπλα από το χαρακτηριστικό του ID. Ο αριθμός του NI γράφεται μέσα στην παρένθεση.



Μπορούμε εύκολα να παρατηρήσουμε την μεγάλη ομοιότητα, των κόμβων με μεγάλο αριθμό NI, και των κόμβων που χαρακτηρίζονται σαν πύλες διασύνδεσης, άλλων κόμβων. Τώρα που ο NI πίνακας υπολογίσθηκε για όλο το δίκτυο συνολικά, μπορούμε να παρατηρήσουμε δομικά χαρακτηριστικά του δικτύου, καλυτέρα απ' ότι ο node degree κάνει. Επιπλέων επιφέρει ένα βαθμό σημαντικότητας του κόμβου, σε σχέση με τη συνεισφορά του, σε όλο το δίκτυο. Για την ακρίβεια, ο αριθμός του NI, υποδεικνύει, αυτό που ονομάζεται, geodesic node του δικτύου, π.χ. κόμβοι που λειτουργούν σαν σημεία άρθρωσης, ή κόμβοι που έχουν πολύ μεγάλο βαθμό συγκριτικά με τους γείτονες τους.

Ο NI πινάκας, θα ήταν χρήσιμος για την σχεδίαση ενός broadcasting πρωτοκόλλου σε ad-hoc δίκτυα, μόνο αν συγκεκριμενοποιηθεί σε επιμέρους τμήματα του συνολικού δικτύου. Π.χ., για την 2-hop γειτονία του κάθε κόμβου, και μόνο αν μπορεί να υπολογισθεί πραγματικά γρήγορα. Με την προϋπόθεση των παραπάνω συνθηκών, μπορεί να χρησιμοποιηθεί, για την σχεδίαση αλγορίθμων εντοπισμού. Ευτυχώς ισχύουν και τα δύο. Μπορούμε να παρατηρήσουμε ότι, για κάθε κόμβο v του πινακα NI, το N12(v), υπολογίζεται, μόνο για το υποδίκτυο LNv, αποκαλύπτοντας έτσι την σχετική σημαντικότητα του κόμβου v στο υποδίκτυο N12, (η όψη του δικτύου από τη μεριά του v). Βρίσκουμε δηλαδή την 2-hop γειτονία για κάθε κόμβο, και πάνω σε αυτήν την 2-hop γειτονιά, μπορούμε να αρχίσουμε τις μετρήσεις μας. Για παράδειγμα στο παρακάτω σχήμα εμφανίζεται, η 2-hop γειτονιά το κόμβου 8, του παραπάνω δικτύου.



Ο NI πίνακας, για την 2-hop γειτονιά του 8, LN8, φαίνεται στον παρακάτω πίνακα. Για έναν κόμβο u που ανήκει στην 2-hop γειτονία του v, ο πίνακας NI γράφεται, NI v(u).

$n(ode)$	$NI_8(n)$	$n(ode)$	$NI_8(n)$	$n(ode)$	$NI_8(n)$
7	0	11	0	15	0
8	14	12	0	16	23
9	0	13	0	18	0
10	0	14	65		

Με μια πρώτη ματιά, ο υπολογισμός του NI φαίνεται ακριβός, O(m*n^2) πράξεις συνολικά για μια 2-hop γειτονιά, η οποία αποτελείται από n κόμβους και m ακμές. Ευτυχώς, δεν είναι ακριβώς έτσι, αν παρατηρήσουμε λίγο καλύτερα τα πράγματα. Παρακάτω θα δώσουμε τον υπολογισμό του NI πίνακα σε ψευδοκώδικα, και εκεί φαίνεται ότι δεν υπάρχει τελικά τόσο μεγάλο κόστος υπολογισμού. Αυτόν τον ψευδοκώδικα θα χρησιμοποιήσουμε και για να υπολογίσουμε τον NI πίνακα, σε κάθε 2-hop γειτονιά. Ο αλγόριθμος λαμβάνει καλά υπόψη του, τον υπολογισμό, των διαφόρων σύντομων μονοπατιών, μεταξύ δύο κόμβων.

Θεώρημα 1. Ο αλγόριθμος CalculateNodeImportanceIndex, είναι σωστός, στη περίπτωση που εκτελείται σε ένα δίκτυο, ή κομμάτι δικτύου, και υπολογίζει σωστά, τους αριθμούς των μονοπατιών που περνούν από έναν κόμβο (του συνολικού δικτύου, ή της γειτονιάς).

Θεώρημα 2. Η πολυπλοκότητα του αλγορίθμου CalculateNodeImportanceIndex, είναι O(n*m), για ένα γράφημα με n κόμβους και m ακμές.

```
Algorithm CalculateNodeImportanceIndex(graph G(N, L))
  N: set of graph nodes, L: set of links between these nodes
  Output: Array NI[1..N] index value for each node of N
begin
NI[t] = 0, για κάθε t E N;
foreach( n E N ) do{
        S: an empty stack;
        P[-1: array of empty lists (one list V node w E N);
        σ[.]: anarray,where σ[t]=O,   για κάθε t E N; σ[n]=l;
        d[.]: an array, where d[t] = -1,για κάθε t E N; d[n]=O;
        Q: an empty queue;
        Q.enqueue(n);
        while( Q.isNotEmpty() ){
                u = Q.dequeue();
                S.pus h(u) ;
                foreach 1-hop neighbor w of u do
                        // w found for the first time?
                        if( d[w] < 0 ) then
                                Q.enqueue(w);
                                d[w] = d[u] + 1;
                        //shortest path to w via u?
                        if( d[w] == (d[u] + 1) ) then
                                σ[w] =σu[w] + σ[u];
                                P[w] .append(v) ;
                }
        δ : an array, where δ[t] = 0,Vt E N;
        // S returns nodes in order of non-increasing hop distance from n
        while( S.isNotEmpty() ) d o
                w = S.pop();
                foreach( v E P[w] ) do
                        δ[u] = δ[u] + σ(u)/σ(w) * (1 + δ[w]);
                if( w !=s )
                        NI[w] = NI[w] + δ[w];
        }
return NI;
end
```

Ο κώδικας σε περιβάλλον γλώσσας προγραμματισμού C, που κατασκευάστηκε για τον παραπάνω αλγόριθμο, πηγάζει από τον ψευδοκώδικα του, και είναι ο εξής (υπολογίζουμε και τις 2-hop γειτονιές κάθε κόμβου):

```
int a[n][n];

void call_arey_file();
int calculate_intex_foreach_node(int r, int a[n][n]);
int enqueue(int k);
int dequeue();
int is_empty_q();
void push(int g);
int pop();
```

```
        int num_of_neib;
        int two_hop_neib[n];
}nodes[n];

struct list{
        int id;
        struct list *next;
}*cur, *cur_ap, *cur_s, *head, *tail, *last[n], *top, *p[n], *pass;

                two_hop(r, nodes, a);
        }

        for(r=0; r<n; r++){
                calculate_intex_foreach_node(r, a);
        }

return 0;
}

                int d[n];
                float delta[n], s[n];
                int i, j;

                for(j=0; j<n; j++){
                        NI[j]=0;
                }

                for(i=0; i<n; i++){
                        if(nodes[r].two_hop_neib[i]==1){

                                                d[j]=-1, //array

                                                while(p[j]!=NULL){  //P[.] array of empty lists
                                                        cur_ap=p[j];
                                                        p[j]=cur_ap->next;
                                                        free(cur_ap);
                                                }
                                                last[j]=NULL;
                                                p[j]=NULL;
                                        //}

                                        cur_s=top;
                                        top=cur_s->next;
```

```
                free(cur_s);
        }

        if(head!=NULL){
                while(head!=NULL){//empty queue
                        cur=head;
                        head=cur->next;



        enqueue(r);

        while(!(is_empty_q())){
                v=dequeue();
         push(v);
                for(w=0; w<n; w++){  //one hop neighbor
                        if(nodes[r].two_hop_neib[w]==1){
                                if(a[v][w]==1){
                                        //printf("\ngeitonas o %d", w);
                                        if(d[w]<0){
                                                enqueue(w);
                                                d[w]=d[v]+1;

                                        if(d[w]==(d[v]+1)){
                                                s[w]=s[w]+s[v];
                                                append(v, w, p); //P[w].append(v)
                                        }
                                }
                        }
                }


        }
        //delta[i]: an array, where delta[t] = 0


                w=pop();
                pass=p[w];
                while(pass!=NULL){
                        if(nodes[r].two_hop_neib[pass->id]==1){
                                delta[pass->id]=delta[pass->id]+((s[pass-
>id]/s[w])*(delta[w]+1));

                                pass=pass->next;
                        }
                }
                if(w!=i){


        printf("\n");
        printf("o NI pinakas gia tin 2_hop geitonia tou komvou %d einai:\n", r);
        for(rz=0; rz<n; rz++){
                if(nodes[r].two_hop_neib[rz]==1){
                        printf("NI[%d]=%f\n", rz, NI[rz]);
                }
```

```
                        }
return 1;
}

void call_arey_file(){
        int i, j;
        for(i=0; i<n; i++){




        FILE *in_file;

        in_file=fopen("connected_nodes.txt","r");
        if(in_file == NULL) {
  printf("Cannot open file.\n");
  exit(1);
        }else{
                while(feof(in_file)== 0){
                fscanf(in_file, "%d%d", &i, &j); // read from file




                for(j=0; j<n; j++){
                        printf("%d ", a[i][j]);
                }
                printf("\n");
        }
}

int enqueue(int k){




        else{
                cur = (struct list *)malloc (sizeof (struct list));
                cur->id=k;
                tail->next=cur;
                tail=cur;
                tail->next=NULL;
        }
        return 1;
}




  fflush( stdout), getchar();
  return 0;
        }else{
                h=head->id;
                cur=head;
                head=head->next;
```

```
                free(cur);
                return h;
        }
}

        }else{
                h=0;
                return h;
        }
}

void push(int g){
        if(top==NULL){
                cur_s = (struct list *)malloc (sizeof (struct list));
                cur_s->id=g;
                cur_s->next=top;
                top=cur_s;
        }
}

int pop(){
        int h=0;
        if(top==NULL){
                printf("empty stuck\n");
prohgoumenos
    //cur_s= 0;
        }
        return h;
}

int is_empty_s(){
        int h;
        if(top==NULL){
}
void append(int h, int i, struct list *p[n]){
        if(p[i]==NULL){
                cur_ap = (struct list *)malloc (sizeof (struct list));
                cur_ap->id=h;
                p[i]=cur_ap;
                last[i]=cur_ap;
                cur_ap->next=NULL;
        }else{
```

```
cur_ap = (struct list *)malloc (sizeof (struct list));
cur_ap->id=h;
last[i]->next=cur_ap;
```

```
int i,j;
nodes[v].num_of_neib=0;
nodes[v].two_hop_neib[v]=1;
for(i=0; i<n; i++){
        if(a[v][i]==1){
                nodes[v].two_hop_neib[i]=1;
                for(j=0; j<n; j++){
                        if((v!=j)&&(a[i][j]==1)){
                                if(nodes[v].two_hop_neib[j]==0){
                                        nodes[v].two_hop_neib[j]=1;
```

```
for(i=0; i<n; i++){
        if(nodes[v].two_hop_neib[i]==1){
                nodes[v].num_of_neib++;
        }
}

printf("O kombos %d exei %d geitones", v, nodes[v].num_of_neib);
for(i=0; i<n; i++){
        if(nodes[v].two_hop_neib[i]==1){
                printf(" %d",i);
```

3.3 Οι συνιστώσες (rounds) του πρωτοκόλλου

Το πρωτόκολλο αυτό, σχεδιάστηκε, με σκοπό τον διαχωρισμό-ορισμό των κόμβων ενός δικτύου, σε masters και slaves. Όπως παρατηρήσαμε και στο προηγούμενο κεφάλαιο, που μελετήσαμε το BlueNet, το BlueMesh, και το BlueStar, και οι τρείς αλγόριθμοι, οδηγούν σε μια [illegible] των master και των slave σε ένα ad-hoc δίκτυο κόμβων Bluetooth. Σκοπός ήτανε, η σύνενωση των επιμέρους piconets, με σκοπό τη δημιουργία, ενός scatternet. Τα piconets αυτά δεν συνδεόταν με τυχαίο τρόπο, άλλα σύμφωνα με κάποιους κανόνες, που όριζε ο εκάστοτε αλγόριθμος, και διαφορετικοί σε κάθε πρωτόκολλο.

Στο δικό μας πρωτόκολλο-αλγόριθμο, σκοπός ήταν, η επιλογή των master και των slaves, σε ένα δίκτυο κόμβων, με τη χρήση του NI πίνακα που εξηγήσαμε προηγουμένως. Αρχικά, υπολογίζουμε τον NI πίνακα, αλλά για κάθε 2-hop γειτονιά (κάθε κόμβου) ξεχωριστά. Παρακάτω δίνουμε τα πέντε βασικά βήματα, που ακολουθεί ο αλγόριθμος μας, και στη πορεία θα περιγράφουμε το καθένα από αυτά.

- 1st round: Υπολογισμός του πίνακα NI για κάθε κόμβο. (για κάθε 2-hop γειτονιά)
- 2st round: Αποφασίζω, ποιοι θα γίνουν masters.
- 3st round: Γνωστοποιώ την απόφασή μου να γίνω master.
- 4st round: Δέσμευση των slaves.
- 5st round: Ανακατανομή των slaves και δημιουργία καινούριων masters.

Αρχικά, στον πρώτο γύρο του πρωτοκόλλου, γίνεται ο υπολογισμός του NI πίνακα, σύμφωνα πάντα με τον αλγόριθμο που περιγράφηκε προηγουμένως. Ο υπολογισμός, γίνεται για κάθε κόμβο ξεχωριστά, και για την δικιά του 2-hop γειτονιά. Έτσι, για κάθε κόμβο θα έχω έναν ξεχωριστό NI πίνακα, μεγέθους όσο είναι και το πλήθος των 2-hop γειτόνων του. Για λόγους ευκολίας, κατασκευάζουμε έναν δισδιάστατο πίνακα NI μεγέθους nxn (όπου n το πλήθος των κόμβων του δικτύου). Αρχικά τα στοιχεία του πίνακα, έχουν την τιμή -1. Η πρώτη γραμμή συμβολίζει τον κόμβο 0 και την 2-hop γειτονιά του. Κάθε στήλη παριστάνει έναν κόμβο. Αν ένας κόμβος δεν ανήκει στην 2-hop γειτονιά της συγκεκριμένης γραμμής, τότε το κελί του πίνακα NI[[η][η], παραμένει -1, σε διαφορετική περίπτωση, παίρνει την τιμή του NI που υπολογίσαμε για τη συγκεκριμένη γειτονιά.

Στο δεύτερο γύρο, αποφασίζω ποιους κόμβους θα κάνω master. Θέτω έναν κανόνα που λέει: Κάνω master τους κόμβους που στη γειτονιά τους, έχουν το μεγαλύτερο NI αριθμό. Αυτοί οι κόμβοι από εκεί και πέρα γίνονται masters.

Στον τρίτο γύρο, στέλνω μηνύματα, στους υπόλοιπος κόμβους για να τους γνωστοποιήσω ότι έγινα master.

Στον τέταρτο γύρο, αποφασίζω ποιοι κόμβοι θα είναι οι standards slaves. Standards slaves, γίνονται οι κόμβοι, όπου το άθροισμα όλων των NI από τις γειτονιές που ανήκουν, είναι μηδέν. Υπάρχουν κόμβοι, όπου το NI τους είναι πάντα 0, σε όποια γειτονιά και αν ανήκουν. Αυτοί γίνονται κατευθείαν slaves.

Στον πέμπτο, και σημαντικότερο γύρο, ελέγχουμε για τους περιορισμούς που ισχύουν σε ένα τέτοιο δίκτυο.

- Αρχικά παίρνω μία απόφαση, και κάνω master τον κόμβο(από αυτούς που ακόμα δεν έχουν αποφασίσει αν θα είναι masters ή slaves), που έχει το μεγαλύτερο, αθροιστικά NI από όλες τις γειτονίες που αυτός ανήκει.
- Με το που γίνει αυτό, κάνω slaves, τους κόμβους που ανήκουν στην 1-hop γειτονιά του νέου master(αφού ο master ο καινούριος δημιουργεί το δικό του Piconet).
- Έπειτα αυτό που ελέγχω , είναι μήπως υπάρχει κάποιο slave, χωρίς master στη δική του 1-hop γειτονιά.
    - Σε περίπτωση που υπάρχει, αποφασίζω και κάνω master τον κόμβο, από τους 1-hop γείτονες του slave, εκείνον που θα έχει αθροιστικά το μεγαλύτερο NI στις γειτονιες που ανήκει(είτε είναι "none", είτε είναι slave).
      Κανω slaves τους κόμβους που ήταν none και είναι γείτονες με τον νέο master.

- Έπειτα, θα έχω κάποιους κόμβους που θα έχουν παραμείνει «none»(ούτε slave ούτε master). Για αυτούς τους κόμβους πρέπει να πάρω απόφαση τη ρόλο θα έχουν στο δίκτυό μας.
    - Αν έχει master γείτονα, τότε θα γίνει slave του, αλλιώς αν είναι όλοι slave, τότε γίνεται ο master και κάνει το δικό του piconet, παίρνοντας, τους γειτονικούς του slaves.
- Δεν πρέπει να ξεχνάμε ότι στα Bluetooth ad-hoc δίκτυα, δεν μπορεί ένας master να έχει υπό την κατοχή του, πάνω από επτά slaves. Άρα κάνω τον έλεγχο για κάθε master, πόσα slaves έχει.
    - Αν έχει κάποιος πάνω από επτά, τότε υπολογίζονται όλα τα NI των slaves της 2-hop γειτονιάς του, αθροιστικά,(κάθε ένα για τις γειτονίες που αυτό ανήκει).
    - Master γίνεται αυτό, με το μεγαλύτερο αθροιστικά NI.

Στη φάση αυτή, τελειώνει ο αλγόριθμός μας, και όλοι οι κόμβοι του δικτύου μας, έχουν πάρει από ένα ρόλο. Ισχύουν οι περιορισμοί για τα Bluetooth ad-hoc δίκτυα, και έχει δημιουργηθεί το scatternet, για την επικοινωνία των συσκευών.

Ο κώδικας σε προγραμματιστικό περιβάλλον C, που κατασκευάστηκε, για τον αλγόριθμο αυτόν δίνεται παρακάτω.

```
#define n 20

int a[n][n];
int k;

void call_arey_file();
int calculate_intex_foreach_node(int r, int a[n][n]);

void append(int b, int i, struct list *p[]);
void two_hop(int v, struct node [], int a[n][n]);
void select_master(struct node [], float arayNI[n][n]);

float arayNI[n][n];
float NI[n];

struct node{
        int num_of_neib;

        int id;
        struct list *next;
```

```
}*cur, *cur_ap, *cur_s, *head, *tail, *last[n], *top, *p[n], *pass;

        for(r=0; r<n; r++){
                two_hop(r, nodes, a);
        }

        for(i=0; i<n; i++){
                strcpy(nodes[i].role, "none");
                for(j=0; j<n; j++){
                        arayNI[i][j]=-1;

                calculate_intex_foreach_node(r, a);
        }

        select_master(nodes, arayNI);

return 0;
}

int calculate_intex_foreach_node(int r, int a[n][n]){

                int i, j;

                for(j=0; j<n; j++){
                        NI[j]=0;
                }

                for(i=0; i<n; i++){

                                                        while(p[j]!=NULL){  //P[.] array of empty lists
                                                                cur_ap=p[j];
                                                                p[j]=cur_ap->next;
                                                                free(cur_ap);
                                                        }
                                                        last[j]=NULL;
                                                        p[j]=NULL;
                                                //}
                                        }
```

```
while(top!=NULL){//empty stuck
        cur_s=top;
        top=cur_s->next;
        free(cur_s);
}

if(head!=NULL){
        while(head!=NULL){//empty queue
                cur=head;
                head=head->next;
        }
        tail=NULL;
}

enqueue(i);

while(!(is_empty_q())){
        v=dequeue();
         push(v);
        for(w=0; w<n; w++){  //one hop neighbor
                if(nodes[r].two_hop_neib[w]==1){
                        if(a[v][w]==1){
                                if(d[w]<0){
                                        enqueue(w);
                                        d[w]=d[v]+1;
                                }
                                if(d[w]==(d[v]+1)){
                                        s[w]=s[w]+s[v];
                                        append(v, w, p); //P[w].append(v)
                                }
                        }
                }
        }
}
//delta[l]: an array, where delta[t] = 0
for(j=0; j<n; j++){
        delta[j]=0;
}
pass=p[w];
while(pass!=NULL){
        if(nodes[r].two_hop_neib[pass->id]==1){
                delta[pass->id]=delta[pass->id]+((s[pass->id]/s[w])*(delta[w]+1));
                pass=pass->next;
        }
}
if(w!=i){
        NI[w]=NI[w]+delta[w];
}
}

nb=0;
for(rz=0; rz<n; rz++){
```

```
void call_arey_file(){
        int i, j;

        for(i=0; i<n; i++){
                for(j=0; j<n; j++){
                        a[i][j]=0;

        in_file=fopen("connected_nodes.txt","r");
        if(in_file == NULL) {
  printf("Cannot open file.\n");
  exit(1);
        }else{
                while(feof(in_file)== 0){
                fscanf(in_file, "%d%d", &i, &j); // read from file
                a[i][j]=1;
                }

                printf("\n");
        }
}

int enqueue(int k){

        if(head==NULL){
                cur = (struct list *)malloc (sizeof (struct list));

                cur->id=k;
                tail->next=cur;
                tail=cur;
                tail->next=NULL;
        }
        return 1;
}

int dequeue(){
```

```
fflush( stdout), getchar();
return 0;
        }else{
                h=head->id;
                cur=head;
                head=head ->next;

int is_empty_q(){
        int h;
        if(head==NULL){
                h=1;
                return h;
        }else{
                h=0;
                return h;
        }

                top=cur_s;
                top->next=NULL;
        }else{
                cur_s = (struct list *)malloc (sizeof (struct list));
                cur_s->id=g;
                cur_s->next=top;
                top=cur_s;
        }

                h=top->id;
                cur_s=top;
                top=top->next;
                free(cur_s);  //Kaneis free ton cur_s o opoios eite den exei desmeftei ka8olou eite egine free proigoumenos
  //cur_s= 0;
        }
        return h;
}

        }else{
                h=0;
                return h;
        }
}

void append(int h, int i, struct list *p[n]){
```

```
                last[i]=cur_ap;
                cur_ap->next=NULL;
        }else{
                cur_ap = (struct list *)malloc (sizeof (struct list));
                cur_ap->id=h;
                last[i]->next=cur_ap;
                cur_ap->next=NULL;
                last[i]=cur_ap;
                }
```

```
        nodes[v].num_of_neib=0;
        nodes[v].two_hop_neib[v]=1;
        for(i=0; i<n; i++){
                if(a[v][i]==1){
                        nodes[v].two_hop_neib[i]=1;
                        for(j=0; j<n; j++){
                                if((v!=j)&&(a[i][j]==1)){
                                        if(nodes[v].two_hop_neib[j]==0){
                                                nodes[v].two_hop_neib[j]=1;
```

```
        }
        for(i=0; i<n; i++){
                if(nodes[v].two_hop_neib[i]==1){
                        nodes[v].num_of_neib++;
                }
        }

        printf("O kombos %d exei %d geitones", v, nodes[v].num_of_neib);
```

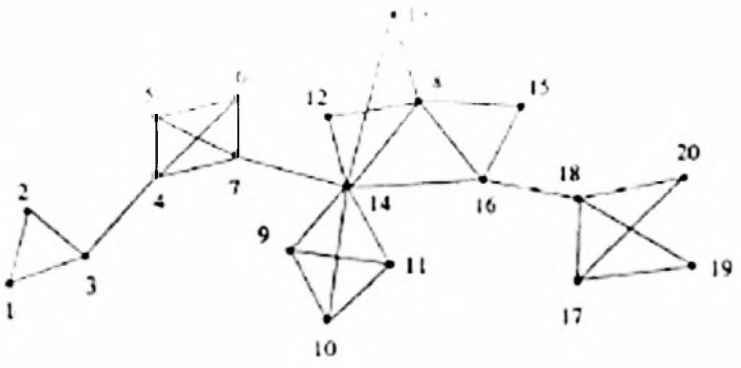
```
}

void select_master(struct node nodes[], float arayNI[n][n]){
        int i, j, sa, w, sq;
        float t, t_big, q, s_q, s_w;
        printf("\n");
        for(i=0; i<n; i++){//orizw tous standar master
                t=1;
                for(j=0; j<n; j++){
```

```
                t=0;
                for(i=0; i<n; i++){
                        if(arayNI[i][j]>=0){
                                t=t+arayNI[i][j];
                        }
                }
                if(t==0){strcpy(nodes[j].role, "slave");}
}

                                                        hop pou anoikei

                                                        none")))&&(arayNI[i][j]>=0)){
                                t=t+arayNI[i][j];
                                //if(arayNI[i][i]>t){
                                //      t=arayNI[i][i];
                                //      sa=i;
                        }
                }
                if(t>t_big){
                        t_big=t;
                        sa=j;



                        nodes pou itan "none" kai einai geitones me ton kainourio "master"
                if((a[sa][i]==1)&&(!(strcmp(nodes[i].role, "none")))){strcpy(nodes[i].role, "slave");}
}


for(i=0; i<n; i++){//edw apokleiw to gegonos na yparxei slave xwris master
                t=0;
                if(!(strcmp(nodes[i].role, "slave"))){
                        for(j=0; j<n; j++){
                                if((a[i][j]==1)&&(!strcmp(nodes[j].role, "master"))){
                                        t=1;//edw elenxw an exei estw kai 1 master sto 1-hop



                                                        auton pou exei to megalutero NI einai eite "none" eite

                                for(w=0; w<n; w++){
                                        if((a[i][w]==1)&&(arayNI[i][w]>=q)){
                                                q=arayNI[i][w];
                                                sq=w;
                                        }
                                }
                                strcpy(nodes[sq].role, "master");
                                for(w=0; w<n; w++){//kanw slave tous nodes pou itan "none" kai einai geitones me ton
kainourio "master"
                                                        &&(!strcmp(nodes[w].role, "none")))){strcpy(nodes[w].role,
```

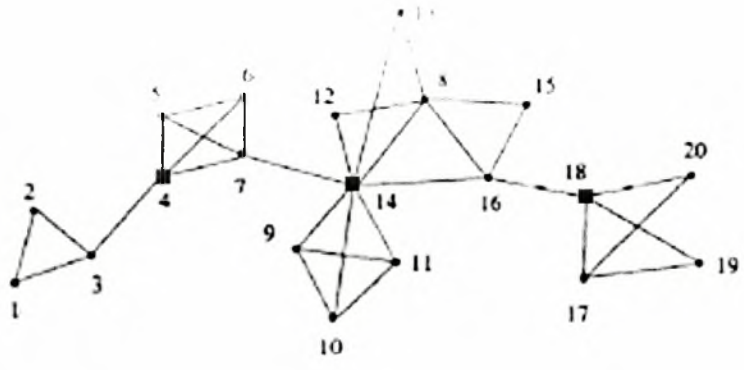
```
		}
	}

	[illegible]//gia tous kombous pou einai akoma "none" apofasizw an 8a ginoun master i slave

				[illegible](nodes[j].role, "master"))){
				t=1;
			}
		}
		if(t==1){strcpy(nodes[i].role, "slave");}//an exei master dipla ginetai slave
		else{strcpy(nodes[i].role, "master");}//an den exei master dipla ginetai master
	}
}

for(i=0; i<n; i++){
	t=0;
	[illegible]"master"))){
				[illegible]"slave")))[illegible]
		}
		if(t>7){//an exei perisoterous apo 7 slaves
			for(j=0; j<n; j++){
				s_q=0;
				s_w=0;
				if((a[i][j]==1)&&(!strcmp(nodes[j].role, "slave"))){
					for(w=0; w<n; w++){//ypologizw a8ristika ta NI twn 2-hop gia ka8e slave
tou master
						if(arayNI[w][j]>=0){
							s_q=s_q+arayNI[w][j];//einai to sunoliko NI to
					[illegible]//epilegw ka8e fora to megalutero NI pou uparxei
						s_w=s_q;
						sq=j;
					}
				}
			}
			strcpy(nodes[sq].role, "master");//kanw master auton pou eixe a8ristika to megalutero NI
		}
	}
}
```

3.4 Παράδειγμα εκτέλεσης σε ένα γράφημα 20 κόμβων

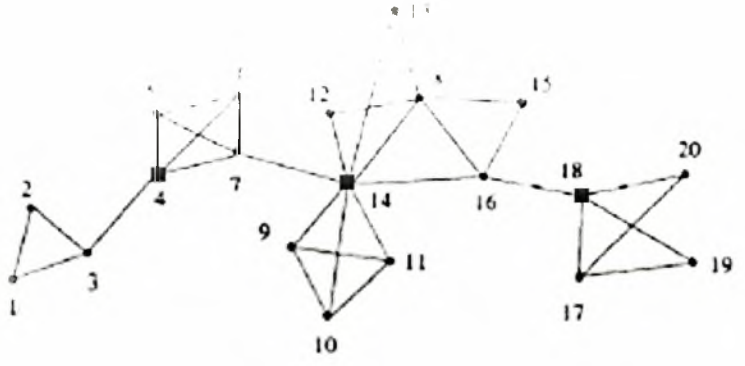
Θα εκτελέσουμε τον αλγόριθμό μας, στο προηγούμενο γράφημα, το οποίο αποτελείται από 20 κόμβους, όπου θέλουμε να ανακαλύψουμε, τα master και τα slave. Το αρχικό γράφημα απεικονίζεται ακριβώς από κάτω, και θεωρούμε ότι μόλις έχουμε ολοκληρώσει, τη φάση ανακάλυψης κόμβων.



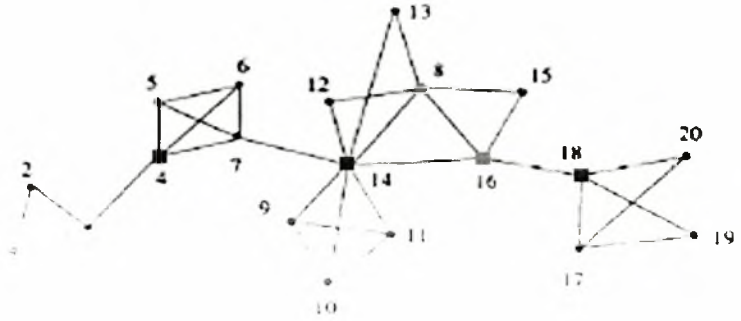
Θεωρώντας ότι έχουμε υπολογίσει τα NI για τις επιμέρους 2-hop γειτονιές όλων των κόμβων, προσπερνάμε τον πρώτο κύκλο (round 1). Ξεκινάμε λοιπόν με τον δεύτερο κύκλο (round 2), ανακαλύπτοντας τα αρχικά master, που εξηγήσαμε στο round 2 πώς ανακαλύπτονται. Έτσι οι αρχικοί master εμφανίζονται στο παρακάτω σχήμα, στους κόμβους 4, 14, 18 με το μπλε τετραγωνάκι.



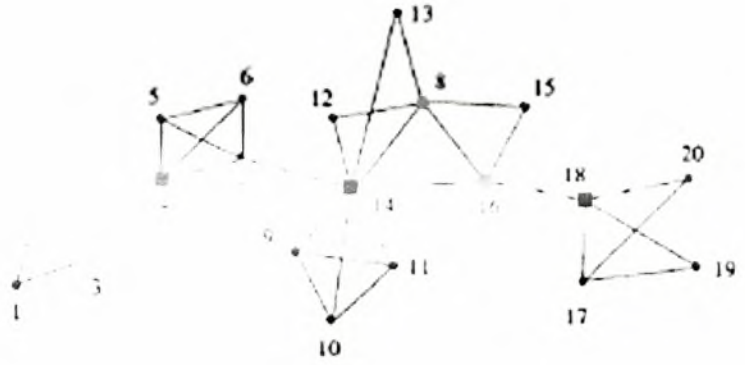
Αφού ολοκληρωσαμε την δεύτερη φάση, προχωράμε στο round 3 και 4, όπου θα ανακαλυψουμε τους standards slaves. Έτσι οι κόμβοι 1, 2, 5, 6, 9, 10, 11, 12, 13, 15, 19 και 20, είχαν το συνολικό άθροισμα των NI τους, μηδέν. Άρα γίνονται οι standards slaves, και εικονίζονται στο παρακάτω σχήμα με μπλε, βούλες.



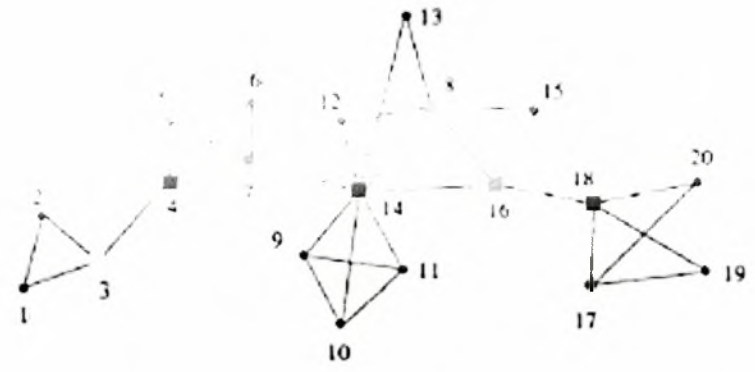
Στον πέμπτο και τελευταίο κύκλο (round 5), αρχικά, θα ορίσουμε ένα master, ο οποίος θα είναι, αυτός ο κόμβος που θα έχει το μεγαλύτερο NI αθροιστικά, στις γειτονιές που ανήκει. Ο κομβος αυτός, είναι ο 16 και συμβολίζεται στο παρακάτω σχήμα με κόκκινο τετραγωνάκι. Ο κόμβος 8, θα γίνει ο slave του, αφού, είναι ο μόνος γειτονικός που δεν είχε πάρει ρόλο ακόμη. Συμβολίζεται με κόκκινη βούλα.



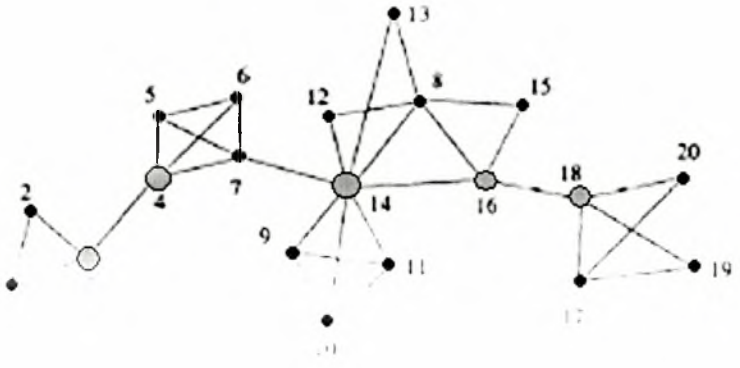
Έπειτα θα γίνει ο έλεγχος, εάν υπάρχει slave που δεν έχει στην 1-hop γειτονιά του κάποιο master. Ο 2 και ο 3 δεν έχουν κάποιο, και γι' αυτό, ο κόμβος 3 γίνεται master. Στο παρακάτω σχήμα εικονίζεται με πράσινο τετραγωνάκι. Αυτή την στιγμή, όλοι οι κόμβοι έχουν έναν master γείτονα.



Στην πορεία ελέγχω, αν υπάρχουν κόμβοι που εξακολουθούν να μην έχουν αποφασίσει το ρόλο τους. Είναι οι κόμβοι 7 και 17. Από τη στιγμή που έχουν κάποιο master σαν γείτονα, θα γίνουν οι slave του και θα προστεθούν στο piconet του. Με καφέ βούλες συμβολίζουμε αυτούς τους κόμβους στο παρακάτω σχήμα.



Έτσι, έχουμε ολοκληρώσει την εκτέλεση του αλγορίθμου μας, και κάθε κόμβος έχει αναλάβει το ρόλο του. Οι περιορισμοί (για τα Bluetooth ad-hoc δίκτυα) ισχύουν, και στο παρακάτω σχήμα έχουμε το συνολικό αποτέλεσμα. Οι κόμβοι 4, 14, 16 και 18, είναι masters και συμβολίζονται με κόκκινους κύκλους, και οι κομβοι 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17, 19 και 20, είναι slaves και εικονίζονται με μαυρες βούλες.

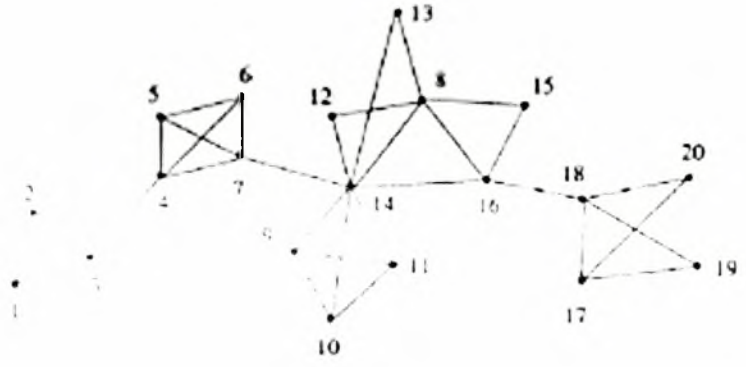


4. Πειραματική αποτίμηση του BCSF πρωτοκόλλου

4.1 Βαθμός density σταθερός

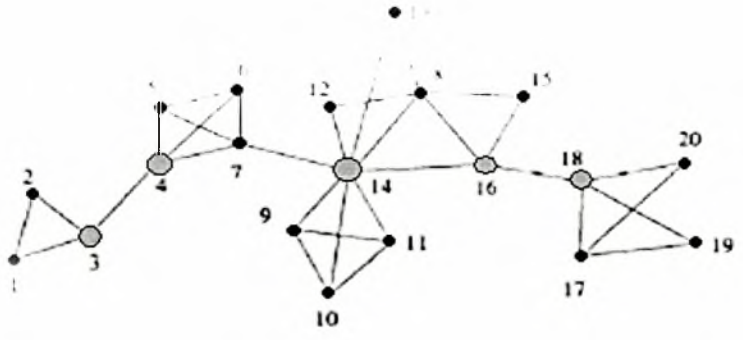
Στο σημείο αυτό, θα κάνουμε μια πειραματική αποτίμηση των όσων κάναμε. Σκοπός μας είναι να παρατηρήσουμε, σε πρώτη φάση, την σχέση που έχουν τα masters με τον αριθμό των κόμβων του δικτύου μας (αριθμός masters versus αριθμών κόμβων Bluetooth δικτύου). Με ένα σταθερό density των ακμών (density-> πυκνότητα των ακμών σε σχέση με τον αριθμό των κόμβων) θα μελετήσουμε τρία δίκτυα. Ένα με 20 κόμβους, ένα με 30, και ένα με 50 κόμβους. Θα εκτελέσουμε το BCSF πρωτοκολλο σε κάθε ένα από αυτά τα δίκτυα, και θα προσπαθήσουμε να εξαγουμε κάποια συμπεράσματα.

1)



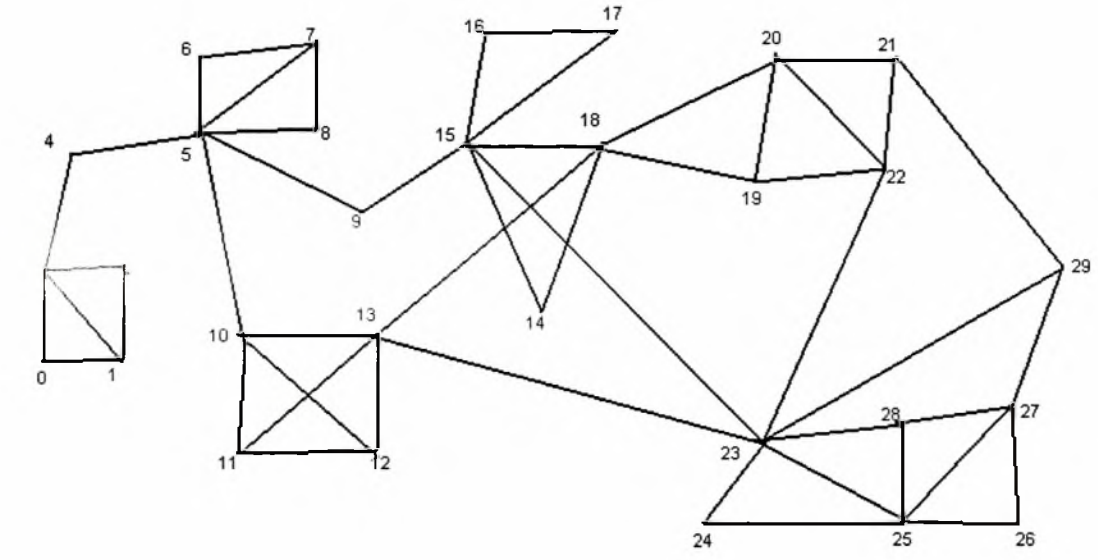
Εκτελούμε τον αλγόριθμο στο παραπάνω δίκτυο20 κόμβων, που φαίνεται στην εικόνα. Το αποτέλεσμα που παράγεται ακολουθεί ακριβώς μετά.

Από τους 20 κόμβους, οι κόμβοι 3, 4, 14, 16 και 18, έγιναν Masters. Οι υπόλοιποι κόμβοι, έγιναν slaves σε κάποιους από αυτούς.

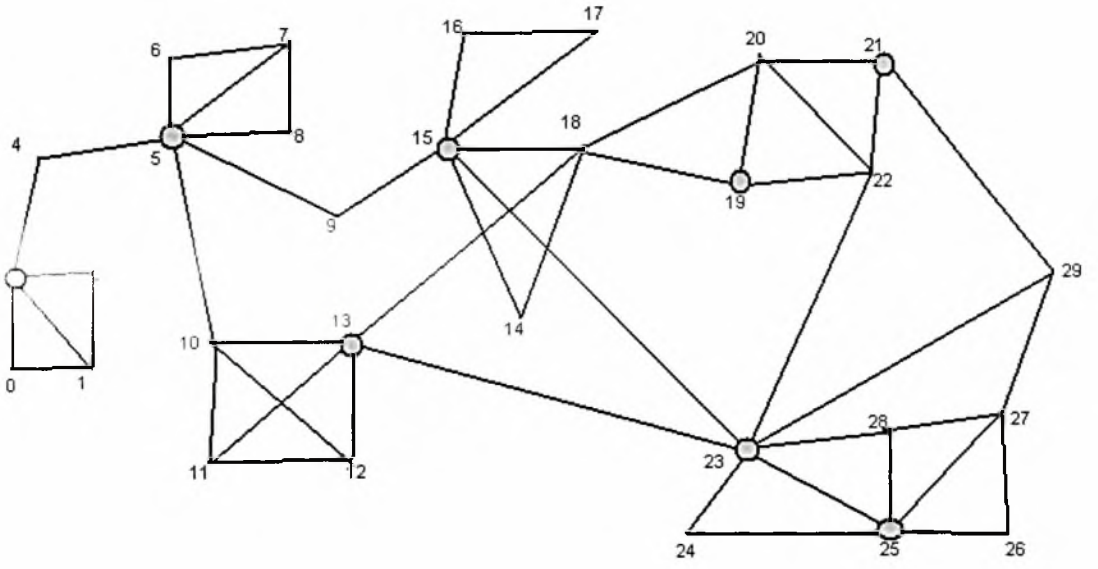


Παρατηρούμε έτσι, ότι σε ένα δίκτυο 20 κόμβων, οι 5 έγιναν masters και οι 15 slaves.

2)

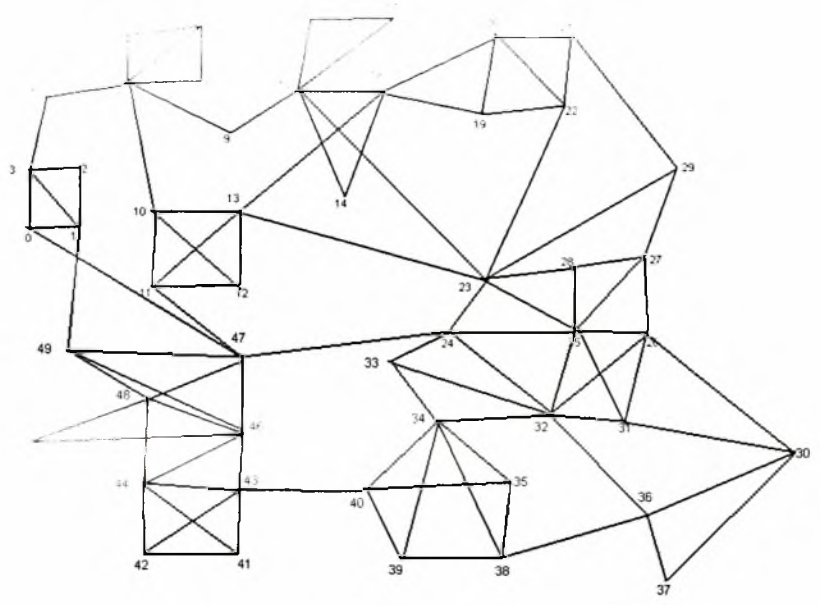


Το παραπάνω δίκτυο, είναι ένα δίκτυο 30 κόμβων. Αφού εκτελέσουμε τον αλγόριθμό μας στο δοθέν δίκτυο, θα παρατηρήσουμε ότι οι κόμβοι 3, 5, 13, 15, 19, 21, 23 και 25 έχουν γίνει masters. Όλοι οι υπόλοιποι, είναι slaves, σε κάποιους από αυτούς τους κόμβους. Τα αποτελέσματα του BCSF πρωτοκόλλου εμφανίζονται στο ακριβώς παρακάτω σχήμα.

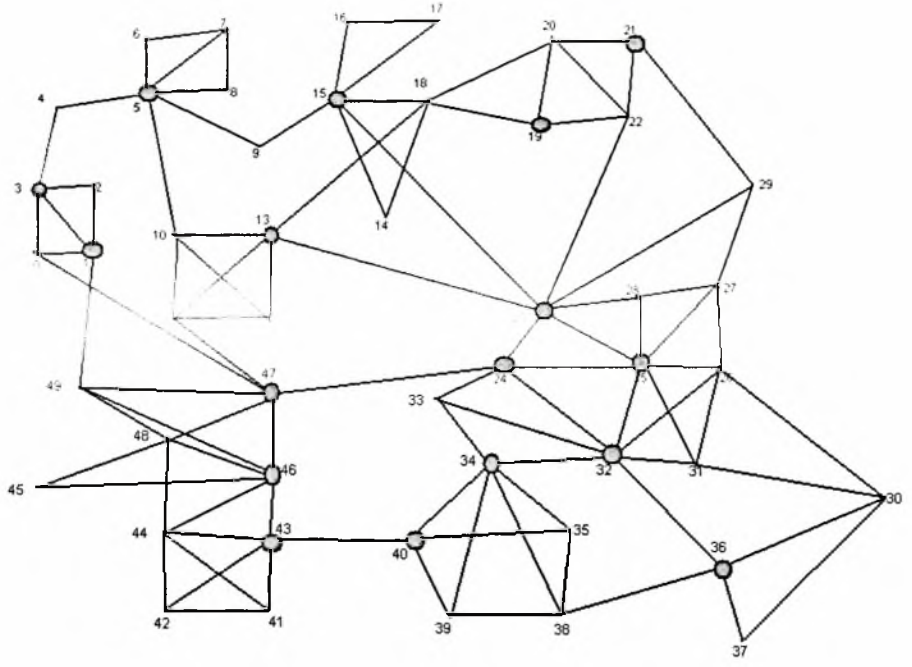


Παρατηρούμε δηλαδή, ότι σε ένα πλήθος 30 κόμβων, οι 8, έχουν γίνει masters, και οι 22 έχουν γίνει slaves κάποιους από αυτούς τους οκτώ.

3)



Το παραπάνω δίκτυο, είναι ένα δίκτυο Bluetooth 50 κόμβων. Θα εκτελέσουμε και σε αυτό το δίκτυο το BCSF αλγόριθμο, για να παρατηρήσουμε τη συμβαίνει σε ένα δίκτυο, όταν ο αριθμός των κόμβων είναι σχετικά μεγάλος. Εκτελώντας τον αλγόριθμο, τα αποτελέσματα που παίρνουμε, είναι: Έχουμε τους κόμβους, 1, 3, 5, 13, 15, 19, 21, 23, 24, 25, 32, 34, 36, 40, 43, 46 και 47, να είναι master κόμβοι του δικτύου και όλοι οι υπόλοιποι να γίνονται slaves σε κάποιους από αυτούς. Τα αποτελέσματα της εκτέλεσης του αλγορίθμου, εμφανίζονται στο επόμενο σχήμα.



Παρατηρούμε δηλαδή, ότι 17 από τους 50 κόμβους έγιναν masters, και 33 από τους 50 slaves σε κάποιους από αυτούς.

Σαν γενική αποτίμηση, μπορούμε να παρουσιάσουμε τον παρακάτω πίνακα που δείχνει την [illegible] αριθμού των κόμβων, με των αριθμό masters και slaves σε ένα δίκτυο όπου η πυκνότητα των ακμών (density) είναι σταθερή.

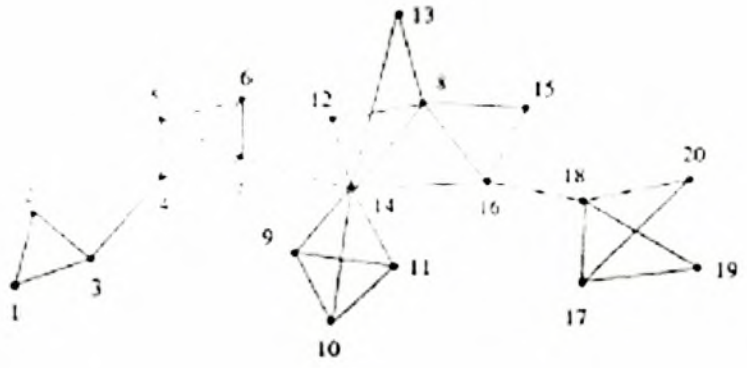
Αριθμός κόμβων δικτύου	Αριθμός masters	Αριθμός slaves	Αριθμός slaves /αριθμό masters
20	5	15	3.00
30	8	22	2.75
50	17	33	1.95

Άρα κατά μέσο όρο, η αναλογία slaves/masters, είναι 2,5.

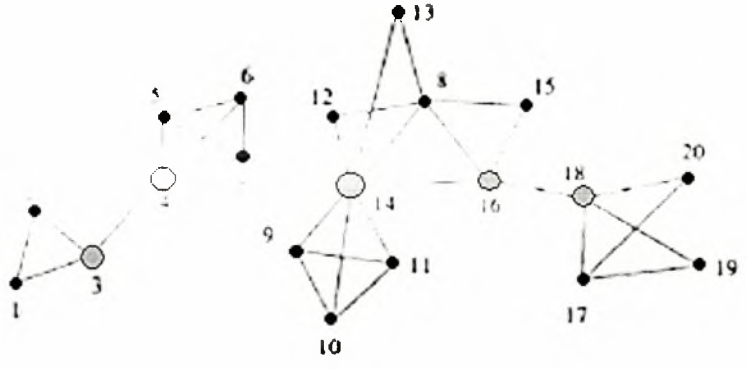
4.2 Αριθμός κόμβων σταθερός

Στην συνεχεια, θα μελετήσουμε την σχέση που έχουν τα masters-slaves με το density του δικτυου Bluetooth κόμβων. Κρατάμε σταθερό τον αριθμό των κόμβων και αυτό που αλλάζει είναι η πυκνότητα των ακμών τους, (αν θα έχει δηλαδή υψηλό ή όχι, density).

1)

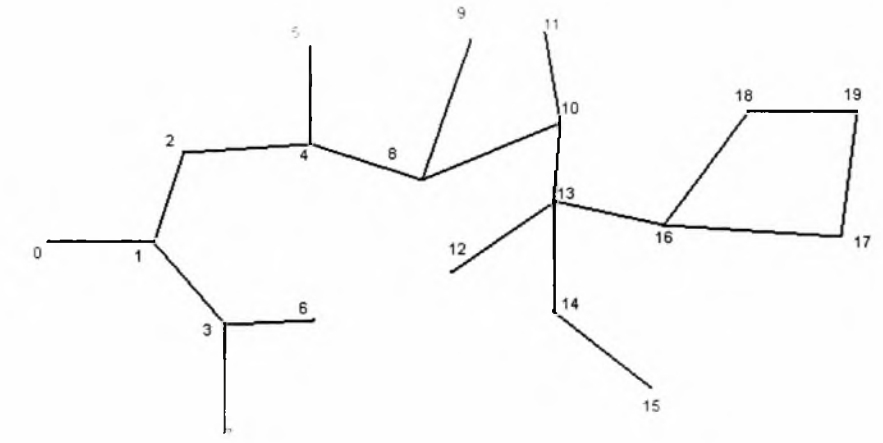


Αρχικά εκτελούμε τον αλγόριθμο σε ένα δίκτυο Bluetooth κόμβων, το οποίο το έχουμε ξανά χρησιμοποιήσει σε προηγούμενο πείραμά μας. Εδώ το density του δικτύου, είναι μετρίου βαθμού, οι κόμβοι μας είναι σταθεροί και 20. Τα αποτελέσματα που παράγονται. Στη φάση αυτή από την εκτέλεση του πρωτοκόλλου είναι: Από τους 20 κόμβους, οι κόμβοι 3, 4, 14, 16 και 18, έγιναν Masters. Οι υπολοιποι κομβοι, έγιναν slaves σε κάποιους από αυτούς. Η παρακάτω εικόνα δίνει την περιγραφή.

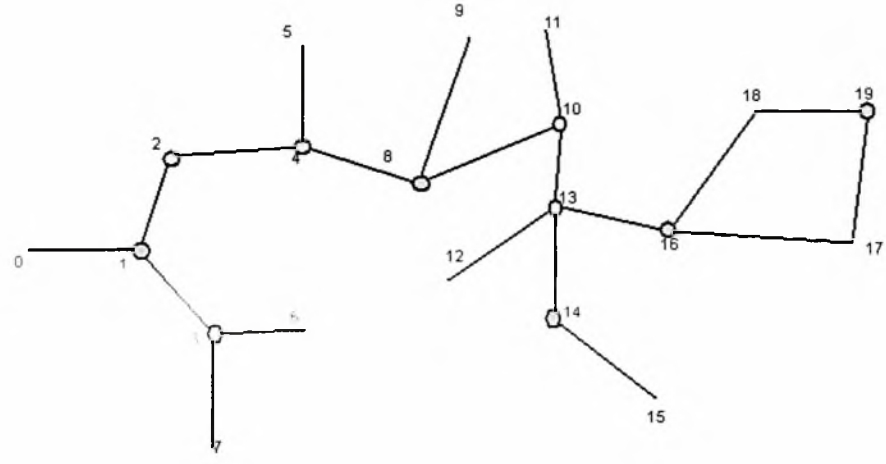


Παρατηρούμε ότι οι 5 από τους 20 κόμβους έγιναν Masters και οι 15 slaves.

2)

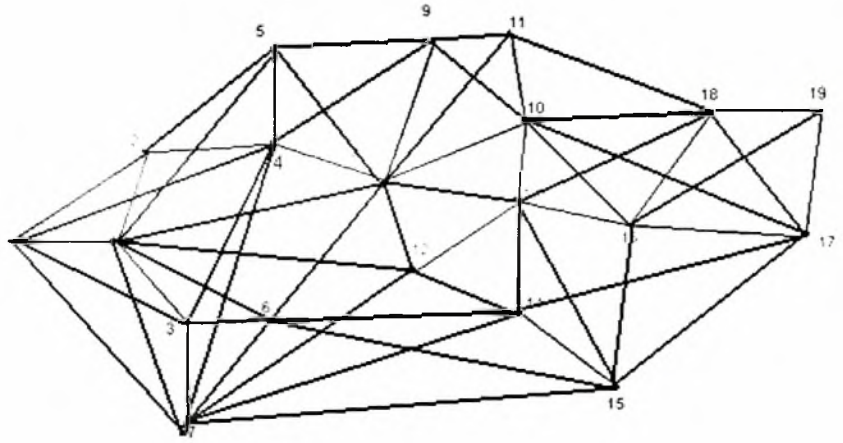


Το παραπάνω δίκτυο, είναι ένα δίκτυο Bluetooth κόμβων, αλλά με πολύ μικρό βαθμό density. Εκτελούμε τον αλγόριθμο και σε αυτό το δίκτυο των 20 κόμβων, και παίρνουμε τα αποτελέσματα. Οι κόμβοι 1, 2, 3, 4, 8, 10, 13, 14, 16 και 19, έγιναν masters. Οι υπόλοιποι κόμβοι έγιναν slaves σε κάποιους από αυτούς τους masters. Η παρακάτω εικόνα περιγράφει ακριβώς, τον ρόλο του κάθε κόμβου.

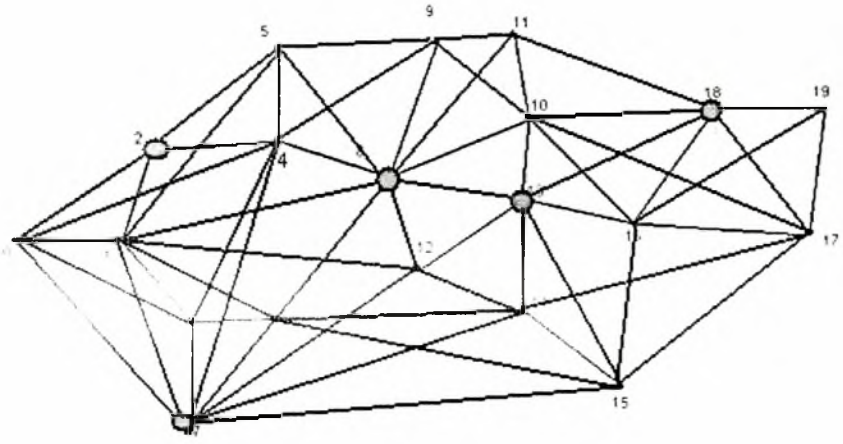


Παρατηρούμε ότι με τη ολοκλήρωση της εκτέλεσης του πρωτοκόλλου, οι 10 από τους 20 κόμβους του δικτύου, έγιναν masters, και οι υπόλοιποι 10 slaves. Ο αυξημένος αριθμός των masters οφείλεται, στο γεγονός ότι έχουμε πολύ μικρό βαθμό density και η επικοινωνία πάνω στο δίκτυο είναι πιο δύσκολη.

3)



Στο τελευταίο σχήμα, έχουμε πάλι ένα δίκτυο 20 κόμβων, αλλά στο παράδειγμα αυτό, το density, είναι πολύ υψηλό. Εκτελώντας τον κώδικα του πρωτοκόλλου, παίρνουμε το κάτωθι αποτέλεσμα που εικονίζεται στο παρακάτω σχήμα. Οι κόμβοι 2, 7, 8, 13 και 18, έχουν γίνει οι Master του δικτύου, και οι υπόλοιποι κόμβοι έχουν γίνει οι slaves.



Παρατηρούμε ότι στο δίκτυο αυτό, λόγο του ότι το density ήταν αρκετά υψηλό, η επικοινωνία ήταν πιο εύκολη. Άρα και ο μεγάλος αριθμός των Master δεν ήταν απαραίτητος. Από τους 20 κόμβους οι 5 έγιναν master και οι 15 slaves.

Σαν γενική αποτίμηση, μπορούμε να παρουσιάσουμε τον παρακάτω πίνακα που δείχνει την σχέση του αριθμού των κόμβων, με των αριθμό masters και slaves σε ένα δίκτυο όπου ο αριθμός των κόμβων είναι σταθερός.

Βαθμός density	masters	slaves	Αριθμός Slaves/αριθμός masters
μέτριος	5	15	3,00
χαμηλός	10	10	1,00
υψηλός	5	15	3,00

Άρα κατά μέσο όρο, η αναλογία slaves/masters, είναι 2,3.

5. Συμπεράσματα και Μελλοντική Έρευνα

Κινητρο για την διεξαγωγή της εργασίας αυτής, ήταν η κατασκευή ενός αλγορίθμου-πρωτοκόλλου, κατασκευής scatternet. Εξηγήθηκε στη διάρκεια αυτής της μελέτης, τι ακριβώς εννοούμε scatternet και τι piconet. Αφού κάναμε μια γενική εισαγωγή στους όρους ενός ad-hoc δικτύου, αρχίσαμε να μελετάμε τους ήδη υπάρχοντες αλγορίθμους κατασκευής Scatternet.

Μελετήσαμε τη λειτουργία του πρωτοκόλλου BlueNet, και τον τρόπο που χρησιμοποιείτε στο πρωτόκολλο αυτό, για την κατασκευή του Scatternet. Ερευνήσαμε το BlueMesh, το οποίο είναι ένα ακόμα πρωτόκολλο για ad-hoc δίκτυα. Και τέλος κάναμε μια καλή αναφορά στο πρωτόκολλο του BlueStar. Είδαμε πως λειτουργούν τα οι αλγόριθμοι αυτοί, και μελετήσαμε τις παραμέτρους που λαμβάνουν υπόψη τους για την κατασκευή του Scatternet. Πειραματιστήκαμε και στους τρεις, και εφαρμόσαμε από ένα πείραμα σε κάθε ένα από αυτούς, παρατηρώντας τα αποτελέσματα, και καταλήγοντας σε κάποια συμπεράσματα.

Στην πορεία ασχοληθήκαμε με το πρωτόκολλο BCSF. Το BCSF πρωτόκολλο, είναι ένας αλγόριθμος, που υπολογίζει την σημαντικότητα, ενός κόμβου σε ένα δίκτυο. Ο τρόπος υπολογισμού όμως του βαθμού σημαντικότητας, έχει να κάνει και με τους άλλους κόμβους του δικτύου, και συγκεκριμένα με την 2-hop γειτονία του εκάστοτε κόμβου. Εδώ υπεισέρχεται και ο όρος «θεωρία κοινωνικών δικτύων». Το αποτέλεσμα που παράγεται είναι ένα αποτέλεσμα που προέρχεται από τον βαθμό κοινωνικότητας κάθε κόμβου. Όλα αυτά υπολογίζονται σε χρόνο ανάλογο με τον αριθμό των κόμβων και των ακμών, έτσι, είναι ένα γρήγορο πρωτόκολλο για τον υπολογισμό του πίνακα Node Importance. Με τη χρήση αυτού του πίνακα, οδηγηθήκαμε, στη κατασκευή ενός πρωτοκόλλου για την δημιουργία ενός Scatternet δικτύου. Υπολογίζοντας το βαθμό σημαντικότητας κάθε κόμβου μπορούμε να πάρουμε αποφάσεις για τον ρόλο κάθε κόμβου σε ένα δίκτυο (αν θα είναι master ή slave). Έτσι κάθε κόμβος ξέρει το ρόλο του, ξέρει τους γείτονές του, και ξέρει το piconet του. Γνωρίζοντας όλα αυτά εύκολα οδηγούμαστε στη κατασκευή και του συνολικού scatternet που είναι ουσιαστικά η διασύνδεση πολλών piconets.

Τέλος, πειραματιστήκαμε πάνω στο πρωτόκολλο που κατασκευάσαμε, εκτελώντας διάφορα πειράματα και παρατηρώντας τα αποτελέσματα. Το συμπέρασμά μας είναι ότι έχει κατασκευαστεί ένα αποδοτικό πρωτόκολλο για τη δημιουργία Scatternet, που στηρίζεται στη κοινωνικότητα κάθε κόμβου. Το αποτέλεσμα υπολογίζεται σε χρόνο ανάλογο του αριθμού των κόμβων και των ακμών, οπότε αντιλαμβανόμαστε ότι είναι ένα γρήγορο πρωτόκολλο κατασκευής Scatternet.

Απομένει η απόφαση για τα gateways, αλλά είναι σχετικά εύκολη η επιλογή τους και το αφήσαμε ως μελλοντική εργασία. Αυτό που χρειαζόμαστε, είναι η επιλογή κατάλληλων gateways, για τη βελτιστοποίηση του πρωτοκόλλου μας, και γίνεται απλά με την αυτό-ανακήρηξη ενός κόμβου σε πύλη gateway μόλις καταλάβει ότι διασύνδεει δυο διαφορετικά piconets (δυο διαφορετικά masters δηλαδή).

6. Βιβλιογραφία

[1] Ad hoc & Sensor Networks. Theory and application. Carlos de Morais Codeio - Dharma Prakash Agrawal, pp 1-10

[2] Dimitrios Katsaros and Yannis Manolopoylos, The Geodesic Broadcast Scheme for Wireless Ad Hoc Networks Proceedings of IEEE Symporium on a World of Wireless and Mobile Multimedia Networks, pp. , 2006, pp 35-45

[3] Stefano Basagni, and Imrich Chlamtac, Configuring BlueStars: Multihop Scatternet Formation for Bluetooth Networks Chiara Petrioli, IEEE TRANSACTIONS ON COMPUTERS, VOL. 52, NO. 6,, pp 22 JUNE 2003

[4] STEFANO BASAGNI, MARCO CONTI, SILVIA GIORDANO, IVAN STOJMENOVIC , MOBILE AD HOC NETWORKING, pp 34-37

[5] STEFANO BASAGNI, RAFFAELE BRUNO, GABRIELE MAMBRINI, CHIARA PETRIOLI , Comparative Performance Evaluation of Scatternet Formation Protocols for Networks of Bluetooth Devices, Wireless Networks 10, 197–213, Kluwer Academic Publishers. Manufactured in The Netherlands, pp 20, 2004

[6] Wen-Zhan Song Xiang-Yang Li Yu Wang Weizhao Wang , Low Diameter and Self-routing Bluetooth Scatternet Dept. of Computer Science, Illinois Institute of Technology 10 W. 31st Street, pp 2, Chicago, IL 60616

[7] ERINA FERRO AND FRANCESCO POTORTI , BLUETOOTH AND WI-FI WIRELESS PROTOCOLS:A SURVEY AND A COMPARISON, IEEE Wireless Communications • pp 3, February 2005

[8] Daniele Miorandi a,*, Simone Merlin b, Arianna Trainito b, Andrea Zanella, On efficient configurations for Bluetooth scatternets, University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy Received 28 January 2005, pp 4.

[9] Csaba Kiss Kallo´ a,*, Carla-Fabiana Chiasserini b, Sewook Jung c, Mauro Brunato a, Mario Gerla , Hop count based optimization of Bluetooth scatternets, Dipartimento di Informatica e Telecomunicazioni, Universita` di Trento, Italy
b Dipartimento di Elettronica, Politecnico di Torino, Italy,
Department of Computer Science, University of California, Los Angeles, United States, pp 51, accepted 6 December 2005

[10] Zhifang Wang · Robert J. Thomas · ygmunt J. Haas , Performance comparison of Bluetooth scatternet formation protocols for multi-hop networks, ECE Cornell University, Ithaca, NY 14853, USA

[11] CHIARA PETRIOLI, STEFANO BASAGNI, IMRICH CHLAMTAC , BlueMesh: Degree-Constrained Multi-Hop Scatternet Formation for Bluetooth Networks, Kluwer Academic Publishers. Manufactured in The Netherlands, pp 18, 2004

[12] Rajarshi Roy, Mukesh Kumar, Navin K. Sharma, and Shamik Sural, Bottom-Up Construction of Bluetooth Topology under a Traffic-Aware Scheduling Scheme, Member, IEEE, IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 6, NO. 1, pp 4-5, JANUARY 2007

[13] Jang-Ping Sheu, Senior Member, IEEE, Kuei-Ping Shih, Member, IEEE Computer Society, Shin-Chih Tu, and Chao-Hsun Cheng, A Traffic-Aware Scheduling for Bluetooth Scatternets, IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 5, NO. 7,pp 40, JULY 2006

[14] Metın Tekkalmaz, Hasan So¨zer, and Ibrahim Korpeoglu, Member, IEEE, Distributed Construction and Maintenance of Bandwidth and Energy Efficient Bluetooth Scatternets, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 17, NO. 9, pp 35, SEPTEMBER 2006