Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Διπλωματική Εργασία

Ανάπτυξη και Πειραματισμός Αλγορίθμων Αναζήτησης Πόρων σε Διομότιμα Συστήματα (Peer-to-Peer networks)

Παπαδάκης Νίκος

Τσάλλας Στέλιος

Νοέμβριος 2007,
Ηράκλειο

Περιεχόμενα

Περίληψη

*Τα συστήματα **Peer-to-Peer (P2P)** είναι συστήματα δικτύων, στα οποία οι κόμβοι συμμετέχουν ισότιμα στο δίκτυο, για να επιτευχθεί η επικοινωνία μεταξύ τους στο δίκτυο. Τα συστήματα αυτά προσφέρουν στους χρήστες τους τη δυνατότητα να μοιραστούν αρχεία και δεδομένα, και στις περισσότερες περιπτώσεις οι χρήστες μοιράζονται αρχεία πολυμέσων.*

*Στην παρούσα έρευνα θα περιγράψουμε δομημένα και αδόμητα συστήματα καθώς και ορισμένους αλγόριθμους αναζήτησης πάνω σε αυτά. Από τα δομημένα συστήματα θα περιγράψουμε το **CAN**, το **Chord**,* το **Pastry** και το **Kademlia** *τα οποία χρησιμοποιούν την τεχνική **Data Hash Tables (DHT)** ενώ από τα αδόμητα θα αναλύσουμε το **Gnutell**, το **Napster** και το **Kazaa** και θα εξετασουμε διάφορους αλγόριθμους αναζήτησης που βασίζονται σε διάφορες παραλλαγές της τεχνικής της πλημμύρα **(flooding)**.*

Abstract

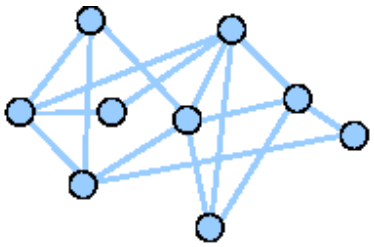
***Peer to peer (P2P)** are systems of networks, in which the nodes participate equivalently, in order to communicate with each other in the network. These systems offer their users the possibility to share files and data, and in most cases the users are sharing multimedia files.*

*In the present research we will describe structured and unstructured systems as well as certain algorithms of data search on them. From the structured systems we will describe **CAN ,Chord, Pastry and Kademlia** systems that use the **Data Hash Tables (DHT)** technique ,while from unstructured systems we will describe **Gnutella, Napster** and **Kazaa** and we will examine variation of **flooding** techniques.*

1.Εισαγωγή

Τα συστήματα ομότιμων κόμβων (**peer-to-peer**) είναι κατανεμημένα δίκτυα στα οποία οι κόμβοι (**nodes**) συνδέονται μεταξύ τους με διάφορους τρόπους και τεχνικές. Το χαρακτηριστικό των συστημάτων αυτών είναι ότι όλοι οι κόμβοι είναι ομότιμοι, δηλαδή έχουν τις ίδιες δυνατότητες, κρατούν ίδιο μέγεθος πληροφορίας και έχουν τις ίδιες ευθύνες. Στα αρχικά συστήματα ομότιμων κόμβων (**Napster**) υπήρχε ένας κεντρικός κόμβος ο οποίος κράταγε περισσότερη πληροφορία από τους άλλους και ήταν υπεύθυνος για την αναζήτηση αντικειμένων καθώς και για την εισαγωγή ή διαγραφή ενός κόμβου. Αργότερα αναπτύχθηκαν τα πλήρως κατανεμημένα δίκτυα **P2P** (π.χ. **Gnutella**) τα οποία, όμως, για μεγάλο αριθμό κόμβων παρουσίαζαν προβλήματα.

Σήμερα, παρουσιάζονται νέα συστήματα ομότιμων κόμβων που προσπαθούν να λύσουν τέτοια προβλήματα και νέες τεχνικές αναζήτησης σε υπάρχοντα συστήματα. Στην παρούσα έρευνα θα παρουσιαστούν τα πιο γνωστά συστήματα καθώς και αλγόριθμοι που βοηθούν στην αναζήτηση, εισαγωγή και διαγραφή κόμβων ή δεδομένων σε τέτοια συστήματα.



Σχήμα 1: P2P δίκτυο

Αυτό που μας ενδιαφέρει στα **P2P** συστήματα είναι να γίνονται όσο το δυνατόν πιο αποδοτικά οι λειτουργίες αναζήτησης, εισαγωγής και διαγραφής ενός κόμβου ή δεδομένου. Όσον αφορά την αναζήτηση δεδομένων, μας ενδιαφέρει να βρούμε τρόπους, οι οποίοι θα μας δίνουν αποτέλεσμα σε μικρό χρόνο και με ανταλλαγή, όσο το δυνατόν, λιγότερων μηνυμάτων. Για την εισαγωγή και διαγραφή κόμβων, μας ενδιαφέρει να βρούμε τρόπους ώστε η εισαγωγή ή διαγραφή ενός κόμβου να επηρεάζει όσο το δυνατόν λιγότερους κόμβους στο δίκτυο. Επίσης, σε μερικές περιπτώσεις μας ενδιαφέρει και η αντιγραφή δεδομένων σε πολλούς κόμβους έτσι ώστε να γίνονται πιο προσιτά.

Ανάλογα με τον τρόπο με τον οποίο συνδέονται οι κόμβοι μεταξύ τους τα συστήματα **P2P** χωρίζονται σε δομημένα και μη δομημένα συστήματα.

Στα δομημένα οι κόμβοι συνδέονται με κάποιο συγκεκριμένο τρόπο έτσι ώστε να δημιουργήσουν μια δομή. Στα μη δομημένα συστήματα οι κόμβοι συνδέονται τυχαία μεταξύ τους.

Επίσης, σε ξεχωριστή κατηγορία εντάσσονται τα συστήματα τα οποία κατηγοριοποιούν τους κόμβους με βάση το περιεχόμενό τους.

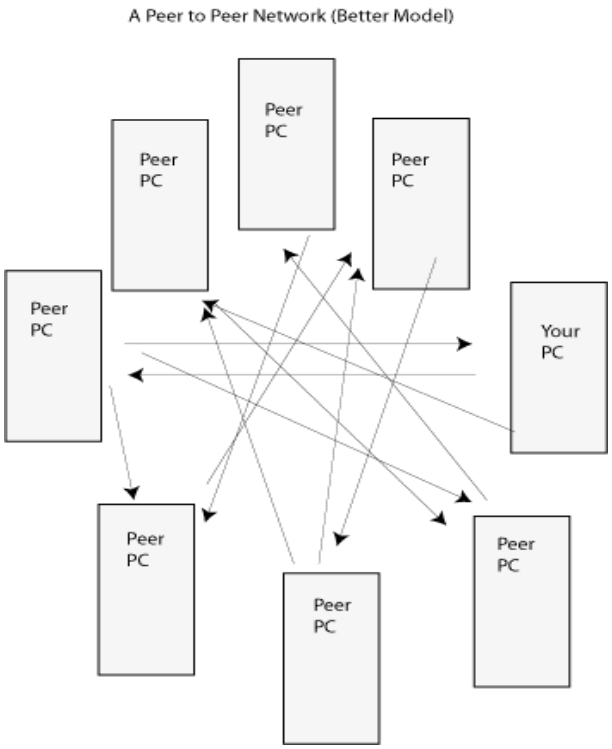
Στην επόμενη παράγραφο θα αναφερθούμε γενικά για τα **Peer** συστήματα καθώς και για κάποια χαρακτηριστικά των ομότιμων κόμβων. Στη συνέχεια θα μελετήσουμε δυο από τα πιο γνωστά δομημένα συστήματα (**CAN** και **Chord**).

Στο $5^{ο}$ Κεφάλαιο θα αναφερθούμε στα αδόμητα συστήματα και στην συνέχεια θα αναλύσουμε κάποιους αλγορίθμους αναζήτησης. Τέλος, θα παρουσιάσουμε ορισμένες γραφικές παραστάσεις καθώς και θα αναφέρουμε μερικά συμπεράσματα από τη μελέτη των συστημάτων που περιγράφηκαν.

2. Γενικά για τα Peer-to-Peer Συστήματα

Ένα **Peer-to-Peer** (**P2P**) δίκτυο υπολογιστών εκμεταλλεύεται τη διαφορετική συνδετικότητα μεταξύ εκείνων των κόμβων που συμμετέχουν σε ένα δίκτυο και του συσσωρευτικού εύρους ζώνης των συμμετεχώντων δικτύων παρά τους συμβατικούς συγκεντρωμένους πόρους όπου ένας σχετικά χαμηλός αριθμός κεντρικών υπολογιστών (**servers**) παρέχει την αξία των πυρήνων σε μια υπηρεσία ή μια εφαρμογή.

Τα **Peer-to-Peer** (Σχ. 2.1) δίκτυα χρησιμοποιούνται χαρακτηριστικά για τη σύνδεση των κόμβων μέσω των κατά ένα μεγάλο μέρος ειδικών συνδέσεων. Τέτοια δίκτυα είναι χρήσιμα για πολλούς λόγους. Μοιράζουν τα αρχεία που περιέχουν τους ήχους, τα βίντεο, τα δεδομένα ή οτιδήποτε είναι σε ψηφιακή μορφή κάτι που είναι πολύ κοινό, καθώς και δεδομένα σε πραγματικό χρόνο, όπως η τηλεφωνική ροή, χρησιμοποιώντας και αυτά την **P2P** τεχνολογία.



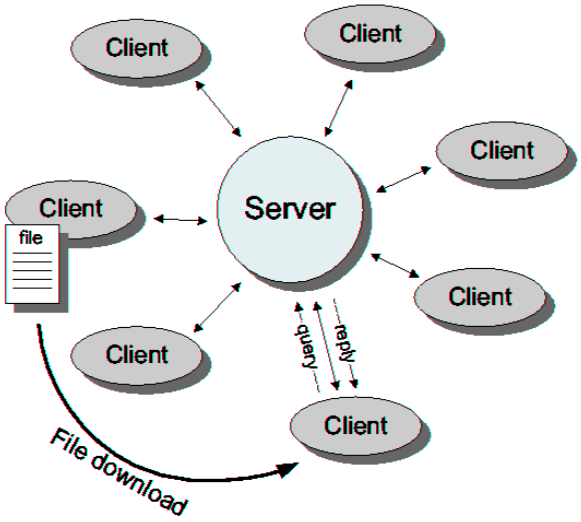
Εικόνα 2.1: Παράδειγμα δικτύου Peer to Peer

Ένα απλό **Peer-to-Peer** δίκτυο δεν έχει την έννοια των πελατών (**clients**) ή των κεντρικών υπολογιστών (**servers**), αλλά είναι ίσο μόνο με τους όμοιους κόμβους που

λειτουργούν ταυτόχρονα όπως τους "**clients**" και "**servers**" στους άλλους κόμβους στο δίκτυο.

Αυτό το μοντέλο δικτύων διαφέρει από το μοντέλο πελάτη-εξυπηρετητή (Σχ. 2.2) όπου η επικοινωνία είναι συνήθως "σε" και "από" έναν κεντρικό υπολογιστή.

Ένα χαρακτηριστικό παράδειγμα για μια μη **Peer-to-Peer** μεταφορά αρχείων είναι ένας κεντρικός υπολογιστής (**server**) **FTP** όπου τα προγράμματα πελατών και κεντρικών υπολογιστών είναι αρκετά ευδιάκριτα, και οι πελάτες αρχίζουν το **download** και **upload** και οι κεντρικοί υπολογιστές αντιδρούν και ικανοποιούν αυτά τα αιτήματα.



Εικόνα 2.2: Παράδειγμα μοντέλου Client-server

Το πρώτο **Peer-to-Peer** δίκτυο σε διαδεδομένη χρήση ήταν το σύστημα **USENET**, στο οποίο οι κόμβοι επικοινωνούσαν ο ένας με τον άλλο για να διαδώσουν τα άρθρα ειδήσεων **USENET** πέρα από το δίκτυο **USENET**. Ιδιαίτερα τις πρώτες ημέρες του **USENET**, χρησιμοποιήθηκε το **UUCP** για να επεκταθεί ακόμη και πέρα από το Διαδίκτυο. Εντούτοις, το **USENET** χρησιμοποιήθηκε επίσης και σε μια μορφή πελάτη-εξυπηρετητή όταν οι μεμονωμένοι χρήστες είχαν πρόσβαση σε έναν τοπικό κεντρικό υπολογιστή ειδήσεων για να διαβάζουν τα άρθρα. Η ίδια εκτίμηση ισχύει για το ηλεκτρονικό ταχυδρομείο **SMTP** υπό την έννοια ότι στέλνοντας ενα e-mail είναι ένα **Peer-to-Peer** δίκτυο.

Μερικά δίκτυα και κανάλια όπως **Napster**, **OpenNAP** και **IRC** χρησιμοποιούν τη δομή πελάτη-εξυπηρετητή για μερικούς στόχους (π.χ. ψάχνοντας) και μια δομή **Peer-to-Peer** για άλλους (σκοπούς).

Δίκτυα όπως το **Gnutella** ή κάποιο ελεύθερο δίκτυο χρησιμοποιούν τη δομή **Peer-to-Peer** για όλους τους σκοπούς τους, και μερικές φορές αναφέρονται ως αληθινά **Peer-to-Peer** δίκτυα, αν και το **Gnutella** διευκολύνεται πολύ από τους κεντρικούς

υπολογιστές καταλόγου (**directory servers**) που ενημερώνουν τους κόμβους για τις διευθύνσεις δικτύων των άλλων κόμβων.

Η αρχιτεκτονική **Peer-to-Peer** ενσωματώνει μια από τις βασικές τεχνικές έννοιες του Διαδικτύου (**Internet**). Πιό πρόσφατα, η έννοια έχει επιτύχει την αναγνώριση στο ευρύ κοινό στα πλαίσια της απουσίας κεντρικών συντάσσοντας **servers** στις αρχιτεκτονικές που χρησιμοποιούνται για την ανταλλαγή αρχείων.

Τέλος η έννοια **Peer-to-Peer** εξελίσσεται όλο και περισσότερο στα κατανεμημένα δίκτυα, όχι μόνο από υπολογιστή σε υπολογιστή, αλλά και από άνθρωπο σε άνθρωπο.

3. Χαρακτηριστικά Ομότιμων Κόμβων

Θα δούμε κάποιες ιδιότητες, οι οποίες είναι αντιπροσωπευτικές και δίνουν μια πιο πλήρη εικόνα για τα συστήματα αυτά και το πώς λειτουργούν.

Δεν υπάρχει κεντρικός συντονισμός: Κάθε κόμβος λειτουργεί σε σχέση με την επικοινωνία που αναπτύσσει με τους γείτονές του, καθώς μόνο αυτούς γνωρίζει. Δεν υπάρχει τρόπος να μιλήσει κάποιος άμεσα σε όλο το δίκτυο και δεν γνωρίζει τι γίνεται σε μια περιοχή του δικτύου μακριά του. Κάθε χρήστης λειτουργεί ως πελάτης (**client**) αλλά και ως εξυπηρετητής (**server**).

Κάθε κόμβος είναι αυτόνομος: Κάθε κόμβος λειτουργεί ανεξάρτητα από τους υπόλοιπους. Μπορεί να μπει στο δίκτυο και να αποχωρήσει από το δίκτυο όποια στιγμή θελήσει, χωρίς να πρέπει να περιμένει κάποιον άλλον κόμβο (π.χ. ολοκλήρωση μεταφοράς αρχείου).

Κάθε κόμβος θεωρείται αναξιόπιστος: Κάθε κόμβος μπορεί να μπει και να αφήσει το δίκτυο οποιαδήποτε στιγμή, και αυτό τον καθιστά αναξιόπιστο. Εφόσον όλοι οι κόμβοι έχουν την ίδια συμπεριφορά, αυτή η ιδιότητα είναι αρκετά σημαντική για τον τρόπο με τον οποίο σχεδιάζεται ένα τέτοιο σύστημα.

Υπάρχουν τρεις γενικές αρχιτεκτονικές, οι οποίες χρησιμοποιήθηκαν από κάποια συστήματα.

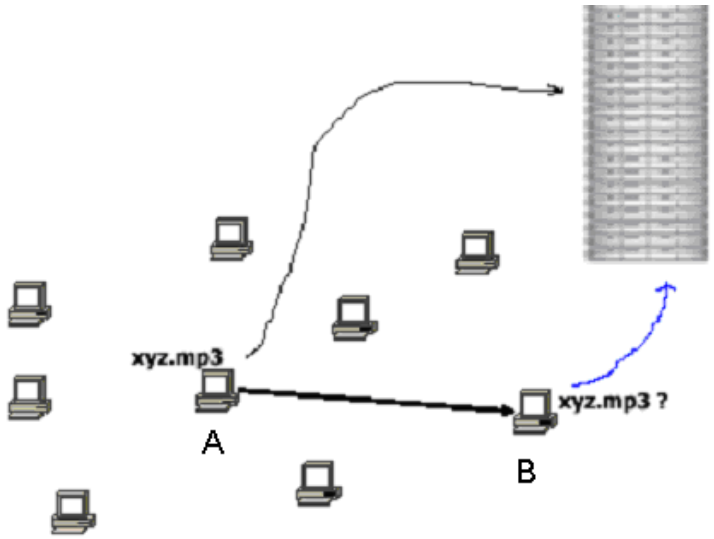
- Κεντρικοποιημένη αρχιτεκτονική: Αυτή την αρχιτεκτονική είχε το **Napster**. Οι χρήστες του συνδεόντουσαν σε ένα κεντρικό υπολογιστή, και όλες οι αναζητήσεις αρχείων γινόταν από εκεί. Ο υπολογιστής αυτός γνώριζε όσους ήταν συνδεδεμένοι, καθώς και τα αρχεία που μοιραζόντουσαν όλοι οι χρήστες.
- Κατανεμημένη αρχιτεκτονική: Αυτήν την αρχιτεκτονική υιοθέτησε το σύστημα **Gnutella** και το **Freenet** (και όχι μόνο αυτοί). Εδώ δεν υπάρχει κανένας κεντρικός συντονισμός των χρηστών. Όλη η κίνηση στο δίκτυο γινόταν με την επικοινωνία μεταξύ των κόμβων.
- Ιεραρχική αρχιτεκτονική: Η τρίτη αρχιτεκτονική είναι ένας συνδυασμός των δύο παραπάνω. Υπάρχουν κάποιοι κόμβοι που είναι περισσότερο σημαντικοί από τους υπόλοιπους (υπερκόμβοι). Το **Kazaa** χρησιμοποιεί υπερκόμβους στο δίκτυο του.

4. Κεντρικοποιημένα P2P Συστήματα

Ο όρος των κεντρικοποιημένων συστημάτων αφορά στην πρώτη από τις δύο κατηγορίες των **P2P** δικτύων. Αυτού του είδους τα δίκτυα διατηρούν έναν **κατάλογο-ευρετήριο** σε μια κεντρική τοποθεσία ο οποίος κρατά πληροφορίες για όλα τα δεδομένα που βρίσκονται αποθηκευμένα στους κόμβους. Ο κατάλογος-ευρετήριο δημιουργείται με δύο τρόπους: είτε με τη συνεργασία των κόμβων που του παρέχουν τακτικά μια λίστα με τα δεδομένα που προσφέρουν στο δίκτυο, είτε με αναζήτηση στο δίκτυο όπως συμβαίνει με μια μηχανή αναζήτησης στο **Internet**. Στο Σχήμα 4.1 φαίνεται ο πρώτος από τους παραπάνω δύο τρόπους.

Έτσι όταν ο κόμβος Α που προσφέρει το δεδομένο "xyz.mp3" εισάγεται στο δίκτυο ενημερώνει τον κατάλογο-ευρετήριο με τα δεδομένα που προσφέρει. Ένας χρήστης-κόμβος που θέλει να βρει ένα συγκεκριμένο δεδομένο στο δίκτυο, για παράδειγμα στο Σχήμα 4.1 ο κόμβος Β που επιθυμεί το "xyz.mp3", κάνει μια ερώτηση στο κεντρικό ευρετήριο κι αυτό μετά από μία τοπική αναζήτηση στις εγγραφές του επιστρέφει ως απάντηση στον κόμβο Β την ακριβή τοποθεσία του δεδομένου μέσα στο δίκτυο.

Τέλος, ο χρήστης-κόμβος Β ζητά απευθείας από τον κόμβο Α που διαθέτει το δεδομένο να του το στείλει. Έτσι παρόλο που το **Napster**, ένα από τα συστήματα αυτής της κατηγορίας, αποτελεί ένα μοντέλο ομότιμων κόμβων για τη μεταφορά των δεδομένων στο δίκτυο, η διαδικασία εντοπισμού κάποιου δεδομένου είναι κεντρικοποιημένη.



Σχήμα 4.1: Διαδικασία αναζήτησης

Το **Napster** βοήθησε πολύ ώστε τα **P2P** δίκτυα να γίνουν ευρέως γνωστά και να αποτελέσουν μία από τις πιο γρήγορα εξελισσόμενες εφαρμογές του διαδικτύου, νομικά προβλήματα όμως ήταν η αιτία να εγκαταλειφθούν όλα τα δίκτυα αυτής της κατηγορίας.

Στα πλεονεκτήματά τους συμπεριλαμβάνονται η αποδοτικότητα του δικτύου καθώς μόνο ένα μήνυμα είναι απαραίτητο για να απαντηθεί η ερώτηση ενός κόμβου, η ικανότητα να βρεθούν εύκολα "σπάνια" δεδομένα και η υποστήριξη των "**partial-match**" ερωτήσεων (δηλαδή ερωτήσεις που περιέχουν ορθογραφικά λάθη ή περιλαμβάνουν ένα υποσύνολο από λέξεις κλειδιά).

Παρόλα αυτά ένα κεντρικοποιημένο σύστημα είναι πολύ εύκολο να καταρρεύσει αν δεχτεί επίθεση ο κεντρικός κόμβος και επίσης αρκετά δύσκολο να διατηρείται πάντα ενημερωμένο το κεντρικό ευρετήριο.

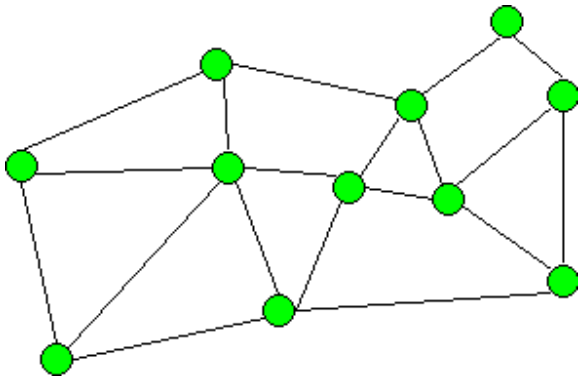
5. Μη Κεντρικοποιημένα P2P Συστήματα

Αποτελούν τη δεύτερη από τις δύο κατηγορίες των **P2P** συστημάτων η οποία στη συνέχεια διασπάται στις υποκατηγορίες των δομημένων και μη δομημένων δικτύων. Αυτό που διαφοροποιεί τα μη κεντρικοποιημένα **P2P** δίκτυα από αυτά της προηγούμενης ενότητας είναι ότι δε διαθέτουν κάποιον κεντρικό **κατάλογο-ευρετήριο**.

5.1 Αδόμητα Συστήματα

Τα δομημένα συστήματα που εξετάσαμε πιο πάνω, έχουν το πλεονέκτημα ότι οι κόμβοι τους ενώνονται με συγκεκριμένο τρόπο οπότε δημιουργείται ένας γράφος. Ο τρόπος με τον οποίο συνδέονται έχει επιλεχθεί έτσι, ώστε η αναζήτηση, η εισαγωγή και η αποχώρηση ενός κόμβου, να γίνονται αποδοτικότερα.

Ωστόσο, είδαμε ότι κατά την εισαγωγή ή την αποτυχία ενός κόμβου, χρειάζεται να ενημερωθεί ένας αρκετά μεγάλος αριθμός από κόμβους προκειμένου το σύστημα να λειτουργήσει σωστά. Το πρόβλημα αυτό, λύνουν τα αδόμητα συστήματα, στα οποία η εισαγωγή και αποχώρηση η αποτυχία γίνεται σε ένα βήμα, χωρίς να επηρεάζονται οι υπόλοιποι κόμβοι. Στο Σχ. 5.2 φαίνεται ένα παράδειγμα αδόμητου συστήματος.



Σχήμα 5.2: Παράδειγμα αδόμητου συστήματος. Οι πράσινοι κύκλοι συμβολίζουν τους κόμβους

Παρατηρούμε ότι οι κόμβοι είναι συνδεδεμένοι τυχαία, χωρίς να υπακούν σε κάποια συγκεκριμένη δομή. Στα αδόμητα συστήματα οι κόμβοι συνδέονται τυχαία μεταξύ τους χωρίς να είναι απαραίτητο να διατηρηθεί κάποια δομή.

Έτσι, λοιπόν, δεν μπορούμε να εκμεταλλευτούμε τις ιδιότητες της δομής ή τον τρόπο σύνδεσης των κόμβων. Αντίθετα, η προσπάθεια για αποδοτική αναζήτηση στηρίζεται στην εύρεση κατάλληλων αλγορίθμων. Έτσι στα αδόμητα συστήματα επικεντρωνόμαστε στους μηχανισμούς αναζήτησης. Μερικούς από αυτούς τους μηχανισμούς εξετάζουμε πιο κάτω.

5.1.1 Παραδείγματα Συστημάτων

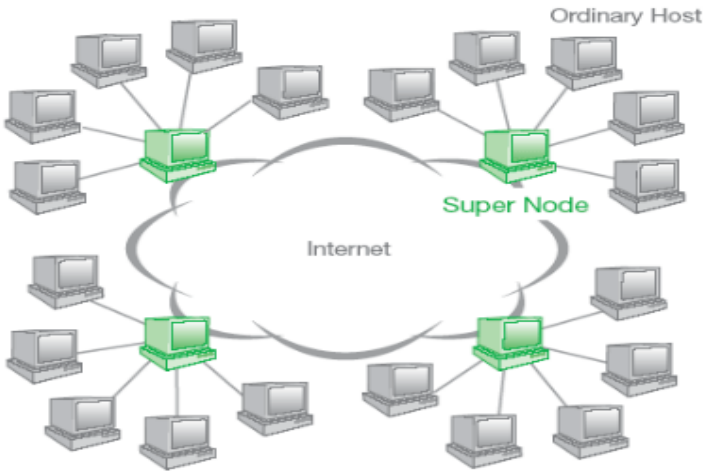
Napster [1]

Το πρώτο σύστημα ομότιμων κόμβων ήταν το **Napster**, το οποίο όμως, δεν ήταν καθαρά κατανεμημένο, αφού είχε μια κεντρικοποιημένη δομή. Υπήρχε ένας κεντρικός εξυπηρετητής, ο οποίος γνώριζε τα αρχεία που μοιράζονταν οι συνδεδεμένοι στο **Napster** χρήστες. Όταν ένας νέος χρήστης έμπαινε στο δίκτυο, επικοινωνούσε με τον κεντρικό εξυπηρετητή, και τον ενημέρωνε για τα διαθέσιμα αρχεία του. Όταν ένας χρήστης έκανε μια αναζήτηση, ρωτούσε μόνο τον εξυπηρετητή. Στη συνέχεια οι χρήστες δημιουργούσαν μία απευθείας σύνδεση μεταξύ τους για να πραγματοποιηθεί η μεταφορά του αρχείου. Το πρόβλημα με το **Napster** ήταν ότι σε περίπτωση αποτυχίας του κεντρικού εξυπηρετητή, όλο το δίκτυο ήταν ανενεργό.

Kazaa [2]

Το **Kazaa** μοιάζει με το **Gnutella** δεδομένου ότι δεν χρησιμοποιεί κάποιο κεντρικό υπολογιστή (**server**) για να εντοπίσει το περιεχομένο που αναζητάται. Αντίθετα από το **Gnutella**, δεν είναι όλοι οι κόμβοι ισότιμοι. Το **Kazaa** έχει δυο κατηγορίες κόμβων, τους συνηθισμένους κόμβους (**Ordinary Nodes** , **ONs**) και τους υπερκόμβους (**Super Nodes** , **SNs**). Οι ισχυρότεροι κόμβοι είναι οι **SNs** και έχουν μεγαλύτερες ευθύνες. Όπως φαίνεται παρακάτω στο Σχ. 5.3, κάθε κόμβος ON συνδέεται με έναν SN. Όταν ο χρήστης ενεργοποιεί το πρόγραμμα **Kazaa**, ο κόμβος ON εγκαθιστά μια σύνδεση TCP με έναν από τους SN και έπειτα φορτώνει στον SN κάποια δεδομένα (**metadata**) για τα αρχεία που μοιράζεται.

Αυτό επιτρέπει στους SN να διατηρούν μια βάση δεδομένων, με τα ID των αρχείων μαζί με **metadata** για τα ίδια τα αρχεία, και τις αντίστοιχες διευθύνσεις IP των κόμβων ON που κρατούν αυτά τα αρχεία. Κατ' αυτό τον τρόπο, κάθε SN έχει παρόμοια δράση με αυτή ενός **server** στο **Napster**. Αλλά σε αντίθεση με το **Napster**, οι SN δεν είναι αποκλειστηκά αφιερωμένοι κεντρικοί υπολογιστές (ή "φάρμα" κεντρικών υπολογιστών) αλλά, κόμβοι που ανήκουν σε μεμονωμένους χρήστες οι οποίοι τυγχάνει να έχουν μεγάλη ταχύτητα σύνδεσης (**broadband connection**).



Σχήμα 5.3: Παράδειγμα συστήματος Kazaa. Με πράσινο συμβολίζουνται οι υπερκόμβοι (supernodes).

Για κάθε αρχείο που μοιράζεται ένας κόμβος ON, στέλνει **metadata** σε έναν υπερκόμβο SN τα οποία περιλαμβάνουν : το όνομα του αρχείου, το μέγεθος του, το **Content Hash** του αρχείου και διάφορες περιγραφές των αρχείων (πχ όνομα καλλιτεχνών και κείμενο που εισάγεται από τους χρήστες) τα οποία βοηθούν στην αναζήτηση.

Όταν ένας χρήστης σε κάποιο κόμβο ON αναζητά κάποιο αρχείο, στέλνει μια ερώτηση με τις λέξεις κλειδιά στον SN που είναι συνδεδεμένος. Ο κόμβος SN επιστρέφει τη διεύθυνση IP και τα **metadata** που αντιστοιχούν στο **keyword** που αναζητά ο χρήστης.

Κάθε SN διατηρεί επίσης συνδέσεις TCP με άλλους κόμβους SN, που δημιουργούν ένα δίκτυο μεταξύ των SN. Όταν ένας SN λαμβάνει μια ερώτηση, μπορεί να διαβιβάσει την ερώτηση σε έναν ή περισσότερους από τους SN με τους οποίους συνδέεται. Μια ερώτηση γενικά θα επισκεφτεί ένα μικρό υποσύνολο των SN, και ως εκ τούτου θα λάβει τα **metadata** ενός υποσυνόλου όλων των ON.

Gnutella [3]

Περισσότερο ενδιαφέρον παρουσιάζει το **Gnutella**, ένα δίκτυο στο οποίο όλοι οι κόμβοι είναι ισότιμοι, και οι διαδικασίες εισαγωγής και αναζήτησης γίνονται χωρίς τη βοήθεια ειδικών κόμβων.

Όταν ένας νέος χρήστης, έστω ο Α, θέλει να συνδεθεί στο δίκτυο της **Gnutella**, απλά επικοινωνεί με έναν συνδεδεμένο στο δίκτυο χρήστη, π.χ. τον Β. Αυτό επιτυγχάνεται από τον Α στέλνοντας ένα **broadcast** μήνυμα. Το μήνυμα αυτό θα το λάβει ο Β, ο οποίος απαντάει, στη συνέχεια ο Α στέλνει ένα αναγνωριστικό μήνυμα μέσα στο δίκτυο της **Gnutella**, με σκοπό να γίνει γνωστός και στους υπόλοιπους, καθώς επίσης και ο Α να μάθει και κάποιους άλλους κόμβους εκτός του Β.

Το μήνυμα αυτό δεν ταξιδεύει σε όλο το δίκτυο, αλλά έχει ένα συγκεκριμένο χρόνο ζωής **TTL** (**Time To Live**).

Έτσι ο Α γνωρίζει τους γείτονες του. Για να γίνει μία αναζήτηση μέσα στο δίκτυο της **Gnutella**, χρησιμοποιείται η τεχνική της **πλημμύρας**. Η τεχνική της **πλημμύρας** περιγράφεται στην ενότητα 7.

Ένα πρόβλημα που εμφανίστηκε, το οποίο είναι αρκετά σημαντικό και συμβάλλει στην αδυναμία κλιμάκωσης του συστήματος είναι οι "ελεύθεροι χρήστες" (**free riders**). Οι χρήστες αυτοί δεν μοιράζονται αρχεία με αποτέλεσμα πολλές αναζητήσεις (οι οποίες έχουν ένα χρόνο ζωής **TTL**) να τερματίζονται χωρίς αρκετά αποτελέσματα.. Ένα άλλο μειονέκτημα είναι ότι η ανάπτυξη λογισμικού για τη χρήση του δικτύου γίνεται από ανεξάρτητους φορείς και πολλές βελτιώσεις αλλά και αλλαγές στο πρωτόκολλο, για την επίλυση τυχόν προβλημάτων, διαφέρουν σε κάθε υλοποίηση, με αποτέλεσμα να μην υπάρχει ουσιαστική επίλυση του προβλήματος.

Γενικά όμως το **Gnutella** έχει αρκετά πλεονεκτήματα, καθώς οι αναζητήσεις επιστρέφουν συχνά αποτελέσματα σε ερωτήσεις των χρηστών. Υπάρχει μεγάλη ανοχή σε σφάλματα, και το σύστημα είναι ευέλικτο σε αλλαγές στην τοπολογία του συστήματος.

Σύστημα Ομότιμων Κόμβων	**Αρχιτεκτονική**	**Τρόπος αναζήτησης**
Napster	Ύπαρξη κεντρικού εξυπηρετητή	Όλες οι ερωτήσεις στον εξυπηρετητή
Kazaa	Ύπαρξη υπερκόμβων	Ανάθεση των αναζητήσεων στους υπερκόμβους
Gnutella	Καθαρά κεντρικοποιημένη δομή	Παραλλαγή πλημμύρας (χρόνος TTL)

Πίνακας 5.4: Συγκεντρωτικά χαρακτηριστικά αδόμητων συστημάτων

5.2 Δομημένα Συστήματα

Τα συστήματα αυτά χαρακτηρίζονται από μια συγκεκριμένη δομή με την έννοια ότι υπάρχει κάποιος κανόνας για τις συνδέσεις μεταξύ των κόμβων καθώς και τα δεδομένα δεν τοποθετούνται τυχαία σε αυτούς άλλα σε προκαθορισμένες τοποθεσίες, γεγονός που βοηθά σημαντικά στην απόδοση του δικτύου.

Το χαρακτηριστικό των δομημένων συστημάτων ομότιμων κόμβων είναι ότι στα δεδομένα αναθέτονται μοναδικά αναγνωριστικά τα οποία λέγονται **κλειδιά**. Με βάση τα κλειδιά γίνεται η αντιστοίχιση μεταξύ κόμβων και δεδομένων, η οποία δείχνει για ποια δεδομένα είναι υπεύθυνος ο κάθε κόμβος. Η ανάθεση κλειδιών σε κόμβους γίνεται με τη χρήση συναρτήσεων κατακερματισμού (***hashing***) έτσι ώστε τα δεδομένα να διαμοιράζονται όσο το δυνατόν περισσότερο ομοιόμορφα. Με αυτόν τον τρόπο δομείται ένας γράφος στον οποίο οι κόμβοι είναι συνδεδεμένοι με καθορισμένο τρόπο.

Τα πιο χαρακτηριστικά δομημένα συστήματα είναι το **Content Addressable Network (CAN)** και το **Chord** για τα οποία θα μιλήσουμε παρακάτω.

5.2.1. Data Hash Table [4]

Οι Πίνακες Κατακερματισμού **(Hash Tables)** είναι δομές δεδομένων οι οποίες χρησιμοποιούνται για την αντιστοίχηση κλειδιών (**keys**) με τιμές (**values**). Βασικό ρόλο στην δημιουργία αυτών των δομών παίζουν οι συναρτήσεις κατακερματισμού (**hash functions**) που έχουν σαν είσοδο ένα στοιχείο οποιασδήποτε μορφής οπως integer,string κ.λ.π. και παράγουν στην έξοδο έναν αριθμό-κλειδί συγκεκριμένου εύρους.

Για να είναι αποδοτική μια συνάρτηση κατακερματισμού θα πρέπει οι παραγόμενες τιμές (**values**) να είναι διαφορετικές για διαφορετικά κλειδιά (**keys**). Μια γνωστή συνάρτηση κατακερματισμού είναι η **SHA-1** που παράγει τιμές εύρους 32bit.

hash_function (value) = key;

Οι Κατανεμημένοι Πίνακες Κατακερματισμού (**Distributed Hash Tables,DHT**) χρησιμοποιούνται κυρίως στα μη κεντρικοποιημένα δομημένα συστήματα (**decentralized structured systems**) οπως το **CAN** και το **Chord**.

Αρχικά αναπτύχθηκαν σε ερευνητικό επίπεδο με σκοπό την εύκολη καταχώρηση και διαγραφή πληροφοριών που συνδέονται με κάποια κλειδιά.

Στα συστήματα που χρησιμοποιούν **DHT** οι IP διευθύνσεις των κόμβων περνάνε μέσα απο τη συνάρτηση κατακερματισμού ώστε κάθε κόμβος να αποκτήσει ένα μοναδικό m-bit αναγνωριστικό (**id**). Τα κλειδιά που δημιουργούνται κατακερματίζοντας τις πληροφορίες (τα ονόματα των αρχείων που έχουν οι κόμβοι) έχουν το ίδιο αριθμητικό εύρος με τα id των κόμβων.

Σαν παράδειγμα μπορούμε να θεωρήσουμε την μετατροπή ενός αλφαριθμητικού σε ακέραιο αριθμό με βάση των πίνακα τιμών **ASCII**. Για παράδειγμα τα γράμματα που αποτελούν την λέξη "World" έχουν τιμές 87,111,114,108 και 100. Το άθροισμα αυτό είναι 520 και θα πρέπει να αποθηκευτεί στο κόμβο εκείνο που το **Id** του είναι μεγαλύτερο ή ίσο απο την τιμή του **key**.

Τα **χαρακτηριστικά των συστημάτων** που κάνουν χρήση των **DHTs** ειναι:

1)Μη κεντρικοποιημένη τοπολογία (**decentralisation**) – κάθε κόμβος συμμετέχει ισότημα χωρίς την ύπαρξη κάποιου κεντρικού server.

2)Μεγάλη κλιμάκωση (**scalability**) - ακόμα και χιλιάδες ή εκατομμύρια κόμβοι να συνδεθούν δεν επειρεάζεται η ικανότητα του συστήματος.

5.2.2 CAN [5]

Το **Content Addressable Network** (**CAN**), είναι δομημένο σαν ένα εικονικό σύστημα συντεταγμένων d διαστάσεων (**d-torus**). Στο Σχ. 1, φαίνεται η δομή ενός **CAN**.

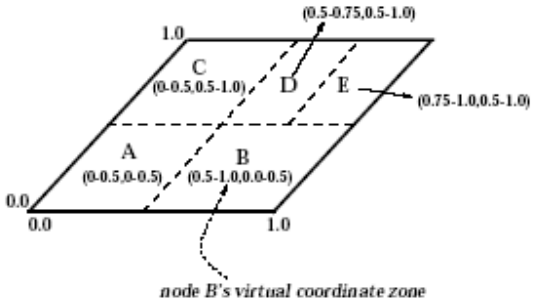
Ο χώρος διαμερίζεται σε ***ζώνες***, δυναμικά ανάμεσα στους κόμβους έτσι ώστε κάθε κόμβος να αναλαμβάνει μία ή περισσότερες ζώνες. Κάθε κόμβος κρατάει τις συντεταγμένες της ζώνης του και τις συντεταγμένες των γειτονικών ζωνών. Επίσης, κάθε κόμβος κρατάει ένα ζεύγος ***κλειδιού-δεδομένου (K,V)***.

Η αντιστοίχιση του ζεύγους (**K,V**) σε κάποιο κόμβο γίνεται με τη χρησιμοποίηση μιας συνάρτησης κατακερματισμού η οποία αντιστοιχίζει το κλειδί K σε ένα σημείο P στον εικονικό χώρο συντεταγμένων. Το (**K,V**) αποθηκεύεται σε εκείνο τον κόμβο που είναι υπεύθυνος για τη ζώνη στην οποία ανήκει το P. Η ανάκτηση ενός δεδομένου με κλειδί K, γίνεται χρησιμοποιώντας την ίδια συνάρτηση κατακερματισμού πάνω στο K, οπότε αυτό αντιστοιχίζεται σε ένα σημείο P.

Στη συνέχεια, από τον κόμβο στον οποίο ανήκει το P, ανακτούμε το δεδομένο V. Αν το σημείο P δεν ανήκει στον κόμβο που έκανε την αίτηση για το (**K,V**) και δεν ανήκει ούτε στους κόμβους γειτονικών ζωνών, τότε η αίτηση δρομολογείται στην κοντινότερη γειτονική ζώνη του P.

Αναζήτηση - Δρομολόγηση στο CAN

Για να δρομολογήσουμε ένα μήνυμα αναζήτησης από μια πηγή σε ένα συγκεκριμένο προορισμό, ακολουθούμε το μονοπάτι από τη πηγή στο προορισμό με βάση τις Καρτεσιανές συντεταγμένες του d-διάστατου χώρου στον οποίο έχει δομηθεί το **CAN**.

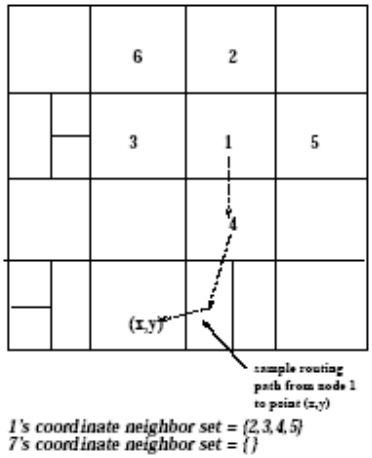


Σχήμα 1 : Δομή του CAN

Κάθε κόμβος κρατάει τις συντεταγμένες της ζώνης την οποία έχει αναλάβει, καθώς και τις συντεταγμένες των γειτονικών ζωνών.

Ένας greedy αλγόριθμος αναλαμβάνει να προωθήσει το μήνυμα στην κοντινότερη στον προορισμό γειτονική ζώνη, με βάση αυτές τις συντεταγμένες. Στο Σχ.2 φαίνεται ένα παράδειγμα δρομολόγησης. Έστω ότι ο κόμβος που έχει αναλάβει τη ζώνη 1, ψάχνει για το δεδομένο με συντεταγμένες (**x, y**). Ο κόμβος 1 θα προωθήσει το μήνυμα, σε μια από τις γειτονικές του ζώνες η οποία απέχει τη μικρότερη απόσταση από το (**x,y**).

Η ζώνη αυτή είναι η 4, άρα το μήνυμα αναζήτησης προωθείται προς τον κόμβο που έχει αναλάβει τη συγκεκριμένη ζώνη. Από τη ζώνη 4, το μήνυμα αναζήτησης προωθείται στον γειτονική ζώνη του (**x,y**), όπως φαίνεται στο σχήμα, αφού αυτή πλέον απέχει λιγότερο από το (**x,y**). Τέλος, το μήνυμα προωθείται στον κόμβο που ανήκει το δεδομένο (**x,y**) οπότε και τελειώνει η αναζήτηση.



Σχήμα 2 : Παράδειγμα αναζήτησης. Ο 1 αναζητάει το (χ, y)

Έτσι, για ένα χώρο διάστασης ***d***, χωρισμένο σε n ίσες ζώνες (κάθε ζώνη αντιστοιχεί σε ένα και μόνο ένα κόμβο, άρα συνολικά έχουμε ***n*** κόμβους), κάθε κόμβος έχει ***2d*** γείτονες και το μέσο μήκος ενός μονοπατιού είναι $(d/4)(n^{1/d})$. Επίσης, για ένα χώρο ***d*** διαστάσεων, αυξάνοντας τον αριθμό των κόμβων (άρα και τον αριθμό των ζωνών) το μέσο μήκος μονοπατιού αυξάνεται με $O(n^{1/d})$, ενώ ο όγκος της πληροφορίας που χρειάζεται να κρατάει ο κάθε κόμβος μένει ίδιος (***2d***). Άρα, η απόδοση της αναζήτησης είναι $O(d*n^{1/d})$, αφού ο αριθμός των μηνυμάτων που θα χρειαστούμε, είναι $O((d/4)(n^{1/d})) = O(d(n^{1/d}))$.

Κατασκευή του CAN - Εισαγωγή Κόμβου

Το **CAN** (πιο συγκεκριμένα ο εικονικός χώρος συντεταγμένων) κατασκευάζεται με την άφιξη κάθε καινούριου κόμβου. Έστω λοιπόν, ότι ένας κόμβος θέλει να ενταχθεί στο **CAN**.

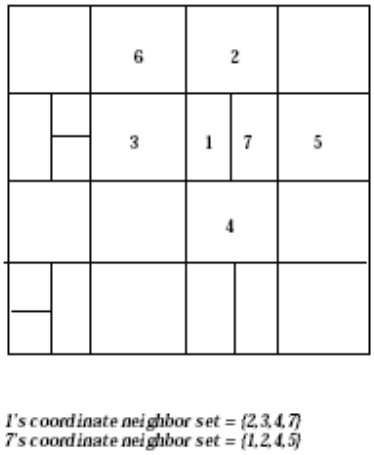
Πρώτα επιλέγει ένα τυχαίο σημείο P του χώρου συντεταγμένων και στέλνει ένα μήνυμα συνένωσης στον κόμβο ο οποίος αντιστοιχεί στη ζώνη που περιβάλλει το P. Όταν το μήνυμα φτάσει στον κόμβο που κατέχει το P, αυτός ο κόμβος διαιρεί τη ζώνη που του αντιστοιχεί σε δύο ίσες ζώνες.

Αν για παράδειγμα, έχουμε ένα χώρο 2 διαστάσεων, τότε ο κόμβος θα χωρίσει τη ζώνη του πρώτα κατά την πρώτη διάσταση (π.χ. κατά την X) και μετά κατά την δεύτερη διάσταση (π.χ. κατά την Y). Αν ο ίδιος κόμβος χρειαστεί να ξαναχωρίσει τη ζώνη του, τότε θα τη χωρίσει πάλι κατά την διάσταση X και η όλη διαδικασία επαναλαμβάνεται χωρίζοντας τη ζώνη κατά X, Y εναλλάξ.

Στη συνέχεια, τα ζεύγη (**K,V**) που ανήκουν στη ζώνη που δημιουργήθηκε και που θα καταλάβει ο καινούριος κόμβος, μεταφέρονται σε αυτόν τον κόμβο και οι γείτονες του καινούριου και του παλιού κόμβου ενημερώνονται για την άφιξη του καινούριου κόμβου και την δημιουργία καινούριας ζώνης.

Τέλος, ενημερώνεται και ο καινούριος κόμβος για τις γειτονικές ζώνες του. Στο Σχ. 3, φαίνεται η εισαγωγή του κόμβου 7. Πριν την εισαγωγή του 7 το **CAN** ήταν όπως στο Σχ. 2. Ο κόμβος 7 βρίσκει έναν κόμβο, τυχαία, ο οποίος υπάρχει ήδη στο σύστημα. Στην προκειμένη περίπτωση, βρίσκει τον κόμβο 1. Ο κόμβος 1 χωρίζει την ζώνη του στη μέση, κατά την διάσταση Y. Την μισή ζώνη αναλαμβάνει ο 1 και την υπόλοιπη ο 7.

Επίσης, ο 7 παίρνει και τα ζεύγη (K, V) που του αναλογούν και ενημερώνεται για τους γείτονές του. Τέλος, ενημερώνονται και όλοι οι γείτονες (οι 2, 3, 4, 5 και 6) για την εισαγωγή του 7.



Σχήμα 1: Παράδειγμα εισαγωγής κόμβου. Ο 7 εισάγεται στο σύστημα

Αποχώρηση Κόμβου, Αποτυχία Κόμβου και Συντήρηση στο CAN

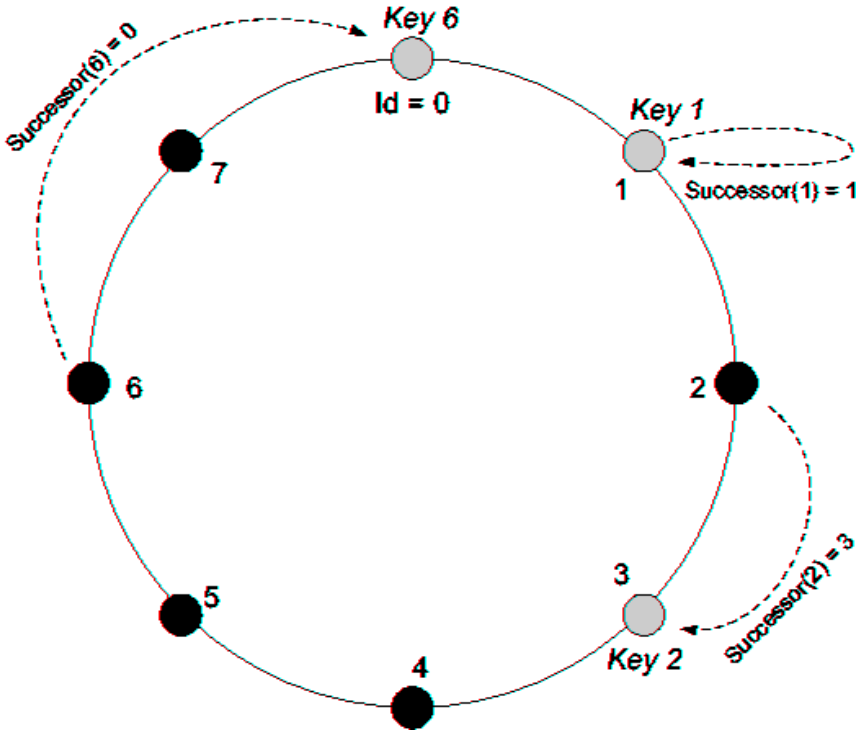
Όταν ένας κόμβος φεύγει εθελοντικά, τότε παραδίδει τη ζώνη του και τα ζεύγη (**K,V**) σε κάποιον γείτονά του (έτσι ώστε να διατηρούνται έγκυρες ζώνες, όπως περιγράφηκαν πιο πάνω, κατά την κατασκευή τους). Το πρόβλημα υπάρχει όταν ένας κόμβος αποτυγχάνει.

Σε αυτή την περίπτωση, οι μη-προσβάσιμοι κόμβοι θέτουν σε λειτουργία ένα ***αλγόριθμο άμεσης ανακατάληψης*** ο οποίος αντιστοιχίζει τη ζώνη του κόμβου που απέτυχε, σε κάποιον γείτονα. Μια αποτυχία ανιχνεύεται με περιοδικά μηνύματα που στέλνουν οι κόμβοι μεταξύ τους.

5.2.3 CHORD [6]

Στο **Chord** οι κόμβοι δομούνται σε ένα δακτύλιο. Σε κάθε κόμβο ανατίθενται κλειδιά, από μια συνάρτηση κατακερματισμού (***hashing***) και κάθε κόμβος επίσης, συνδέεται με O(logN) άλλους κόμβους με ντετερμινιστικό τρόπο, όπου N ο αριθμός των κόμβων.

Η συνάρτηση κατακερματισμού είναι τέτοια ώστε να διαμοιράζεται ο φόρτος σε όλους τους κόμβους του συστήματος, έτσι ώστε όταν εισέρχεται ή φεύγει ένας κόμβος από το σύστημα να απαιτείται μικρή μετακίνηση κλειδιών σε άλλους κόμβους (μετακινούνται μόνο O(1/N) κλάσμα των κλειδιών).



Εικόνα 5.2: Παράδειγμα δικτύου CHORD

Πιο συγκεκριμένα, η συνάρτηση κατακερματισμού αναθέτει σε κάθε κόμβο και σε κάθε κλειδί ένα m-bit ***αναγνωριστικό***. Για το αναγνωριστικό του κόμβου η συνάρτηση κατακερματισμού παίρνει ως είσοδο την IP διεύθυνσή του, ενώ για το αναγνωριστικό του κλειδιού παίρνει ως είσοδο το ίδιο το κλειδί. Τα αναγνωριστικά των κόμβων οργανώνονται σε ένα ***κύκλο modulo*** 2^m. Ένα κλειδί K ανατίθεται στον κόμβο εκείνο του οποίου το αναγνωριστικό είναι ίσο ή αμέσως μεγαλύτερο από το αναγνωριστικό του κλειδιού.

Ο κόμβος αυτός ονομάζεται ***successor(K)***. Τα αναγνωριστικά των κόμβων μπορούν να τοποθετηθούν σε δομή δακτυλίου αριθμημένα από 0 - 2^m-1.

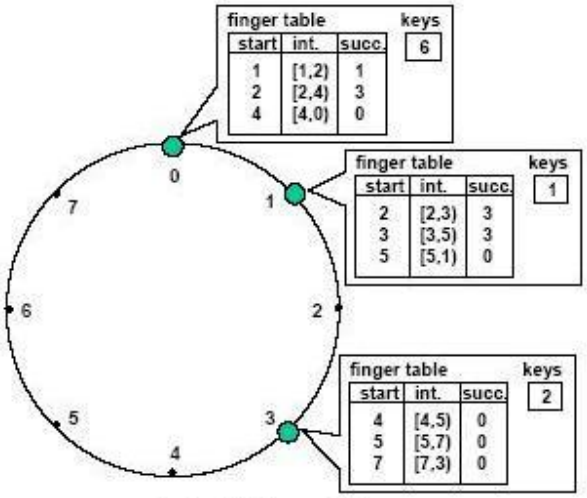
Σε αυτήν την περίπτωση ο ***successor(K)*** είναι ο επόμενος του K κόμβος στον κύκλο, μετρώντας με τη φορά του ρολογιού.

Έτσι, όταν ένας κόμβος n εισέρχεται στο δίκτυο, τότε τα κλειδιά που είχαν ανατεθεί στον ***successor*** του n, τώρα ανατίθενται στον n, ενώ αντίστοιχα, όταν ένας κόμβος n φεύγει από το δίκτυο, τότε όλοι οι κόμβοι που του είχαν ανατεθεί, τώρα ανατίθενται στον ***successor*** του n. Δεν χρειάζεται καμία άλλη αλλαγή στην ανάθεση των κλειδιών.

Στο Σχ. 5.2, φαίνεται ένας δακτύλιος **Chord** στον οποίο είναι παρόντες μόνο οι κόμβοι 0,1 και 3. Βλέπουμε ότι το δεδομένο 1 ανατίθεται στον ***successor(1)*** που είναι ο κόμβος 1. Το δεδομένο 2 θα ανατεθεί στον ***successor(2)*** που είναι ο κόμβος 3. Τέλος, το δεδομένο 6 θα ανατεθεί στον ***successor(6)*** που είναι ο κόμβος 0 (εδώ θα πρέπει να σημειώσουμε ότι και για τα δεδομένα 4, 5, 7 ο ***successor*** τους θα ήταν επίσης ο κόμβος 0).

Εντοπισμός κλειδιού στο Chord

Κάθε κόμβος διατηρεί ένα πίνακα δρομολόγησης m γραμμών, που καλείται ***finger table***. Η i-οστή γραμμή του ***finger table***, ενός κόμβου n, περιέχει την ταυτότητα του πρώτου κόμβου, s, που έπεται του n, τουλάχιστον 2^{i-1}, στον κύκλο των αναγνωριστικών, δηλαδή: s = ***successor***$(n + 2^{i-1})$, όπου $1 \le i \le m$. Ο κόμβος s καλείται i-οστός ***finger*** του n. Ο πρώτος ***finger*** του κόμβου n είναι ο αμέσως επόμενος κόμβος από αυτόν στον κύκλο. Κάθε γραμμή του ***finger table*** περιλαμβάνει εκτός από το αναγνωριστικό και την IP διεύθυνση του σχετικού κόμβου. Στο σχήμα 7 φαίνεται ο κύκλος αναγνωριστικών του σχήματος 6, με τα ***finger tables*** των κόμβων.



Σχήμα 7: Finger tables

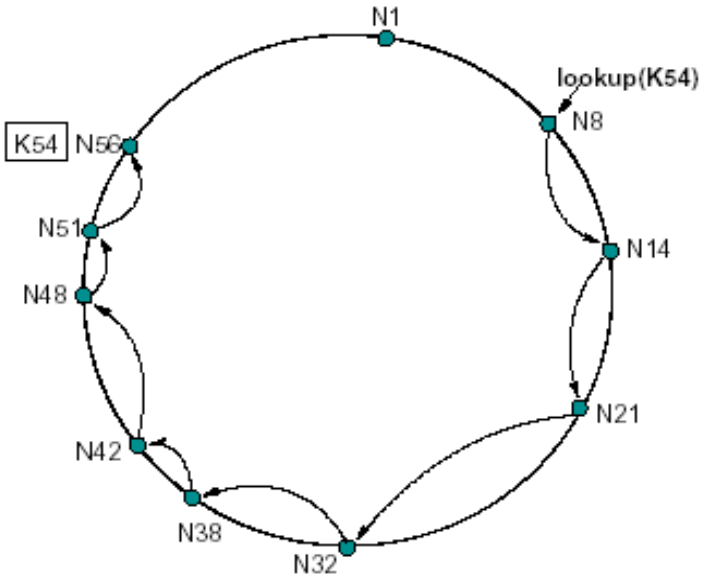
Όπως είναι εμφανές, το ***finger table*** ενός κόμβου δεν του παρέχει την πληροφορία για το ***successor*** οποιουδήποτε κλειδιού. Έτσι στην περίπτωση που ένας κόμβος n δεν έχει πληροφορία για το ***successor*** ενός κλειδιού k, ψάχνει στο ***finger table*** του για έναν κόμβο j που βρίσκεται πιο κοντά στο k (και πριν από αυτόν),στον κύκλο και τον «ρωτάει» για τον κόμβο που ξέρει ότι βρίσκεται πιο κοντά στο k. Επαναλαμβάνοντας αυτή τη διαδικασία, ο κόμβος n μαθαίνει για κόμβους, τα αναγνωριστικά των οποίων, βρίσκονται όλο και πιο κοντά στο k. Με τη διαδικασία αυτή, το κλειδί μπορεί να εντοπιστεί σε O(logN) βήματα.

<u>Αναζήτηση στο Chord</u> [9]

Όταν γίνεται μία αναζήτηση, ένας κόμβος n που θέλει να ανακτήσει κάποιο δεδομένο πρέπει να βρει τον κόμβο που το κατέχει. Η ιδέα είναι να βρεθεί το id του προηγούμενου κόμβου (***predecessor***) από αυτόν που κατέχει το ζητούμενο δεδομένο. Από εκεί με ένα βήμα φτάνει στον προορισμό. Αν ο n είναι ο "***predecessor***" του ζητούμενου κόμβου τότε η διαδικασία τελείωσε. Αν όχι τότε ο κόμβος n με βάση τον πίνακα που διατηρεί, με πληροφορίες για κάποιους κόμβους, βρίσκει τον κοντινότερο κόμβο στον "***predecessor***". Η διαδικασία αυτή επαναλαμβάνεται μέχρι να βρεθεί ο ζητούμενος κόμβος.

Στην περίπτωση που ένας κόμβος δεν περιέχει αρκετή πληροφορία για να προσδιορίσει τον ***successor*** ενός κλειδιού, ο κόμβος πρέπει να βρει ένα κόμβο του οποίου το αναγνωριστικό να είναι πιο κοντά στο K από ότι το δικό του αναγνωριστικό. Η διαδικασία αυτή προχωράει από κόμβο σε κόμβο μέχρις ότου βρούμε ένα κόμβο ο οποίος είναι ***successor*** του K. Στο Σχ. 5.3, φαίνεται μια αναζήτηση του κόμβου 8 (N8) για το κλειδί 54 (K54).

Αρχικά, ο N8 εξετάζει αν το K54 βρίσκεται μεταξύ αυτού και του ***successor*** του (N14). Δεν το βρίσκει εκεί, άρα η αναζήτηση προχωράει στον ***successor*** τού N8, στον N14. Ο N14 εξετάζει αν το K54 βρίσκεται μεταξύ αυτού και του ***successor*** του, N21. Δεν το βρίσκει και η αναζήτηση προχωράει στον ***successor*** τού N14, στον N21. Η διαδικασία αυτή συνεχίζεται μέχρις ότου βρεθεί ο ***successor*** τού κλειδιού K54, που είναι ο κόμβος N56.



Σχήμα 5.3: Παράδειγμα αναζήτησης. Ο κόμβος 8 αναζητά το κλειδί 54

Εισαγωγή Κόμβου στο Chord

Στη συνέχεια, θα δούμε τι ενέργειες γίνονται όταν ένας κόμβος εισάγεται στο δίκτυο. Για να το κάνουμε αυτό, θα πρέπει πρώτα να ορίσουμε τον ***predecessor*** ενός κόμβου. Έστω ένας κόμβος n και n΄ ο ***successor*** του. Τότε ο ***predecessor*** του n΄ είναι ο n. Όταν ένας κόμβος εισέρχεται στο δίκτυο, πρέπει να γίνουν οι εξής ενέργειες από το **Chord**:

α) να αρχικοποιήσει τον ***predecessor*** του και το ***finger table*** του καινούριου κόμβου,

β) να ενημερώσει τους ***predecessors*** και τα ***finger tables*** των παλιών κόμβων έτσι ώστε αυτά να ανταποκρίνονται και στην είσοδο του καινούριου κόμβου στο σύστημα,

γ) να ενημερωθεί το λογισμικό του ανώτερου επιπέδου έτσι ώστε να γίνει η μεταφορά των κλειδιών για τα οποία είναι πλέον υπεύθυνος ο καινούριος κόμβος.

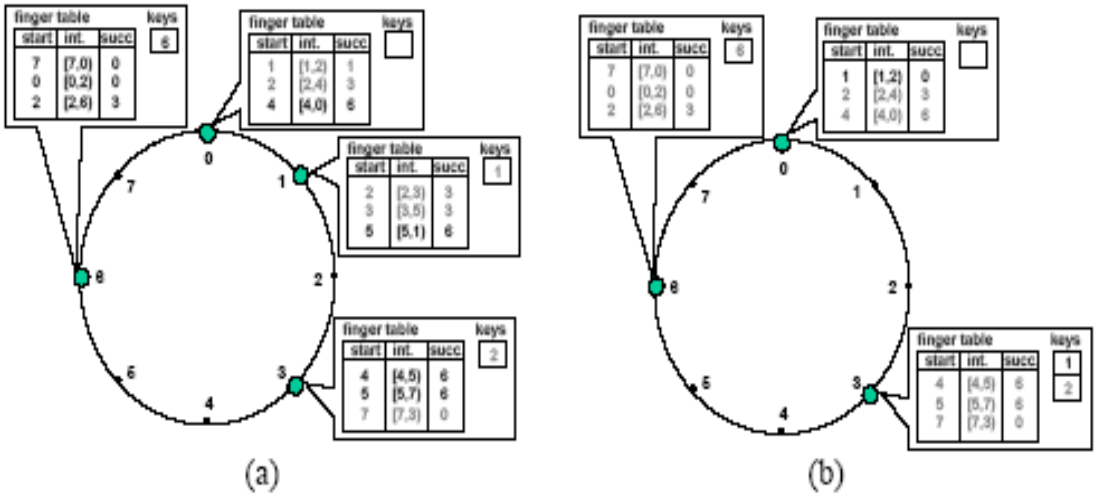
Στο Σχ. 5.4a), φαίνονται τα ***finger tables*** μετά την εισαγωγή του κόμβου 6. Επίσης, περιοδικά, οι κόμβοι του **Chord** «τρέχουν» ένα ***Stabilization*** πρωτόκολλο έτσι ώστε να εξασφαλίζεται η ορθότητα των δεικτών στους ***successors*** (είναι απαραίτητη για την επιτυχία των αναζητήσεων). Το πρωτόκολλο αυτό, χρησιμοποιεί τον "***Stabilization***" αλγόριθμο ο οποίος δουλεύει ως εξής. Έστω ότι ο κόμβος n θέλει να εισαχθεί στο σύστημα. Ο αλγόριθμος, ζητά από έναν τυχαίο κόμβο n΄ να βρει τον ***successor*** του n.

Στη συνέχεια, ο n ρωτάει τον ***successor*** του για τον ***predecessor*** p τού ***successor*** τού ίδιου τού n και αποφασίζει αν θα πρέπει ο n να γίνει ο ***predecessor*** του p. Ειδοποιεί τον ***successor*** τού n (έστω k) για την ύπαρξη του n, ώστε ο n να γίνει ο ***predecessor*** του k.

Τέλος, ο αλγόριθμος ενημερώνει τα ***finger tables*** όλων των κόμβων και αρχικοποιεί το ***finger table*** του καινούριου κόμβου. Ο αλγόριθμος εγγυάται ότι σε κάποιο χρονικό διάστημα μετά τη τελευταία εισαγωγή, όλοι οι κόμβοι θα έχουν σωστούς ***successor*** δείκτες.

Αποχώρηση Κόμβου, Αποτυχία Κόμβου στο Chord

Όταν ένας κόμβος n αποχωρεί εθελοντικά, τότε ενημερώνει τον ***successor*** του, ότι τώρα, ο καινούριος ***predecessor*** του θα είναι ο ***predecessor*** του n (μεταφέρει στον ***successor*** και όλα τα κλειδιά που του αντιστοιχούν) και αντίστοιχα, ενημερώνει τον ***predecessor*** του ότι ο καινούριος ***successor*** του είναι ο παλιός ***successor*** του n. Επίσης, ενημερώνονται, αντίστοιχα, για τις αλλαγές αυτές και όλα τα ***finger tables*** τα οποία περιείχαν τον n.



Σχήμα 5.4: a) Παράδειγμα εισαγωγής κόμβου. Ο κόμβος 6 εισάγεται στο σύστημα**, b)** Παράδειγμα αποχώρησης κόμβου. Ο κόμβος 3 αποχωρεί από το σύστημα. Οι αλλαγμένες καταχωρήσεις φαίνονται με μαύρο χρώμα και οι υπόλοιπες με γκρι.

Στην περίπτωση που ένας κόμβος n αποτύχει, πρέπει να βρεθεί ο ***successor*** του n προκειμένου να ενημερωθούν τα ***finger tables*** στα οποία εμπεριέχεται ο n. Στο Σχ. 6 b) φαίνεται ένα παράδειγμα αποχώρησης κόμβου (του κόμβου 3).

5.2.4 Διαφορές *CAN-CHORD*

Το **CAN** και το **CHORD** αποτελούν δύο αρχιτεκτονικές δομημένων ***peer-to-peer*** συστημάτων, που βασίζονται στη χρήση κατανεμημένου πίνακα κατακερματισμού.

Στον Πίνακα 1 φαίνονται, συγκεντρωτικά, οι διαφορές μεταξύ **CAN** και **Chord**:

Πίνακας 1: Διαφορές CAN και Chord

Σύστημα	**CAN**	**Chord**
Αρχιτεκτονική	Πολυδιάστατος χώρος Καρτεσιανών συντεταγμένων	Δακτύλιος αναγνωριστικών Κόμβων
Πρωτόκολλο Αναζήτησης	Συναρτήσεις κατακερματισμού για αντιστοίχιση ζευγών (K,V) σε σημείο P του χώρου συντεταγμένων	Συναρτήσεις κατακερματισμού για αντιστοίχιση κλειδιού και id κόμβου
Μήκος μονοπατιού	$O(dN^{1/d})$	O(logN)
Καταστάσεις γειτόνων (εισαγωγή/διαγραφή)	2d	$\log^2 N$
Καταστάσεις γειτόνων (αναζήτηση)	2d	logN

5.2.5 PASTRY [7]

Κάθε κόμβος στο σύστημα **Pastry** έχει ένα μοναδικό, 128-bit Id. Το σύνολο των **Ids** διανέμεται ομοιόμορφα αυτό μπορεί να επιτευχθεί με τον κατακερματισμό (**hash**) της διεύθυνσης IP του κόμβου. Υποθέτοντας ένα δίκτυο **Pastry** που αποτελείται από τους κόμβους N, μπορεί να καθοδηγήσει σε οποιοδήποτε κόμβο λιγότερο από τα βήματα $\mathbf{log_{2^b} N}$ κατά μέσον όρο (το b είναι μια παράμετρος διαμόρφωσης με σύνηθες τιμή το 4).

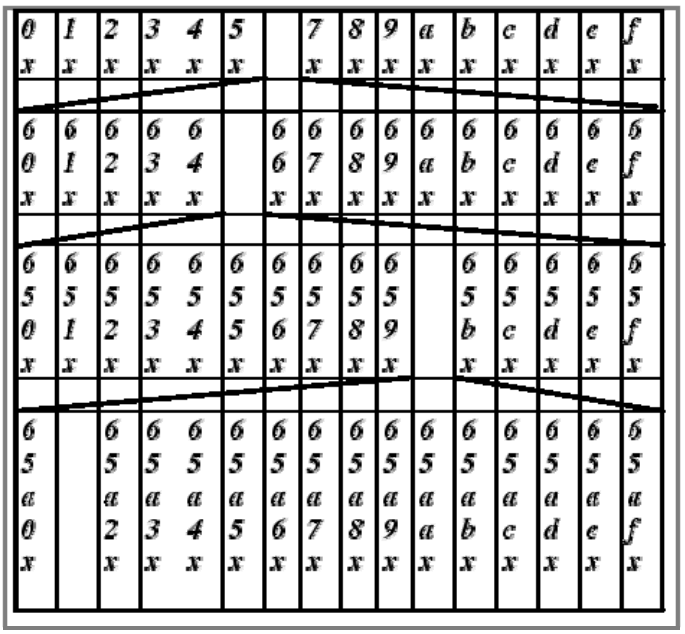
Το σύστημα **Pastry** δομεί και αυτό το δίκτυο του σε έναν κύκλο. Κάθε μήνυμα έχει ένα κλειδί, στέλνεται στον κόμβο με **id** πλησιέστερο αριθμητικά στο κλειδί του μηνύματος. Κάθε κόμβος διατηρεί έναν πίνακα με **L** καταχωρήσεις. Οι μισές έχουν στοιχεία για τους **L/2** πλησιέστερους κόμβους με μεγαλύτερο **id**, και οι υπόλοιπες μισές έχουν τους **L/2** κόμβους με αριθμητικά μικρότερο **id**.

Για λόγους απόδοσης, κάθε κόμβος διατηρεί και έναν δεύτερο πίνακα με δείκτες σε διάφορους κόμβους του δικτύου. Το **id** κάθε κόμβου αναπαριστάται με έναν αριθμό με **b bits**. Η i-οστή εγγραφή του πίνακα περιέχει τον κόμβο που έχει τα i πρώτα στοιχεία ίδια με τον κόμβο που διατηρεί το πίνακα και διαφέρουν στο **i+1 bit**.

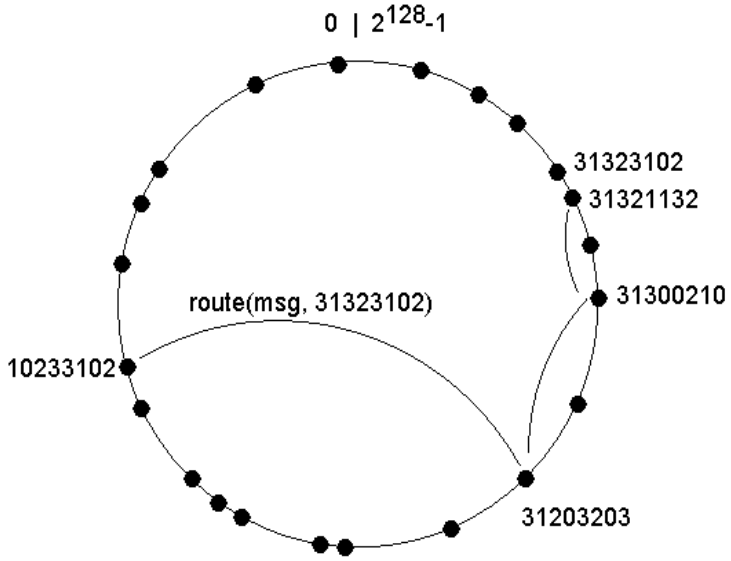
Όταν ένας κόμβος κάνει μία αναζήτηση, τότε κοιτάει στον πρώτο πίνακα αν έχει καταχωρημένο τον κόμβο αυτό. Αν δεν υπάρχει, τότε χρησιμοποιεί τις πληροφορίες του δεύτερου πίνακα. Για την ακρίβεια προωθεί την αναζήτηση σε έναν κόμβο ο οποίος είναι στον δεύτερο πίνακα και έχει μεγαλύτερο πρόθεμα στο **id**, που είναι ίδιο με το **id** του προς αναζήτηση κόμβου.

Αν ο κόμβος με το μεγαλύτερο πρόθεμα δεν υπάρχει, είτε διότι δεν είναι στον πίνακα, είτε δεν υπάρχει, τότε πρέπει να προωθηθεί σε έναν κόμβο με το ίδιο μήκος κοινού προθέματος, ο οποίος έχει **id** αριθμητικά μικρότερο από το κλειδί αναζήτησης. Έτσι η αναζήτηση θα πάει στον κόμβο που είναι πλησιέστερος στο κλειδί, αλλά πριν από αυτό. Αυτή η πληροφορία πρέπει να υπάρχει στον πρώτο πίνακα. Σε διαφορετική περίπτωση, η αναζήτηση έχει φτάσει στο πλησιέστερο κόμβο ή στον αμέσως προηγούμενο κόμβο.

Ακόμα και με ταυτόχρονες αποχωρήσεις κόμβων, η ενδεχόμενη παράδοση είναι εγγυημένη εκτός αν **L/2** ή περισσότεροι κόμβοι με παρακείμενο **id** αποσυνδεθούν ταυτόχρονα (το **L** είναι μια παράμετρος με σύνηθες τιμή το 16). Στον πίνακα με τους γείτονες κάθε κόμβου συνδυάζονται τα **id** με τις διευθύνσεις IP.



Σχήμα 1: Πίνακας δρομολόγησης κόμβου Pastry με ID 65a1x



Σχήμα 2: Παράδειγμα δρομολόγησης σε δομημένο σύστημα Pastry. Αναζήτηση κλειδιου 31323102 απο τον κόμβο με ID 10233102

5.2.6 KADEMLIA [8]

Το **Kademlia** υιοθετεί τη βασική μέθοδο των δομημένων **p2p** συστημάτων. Τα κλειδιά κατακερματίζονται σε κλειδιά των 160-bit. Κάθε ένας από τους συμμετέχοντες υπολογιστές έχει ένα Id 160-bit στο ίδιο διάστημα με τα κλειδιά. Τα ζευγάρια **value / keys** αποθηκεύονται στους κόμβους με **id** πλησιέστερα στο κλειδί. Τέλος, ένας βασισμένος αλγόριθμος δρομολόγησης αφήνει καθενός να εντοπίσει τους κεντρικούς υπολογιστές κοντά σε ένα κλειδί προορισμού.

Πολλά από τα οφέλη του **Kademlia** προκύπτουν από τη χρήση του ενός νέου μετρικού συστήματος για την απόσταση μεταξύ των σημείων και βασίζεται στην λογική **XOR**. Οποιοσδήποτε κόμβος έχει μια πολύ λεπτομερή γνώση των κόμβων που είναι κοντά σε αυτόν,και ταυτόχρονα έχει ελλιπή γνώση των πολύ απόμακρων κόμβων.

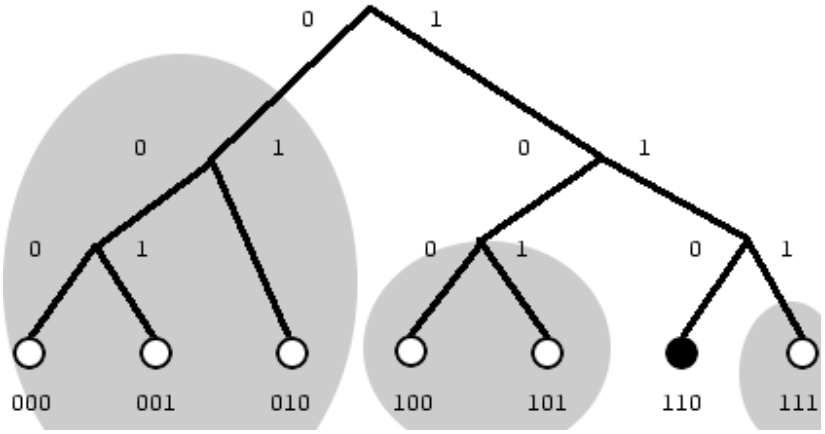
Η γνώση που έχει ένας κόμβος ποικίλλει με την απόσταση. Μια έννοια της απόστασης καθορίζεται, έτσι ώστε ένας κόμβος Α μπορεί πάντα να πει ποιός απο τους κόμβους Β και Γ είναι πιο κοντά σε αυτον (στον Α). Οποιοσδήποτε κόμβος έχει μια πολύ λεπτομερή γνώση των κόμβων που είναι κοντά σε αυτόν,και ταυτόχρονα έχει ελλιπή γνώση των πολύ απόμακρων κόμβων.

Ο σκοπός του **Kademlia** είναι να αποθηκεύσει, και να ανακτήσει τιμές (**values**), συνήθως δείκτες στα αρχεία. Οι τιμές δείχνονται από τα κλειδιά. Τα αναγνωριστίκα των κόμβων και τα κλειδιά μπορούν να υπολογίσουν την απόσταση ακριβώς όπως μεταξύ δύο ταυτότητες κόμβων. Επομένως οποιοσδήποτε κόμβος Α μπορεί να υπολογίσει την απόσταση του ανάμμεσα σε δύο κλειδιά K1 και K2 και να αποφασίσει ποιο ένα είναι πιό κοντά σε αυτόν. Με τον ίδιο τρόπο, ένας κόμβος Α μπορεί να υπολογίσει ποιο από τα δυο κλειδιά K1 και K2 είναι πιο κοντά σε έναν άλλο γνωστό κόμβο Β από το άλλο κλειδί. Οι κόμβοι Α και Β θα συμφωνήσουν σε αυτόν τον υπολογισμό.

Ο αλγόριθμος **Kademlia** είναι βασισμένος στον υπολογισμό της απόστασης μεταξύ δύο κόμβων. Αυτή η απόσταση είναι ο υπολογισμός της λογικής πράξης "αποκλειστικό ή" (**XOR**) των **id** δύο κόμβων, και το αποτέλεσμα είναι ένας ακέραιος αριθμός. Τα κλειδιά και οι ταυτότητες κόμβων έχουν το ίδιο μήκος, έτσι η απόσταση μπορεί να υπολογιστεί μεταξύ τους με ακριβώς τον ίδιο τρόπο.

Οι πίνακες δρομολόγησης στο **Kademlia** αποτελούνται από μια λίστα με τόσες θέσεις όσες τα ψηφία της ταυτότητας των κόμβων (π.χ. εάν ένα id κόμβου αποτελείται από 128 bit, ο κόμβος θα κρατήσει 128 θέσεις στη λίστα.). Κάθε είσοδος σε μια λίστα κρατά τα απαραίτητα στοιχεία για να εντοπίσει κάποιον άλλο κόμβο. Το στοιχείο σε κάθε καταχώτηση λιστας είναι η διεύθυνση IP, το port, και το id του κόμβου. Κάθε καταχώρηση αντιστοιχεί σε μια συγκεκριμένη απόσταση από τον κόμβο. Οι κόμβοι που μπορούν να πάνε στον N^{θ} κατάλογο πρέπει να έχουν ένα διαφορετικό N^{θ} ψηφίο από την ταυτότητα του κόμβου. Τα πρώτα N-1 bit της ταυτότητας των επόμενων κόμβων πρέπει να ταιριάζουν με εκείνοι της ταυτότητας του αρχικού κόμβου.

Στο **Kademlia**, οι λίστες αναφέρονται ως K-κάδοι (**K-buckets**).Το K είναι ένας αριθμός όπως το 20. Κάθε K-κάδος είναι ένας κατάλογος που έχει μέχρι K καταχωρήσεις στο εσωτερικό του. Δηλαδή όλοι οι κόμβοι στο δίκτυο θα έχουν στους καταλόγους τους μέχρι 20 κόμβους για κάθε διαφορετικο ψηφίο.



Σχήμα 1:Παράδειγμα δικτύου Kademlia

Σαν παράδειγμα θα δουμε το δίκτυο στο Σχήμα 1. Υπάρχουν επτά κόμβοι στο κατώτατο σημείο. Ο κόμβος υπό εξέταση είναι κόμβος έξι (δυαδικό 110) που φαίνεται με μαύρο χρώμα. Υπάρχουν τρεις K-κάδοι (**K-buckets**) σε αυτό το δίκτυο. Οι κόμβοι 0, 1 και 2 (το δυαδικό 000 ,001, και 010) είναι υποψήφιοι για τον πρώτο K-κάδο. Ο κόμβος 3 (το δυαδικό 011) δεν συμμετέχει στο δίκτυο. Στον δεύτερο K-κάδο τοποθετούται οι κόμβοι 4 και 5 (το δυαδικό 100 και 101). Τέλος, στον τρίτο K-κάδος μπορεί μόνο να περιέχει τον κόμβο 7 (το δυαδικό 111). Κάθε ένας από τους τρεις K-κάδους εσωκλείεται σε έναν γκρίζο κύκλο. Εάν το μέγεθος του K-κάδου ήταν δύο, κατόπιν ο πρώτος 2-κάδος θα μπορούσε μόνο να περιέχει δύο από τους τρεις κόμβους.

6. Προσομοίωση Δικτύου CHORD

Προφανώς, είναι αδύνατο να μοντελοποιήσουμε όλες τις παραμέτρους ενός **P2P** συστήματος. Για αυτό το λόγο, για την μελέτη των παραπάνω αλγορίθμων αναζήτησης της παρούσας πτυχιακής εργασίας κατασκευάστηκε ένας προσομοιωτής δικτύου βασισμένος στο σύστημα **Chord**. Για τις ανάγκες της εργασίας κατασκευάστηκαν 4 προσομοιωτές του συστήματος **Chord** όπου στον καθένα υλοποιείται και ένας διαφορετικός αλγόριθμος αναζήτησης.

- Ο $1^{ος}$ αλγόριθμος αναζήτησης που μελετάται είναι το σύστημα **Chord** και βασίζεται στο **DHT** για την επιτυχή εύρεση του κόμβου που έχει το ζητούμενο αρχείο.

- Οι υπόλοιποι 3 αλγόριθμοι που υλοποιήθηκαν αποτελούν παραλλαγές της μεθόδου της Πλυμμήρας (**flooding**) με σκοπό να μελετηθούν τα αποτελέσματα που έχει πάνω σε δομημένα συστήματα.

Η υλόποιησή τους έγινε με τη γλώσσα προγραμματισμού C .

<u>Παράμετροι προσομοίωσης δομημένου δικτύου</u>

- 1) Ο κάθε κόμβος του δικτύου περιγράφεται από ένα 13-bit ID. Αυτό σημαίνει ότι η μέγιστη δυνατή χωρητικότητα του συστήματος ειναι 2^{13} (= 8192) κόμβοι. Από αυτές τις θέσεις, οι 1600 καταλαμβάνονται από ενεργούς κόμβους.

- 2) Κάθε κόμβος διαθέτει 13 δείκτες (**fingers**) οι οποίοι δείχνουν σε διάφορους άλλους κόμβους με βάση τον τύπο id + 2^{i-1} όπου το id αντιστοιχεί στο αναγνωριστικό κάθε κόμβου και η μεταβλητή i παίρνει τιμές από i = 0,1,2….,M(= 13).

- 3) **TTL**(**Time to Live**) Η παράμετρος αυτή δείχνει την διάρκεια ζωής που έχουν τα queries τα οποία δρομολογούνται στο δίκτυο. Η μέγιστη διάρκεια του **TTL** στην προσομοίωσή μας ειναι 8, όπως προέκυψε μετά από διάφορα πειράματα.

Query types για την προσομοιωση

Για τις ανάγκες του αλγορίθμου αναζήτησης **DHT** δημιουργήσαμε βάση δεδομένων μεγέθους 300 δεδομένων τα οποία με τη βοήθεια της συνάρτησης κατακερματισμού **(hashing)** μετατράπηκαν σε ακέραιους αριθμούς (**keys**). Στη συνέχεια χρησιμοποιώντας τη συνάρτηση **successor()** γίνεται η κατανομή των κλειδιών.

Στους αλγορίθμους αναζήτησης με **Flooding** υλοποιήσαμε **multi attribute queries**. Τα queries αυτά αποτελούνται απο 4 διαφορετικές **resource classes**.

Computing Power	1.70 , 1.80 , 1.90 ,…., 3.60 GHz
Memory Amount	512 , 1024 , 2048 , 4096 MByte
Storage Space	100 , 150 , 250 , 400 GByte
Operating System	“Windows XP” , “Linux”

Πινακας 6.1: Resource classes & Attribute Queries

Τα παραπάνω **attributes** κατανέμονται με τυχαίο τρόπο στους κόμβους του δικτύου χρησιμοποιώντας την βοηθητική συνάρτηση **rand()** όπως φαίνεται παρακάτω:

```
current->RAM = Memory[rand()%7];
current->CPU  = CPUfreq[rand()%12];
current->HDD = HDDrive[rand()%8];
current->OS   =   OpS[rand()%3][0];
```

7. Ανάλυση Αλγορίθμων Αναζήτησης

Ανάλυση αλγορίθμου αναζήτησης βασισμένου σε DHT

Υλοποίηση συνάρτησης Κατακερματισμού

Για την μετατροπή των δεδομένων (**values**) σε τιμές (**keys**) υλοποιήθηκε η παρακάτω συνάρτηση κατακερματισμού (**hash function**). Η συνάρτηση αυτή, όπως φαίνεται στο παράδειγμα 7.1, λαμβάνει από τον πίνακα δεδομένων (**sdata[][]**) τα ονόματα των αρχείων και τα μετατρέπει σε ακέραιους αριθμούς προσθέτοντας τις τιμές **ASCII** όλων των χαρακτήρων που αποτελούν το όνομα του αρχείου.

```
for(i=0; i<100; i++)
   {
      if(sdata[j][i]!='\0')
      {
         fprintf(fp3,"%c",sdata[j][i]);
         result = (result+sdata[j][i]);
      }
   }
```

Παραδειγμα 7.1: Συναρτηση Κατακερματισμου (hashing)

Υλοποίηση αλγορίθμου αναζήτησης βασισμένου σε Κατανεμημένους Πίνακες Κατακερματισμού (DHT)

Η αναζήτηση δεδομένων σε δομημένα συστήματα με τη χρήση πινάκων κατακερματισμού γίνεται με τη χρήση των δεικτών (**fingers**) του κάθε κόμβου. Όταν γίνεται μία αναζήτηση, ένας κόμβος που θέλει να ανακτήσει κάποιο δεδομένο πρέπει να βρει τον κόμβο που το κατέχει. Η όλη διαδικασία δρομολόγησης του μηνύματος (**query**) γίνεται με τη χρήση των **fingers**.

Ο κόμβος ο όποιος ξεκινάει τη διαδικασία αναζήτησης, ελέγχει αρχικά τα **fingers** του για να δει αν το συγκεκριμένο **Key** που αναζητά βρίσκεται στον πίνακα δεικτών του. Αν δεν συμβαίνει αυτό τότε αναζητά τον πλησιέστερο κόμβο με **ID** μικρότερο ή ίσο από το ζητούμενο **Key**. Από κει και πέρα η διαδικασία συνεχίζεται προωθώντας το μήνυμα στους πλησιέστερους κάθε φορά κόμβους έως ότου βρεθεί ο κάτοχος του **Key**.

```
while(current->id!=point->id)  //kanoume anazhthsh mexri na vroume ton node pou psaxnoume
 {
    for(b=0; b<Mbits; b++)
    {
       if(current->finger.table[b]->id == point->id)
       {
          current = current->finger.table[b];
          fprintf(fp8,"We are in Node[%d]!\n",current->id);
          fprintf(fp8,"We made %d steps to find the song!!!\n",a);
          return ;
       }
    }

    for(b=0; b<Mbits; b++)
    {
       if(b == 12)
       {
          if(point->id > current->finger.table[11]->id &&  point->id < current->finger.table[12]->id)
          {
                current = current->finger.table[11];
                fprintf(fp8,"We are in Node[%d]!\n",current->id);
                a++;
                break;
          }
```

```
            else
            {
                  current = current->finger.table[12];
                  fprintf(fp8,"We are in Node[%d]!\n",current->id);
                  a++;
                  break;
            }
      }

      else
      if(point->id > current->finger.table[b]->id && point->id < current-
>finger.table[b+1]->id)
      {
            current = current->finger.table[b];
            fprintf(fp8,"We are in Node[%d]!\n",current->id);
            a++;
            break;
      }

   }

 }
```

Παραδειγμα 7.2: Τμήμα κώδικα για την υλοποίηση του αλγόριθμου αναζήτησης με DHT

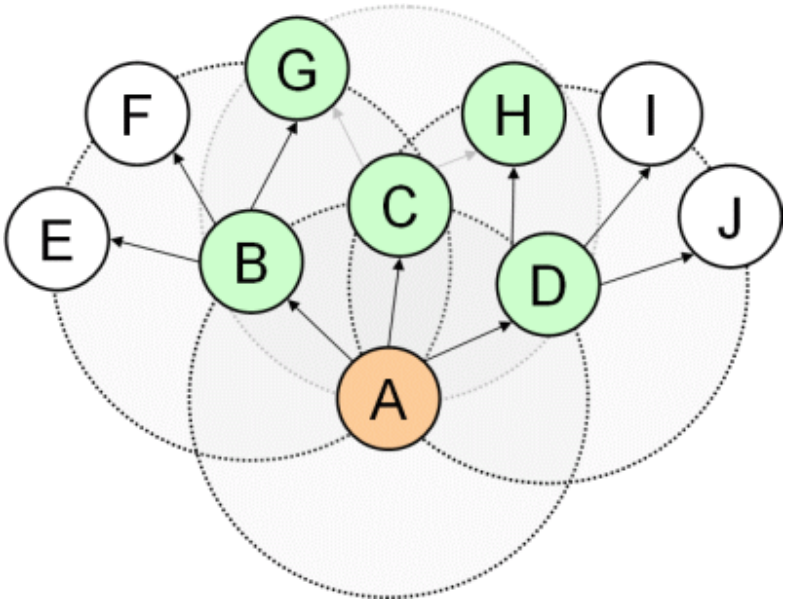
Ανάλυση Αλγορίθμων Flooding

Gnutella Flooding

Είναι η πρώτη μορφή πλημμύρας (**flooding**) που μελετήσαμε στην οποία κάθε κόμβος που δέχεται ένα μήνυμα, το προωθεί με βάση τον πίνακα των δεικτών του στους γείτονες που γνωρίζει.

Στη συνέχεια κάθε νέος κόμβος που έλαβε το μήνυμα, το προωθεί και αυτός με τη σειρά του στους γείτονες του. Η διαδικασία τερματίζεται, όταν τελειώσει η διάρκεια ζωής του μηνύματος (**TTL**) ή όταν λάβουμε ικανοποιητικό αριθμό θετικών απαντήσεων. Ο αλγόριθμος αυτός, εξασφαλίζει ότι όλοι οι κόμβοι θα λάβουν το μήνυμα, αλλά το ίδιο μήνυμα θα το λάβουν κάποιοι πολλές φορές, λόγω κύκλων στο γράφημα του δικτύου.

Στο πάρακάτω σχήμα φαίνεται ένα παράδειγμα αλγορίθμου πλημμύρας (**flooding**).



Σχήμα 7.3: Παράδειγμα αλγορίθμου flooding

Βλέπουμε ότι ο κόμβος A στέλνει το **μήνυμα** στα **fingers** του δηλαδή, B, C, D οι οποίοι με τη σειρά τους το προωθούν στα δικά τους **fingers**. Η διαδικασία τερματίζει όταν δεχθούν το **query** όλοι οι κόμβοι που βρίσκονται στο δομημένο αύτο δίκτυο.

Limited Flooding

Ο δεύτερος αλγόριθμος αναζήτησης που μελετήσαμε αποτελεί μια παραλλαγή του **Gnutella Flooding**. Σκοπός του αλγόριθμου αυτού είναι να περιορίσει το πλήθος των μηνυμάτων που προωθεί κάθε κόμβος που λαμβάνει το μήνυμα.

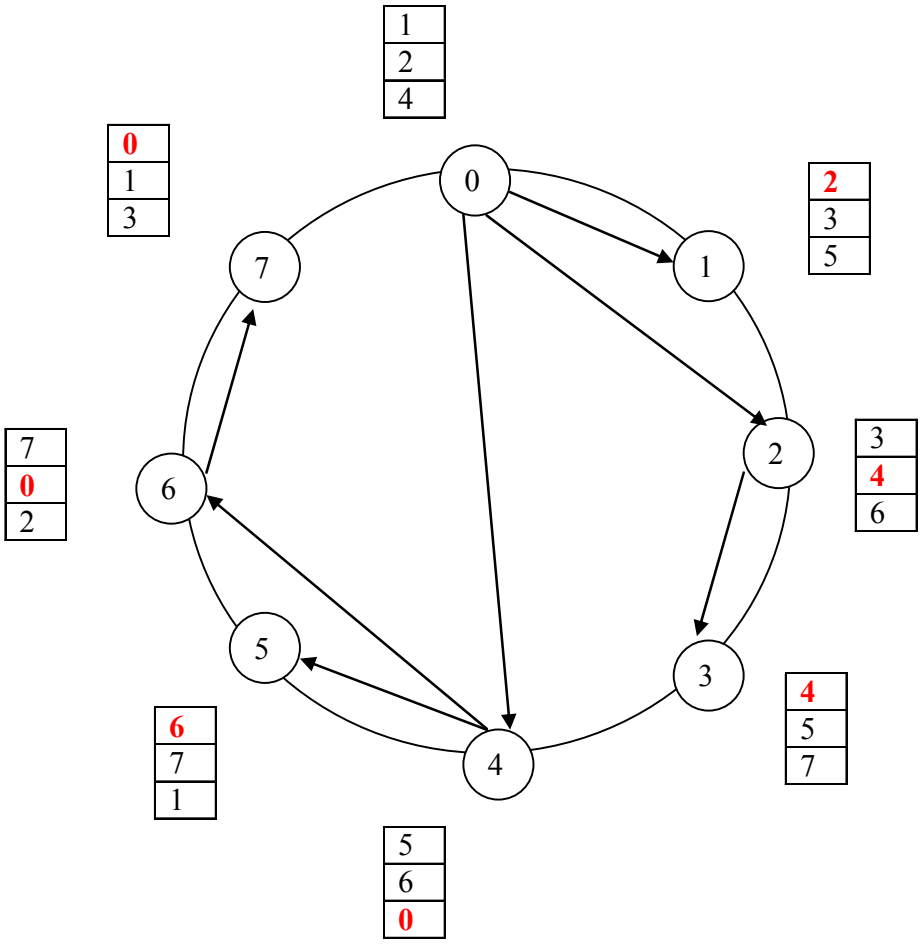
Σε κάθε μήνυμα, που επεξεργάζεται από τους κόμβους, υπάρχει ένας παράγοντας περιορισμού (**limit argument**), ο οποίος καθορίζει σε ποιους κόμβους θα προωθήσει το μήνυμα.

Όταν κάποιος κόμβος συναντήσει στον πίνακα με τους γείτονές του το **limit**, τότε δεν προωθεί το μήνυμα στους κόμβους που βρίσκονται στις παρακάτω θέσεις από αυτό (δηλαδή το **limit**).

Στην περίπτωση που το **limit** βρεθεί στην πρώτη θέση του πίνακα, τότε ο κόμβος δεν προωθεί το μήνυμα σε κανένα κόμβο. Τέλος, αν το **limit** υπάρχει στην τελευταία θέση του πίνακα, τότε ο κόμβος στέλνει το μήνυμα σε όλους τους κόμβους εκτός από αυτόν που έχει το **limit argument** (δηλαδή τον τελευταίο).

Στο Σχ 7.4 φαίνεται η τεχνική που χρησιμοποιεί ο αλγόριθμος της Περιορισμένης Πλημμύρας. Στο παράδειγμα αυτό, οι αριθμοί με κόκκινο χρώμα είναι οι κόμβοι που εχουν το **limit argument** στον πίνακα δεικτών του κάθε κόμβου.

Περιγράφεται ακόμη η διαδικασία με την οποία τοποθετείται ένα **limit argument** σε έναν κόμβο.



Σχήμα 7.4: Παράδειγμα αλγορίθμου Limited-Flooding

Υποθέτουμε ότι κόμβος 0 ξεκινάει την **πλημμύρα** και προωθεί αρχικά το μήνυμα στους κόμβους που δείχνουν τα **fingers** του (δηλαδή στους κόμβους 1, 2, 4). Στη συνέχεια ο κόμβος 1, επειδή δείχνεται απο το 1^{ο} **finger** του κόμβου 0, τοποθετεί το **limit** στο πρώτο **finger** του πίνακα δεικτών του (δηλαδή στο finger που δείχνει τον κόμβο 2).

Αυτό έχει σαν αποτέλεσμα, ο κόμβος 1 να μην προωθεί το μήνυμα σε κανένα κόμβο από τον πίνακα δεικτών του, επειδή το **limit** βρίσκεται στην πρώτη θέση.

Ο κόμβος 2, δείχνεται από το 2^{ο} **finger** του κόμβου 0, άρα θα έχει το **limit** στη δεύτερη θέση του πίνακά του. Γι' αυτό και θα προωθήσει το μήνυμα μόνο στο πρώτο **finger** του (δηλαδή στον κόμβο 3).

Τέλος, ο κόμβος 4 δείχνεται από το 3^{ο} **finger** του κόμβου 0, άρα θα έχει το **limit** στην τρίτη θέση του πίνακά του. Δηλαδή θα στείλει το μήνυμα στο πρώτο και δεύτερο **finger** του, και όχι στο τρίτο (δηλαδή θα στείλει στους κόμβους 5 και 6).

Η διαδικασία αυτή, συνεχίζεται σε όλους τους κόμβους του δικτύου έχοντας σαν αποτέλεσμα τον περιορισμό των πλεονάζωντων μηνυμάτων. Η τεχνική αυτή, όπως φαίνεται και στο παράδειγμα 7.5, αναγκάζει τον κάθε κόμβο να δέχεται μονάχα ένα μήνυμα κάθε φορά που κάποιος κόμβος ξεκινάει τη διαδικασία αναζήτησης.

Στο παράδειγμα 7.5 τοποθετούνται τα **limit arguments** στους κόμβους και γίνεται η προώθηση των μηνυμάτων μόνο στους κόμβους που επιτρέπεται (όπως εξηγήσαμε παραπάνω).

```
if(cur->query_limit==0)
  {
    if (cur->RAM == RAMquery && cur->CPU == (float)CPUquery && cur-
      >HDD == HDDquery && cur->OS == 'X')
    {
       counter++;
       cur->query_limit++;
       results++;
       return cur->id;
    }
   else
    {
       counter++;
       cur->query_limit++;
       return 0;
    }
  }
```

Παραδειγμα 7.5: Τμήμα κώδικα για τον περιορισμό των μηνυμάτων

Incremental Flooding

Ο τρίτος αλγόριθμος που υλοποιήσαμε είναι ο **Incremental Flooding**, ο οποίος συνδυάζει τεχνικές σειριακής-παράλληλης δρομολόγησης. Αυτός ο αλγόριθμος διατηρεί το **limit argument** κατά τη δρομολόγηση ενός μηνύματος, και επιπλέον εισάγει και τον παράγοντα **D**, τον βαθμό δηλαδή παραλληλίας του συστήματος.

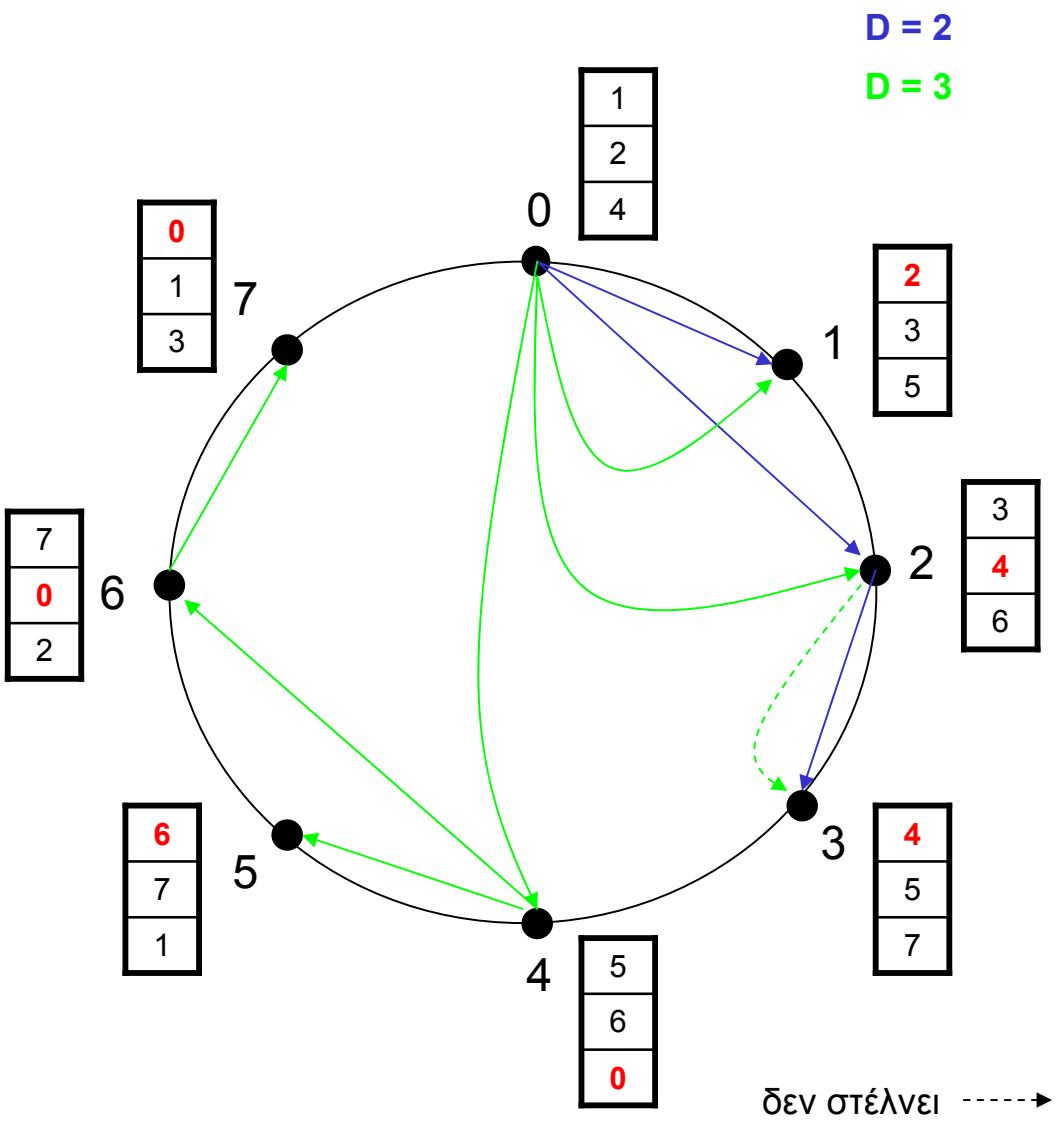
Κατά την έναρξη της διαδικασίας της αναζήτησης, ο χρήστης έχει τη δυνατότητα να επιλέξει το μέγεθος του **D**, δηλαδή σε πόσα **fingers** από τον πίνακα δεικτών θα προωθηθεί το **query**.

Αν η επιλογή του **D** είναι τέτοια ώστε να βρεθεί ικανοποιητικός αριθμός απαντήσεων κατά τη δρομολόγηση του μηνύματος, τότε η αναζήτηση τερματίζεται. Αν ο αριθμός των θετικών απαντήσεων (**query hits**) δεν είναι ο επιθυμητός, τότε η αναζήτηση αρχίζει πάλι από τον αιτών κόμβο (**requester**) με το **D** να αυξάνεται σε **D + 1**.

Η διαδικασία αυτή συνεχίζεται μέχρι να βρεθεί ο ζητούμενος αριθμός απαντήσεων ή μέχρι να γίνει το **D = M**, όπου **M** είναι ο μέγιστος αριθμός των **fingers** των κόμβων. Αν συμβεί το **D = M**, τότε ο αλγόριθμος γίνεται πλήρως παράλληλος και ταυτίζεται με τον **Limited Flooding**.

Στην περίπτωση που το **D** γίνει **D = 1**, τότε ο αλγόριθμος λειτουργεί σειριακά, δηλαδή το μήνυμα δρομολογείται από κόμβο σε κόμβο με τη σειρά που είναι τοποθετημένοι στο **Chord** (ο κάθε κόμβος στέλνει ουσιαστικά στον «διπλανό» του).

Στο σχήμα 7.6 φαίνεται ένα παράδειγμα δρομολόγησης μηνυμάτων, ξεκινώντας με **D = 2** και συνεχίζοντας με **D = 3** (δηλαδή μέχρι να βρεθεί ο ζητούμενος αριθμός απαντήσεων το D αυξάνεται). Στο σχήμα φαίνεται η δρομολόγηση του μηνύματος, ξεκινώντας από τον κόμβο 0. Τα βελάκια με μπλέ χρώμα αντιπροσωπεύουν τα μηνύματα που προωθούνται για **D = 2** και με πράσινο για **D = 3**. Τέλος με κόκκινο χρώμα φαίνεται το **limit argument**.



Σχήμα 7.6 : Παράδειγμα Incremental Flooding

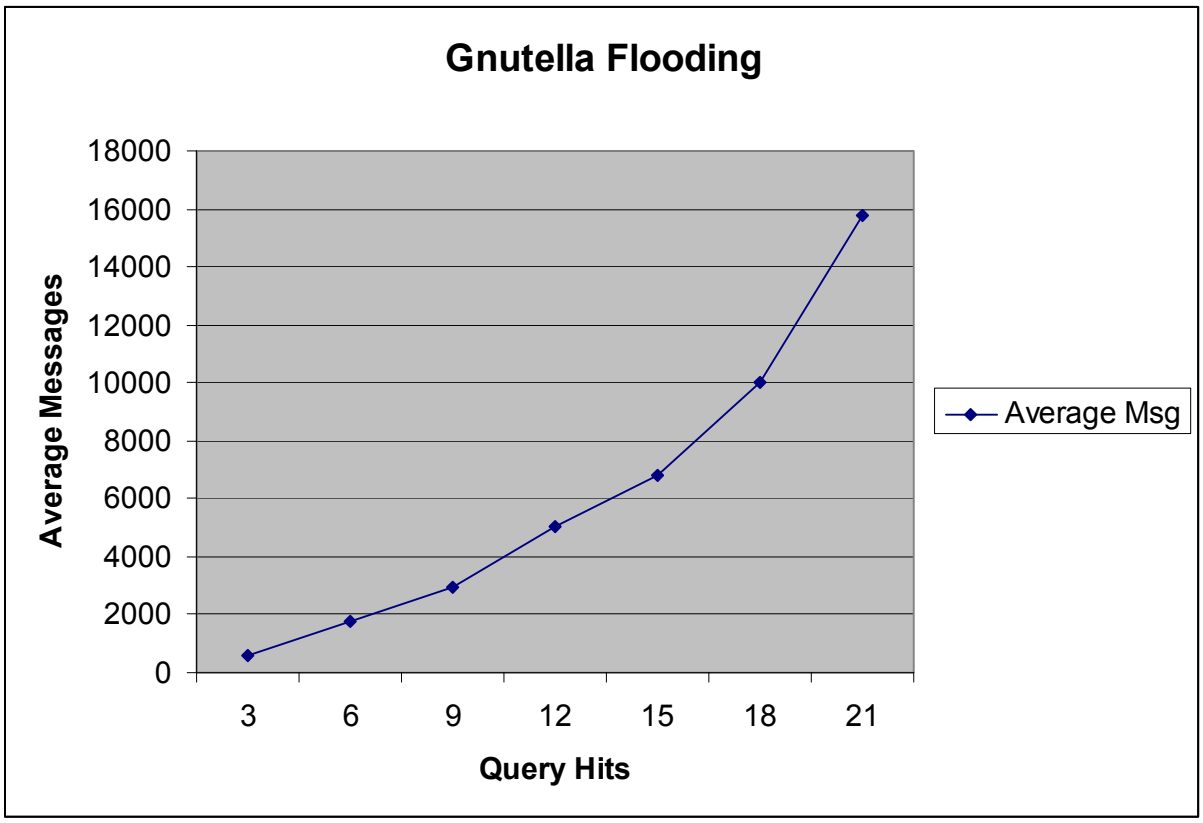
8. Σχολιασμός Γραφικών Παραστάσεων

Στην παρακάτω γραφική παράσταση ζητούνται τα εξής δεδομένα:

CPU: 2.60 GHz και **RAM** : 2048 MByte

Έχοντας εκτελέσει τον αλγόριθμο του **Gnutella Flooding** σε 50 κόμβους, παρατηρούμε ότι όσες περισσότερες θετίκες απαντήσεις ζητάμε, τόσο περισσότερα είναι και τα μηνύματα που πρέπει να στείλουμε. Ο αριθμός δηλαδή των θετικών απαντήσεων είναι ανάλογος των μηνυμάτων που προωθούνται προκειμένου αυτές να βρεθούν.

Όπως φαίνεται από την γραφική παράσταση, το πλήθος των queries που στέλνεται είναι αρκετά μεγάλο, αφού κάποιοι κόμβοι του δικτύου μπορεί να λάβουν περισσότερες από μια φορές ένα μήνυμα, λόγω κύκλων στο γράφημα του δικτύου.

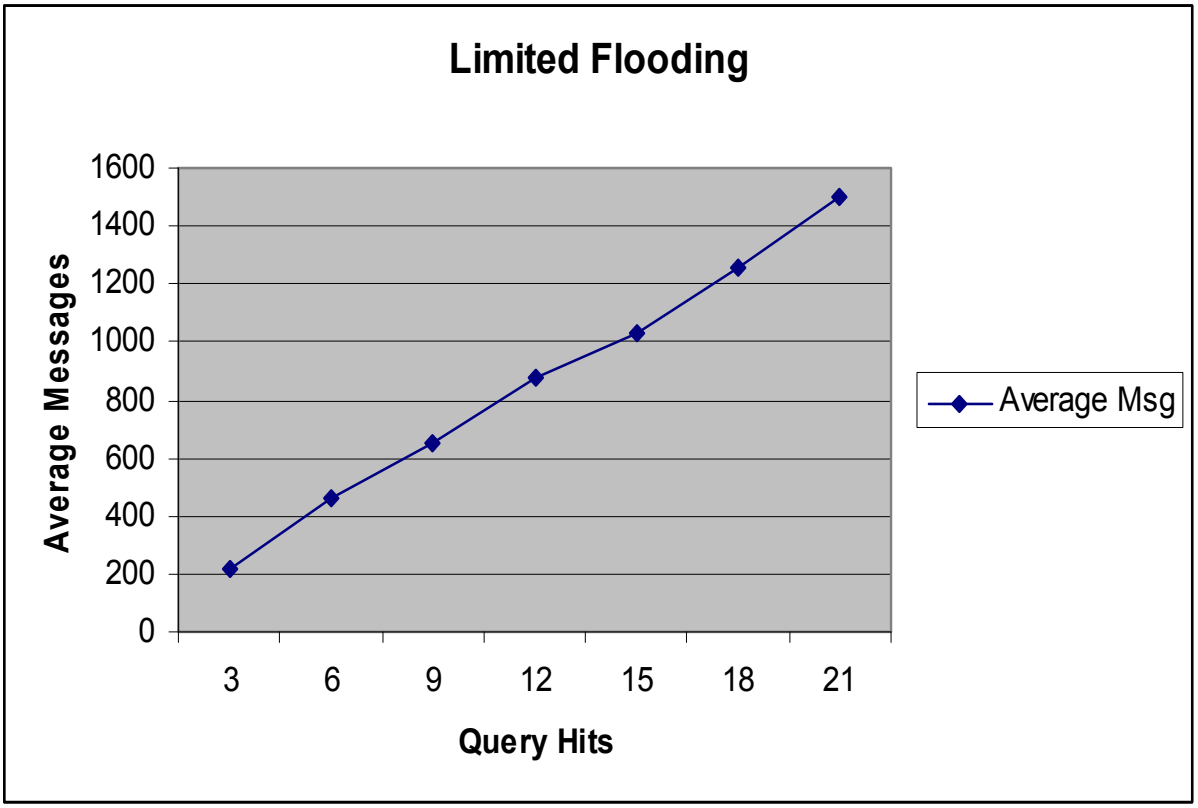


Στην παρακάτω γραφική παράσταση ζητούνται τα εξής δεδομένα:

CPU: 2.60 GHz και **RAM** : 2048 MByte

Έχοντας εκτελέσει τον αλγόριθμο του **Limited Flooding** σε 50 κόμβους, παρατηρούμε και εδώ ότι, ο αριθμός των θετικών απαντήσεων είναι ανάλογος των μηνυμάτων που προωθούνται προκειμένου αυτές να βρεθούν.

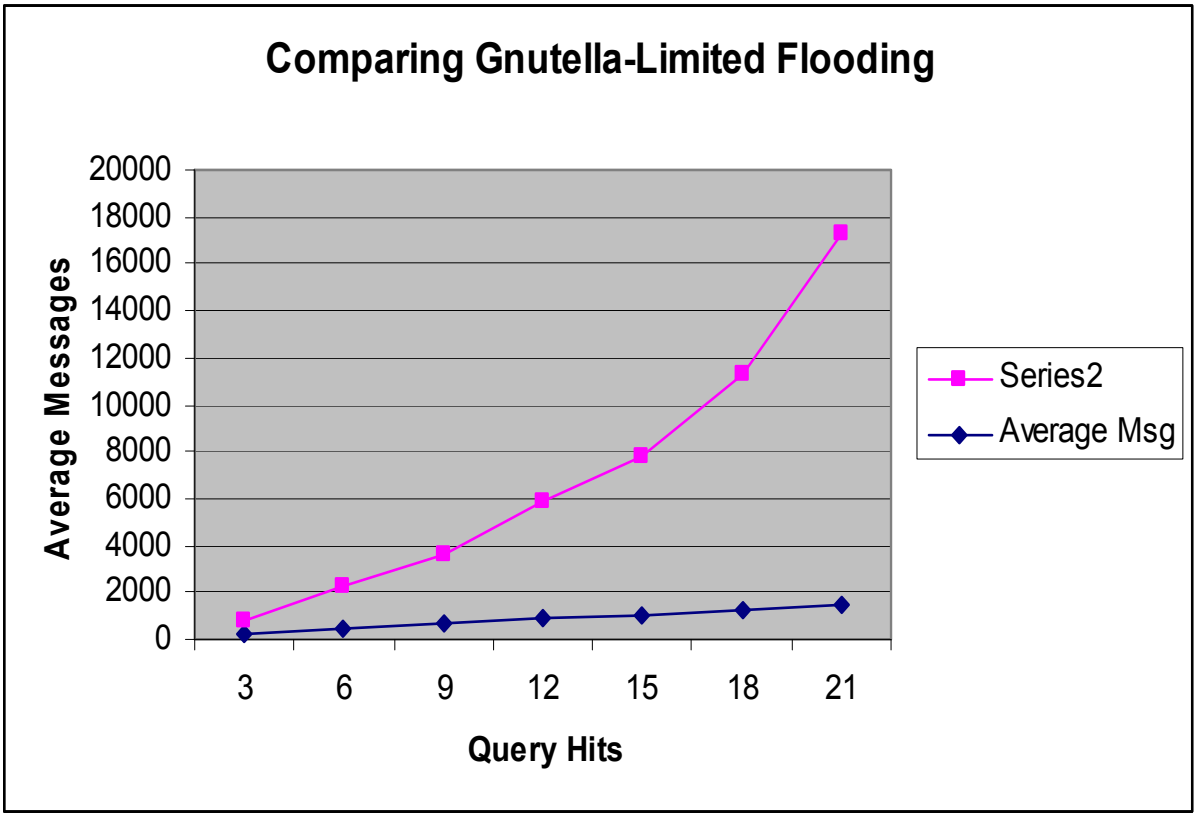
Παρόλα αυτά, στην παρακάτω γραφική παράσταση βλέπουμε ότι, το πλήθος των queries που στέλνεται είναι σχετικά μικρό σε σχέση με τα αποτελέσματα του **Gnutella Flooding**, αφού οι κόμβοι του δικτύου χρησιμοποιούν το **limit argument**, με αποτέλεσμα κάθε κόμβος να λαμβάνει το πολύ ένα μήνυμα.



Η παρακάτω γραφική παράσταση παρουσιάζει την διαφορά μεταξύ του **Gnutella** και του **Limited Flooding**.

Παρατηρούμε ξεκάθαρα την μεγάλη διαφορά που υπάρχει στην αποστολή των μηνυμάτων ανάμεσα στους 2 αυτούς αλγορίθμους. Το **Limited Flooding** προωθεί στους κόμβους του δικτύου πολύ λιγότερα μηνύματα από το **Gnutella Flooding**.

Από την παρακάτω γραφική παράσταση συμπεραίνουμε ότι, η τεχνική που χρησιμοποιεί το **limit argument**, δίνει σαφώς λιγότερα σφάλματα ενώ είναι και πιο αποδοτική.

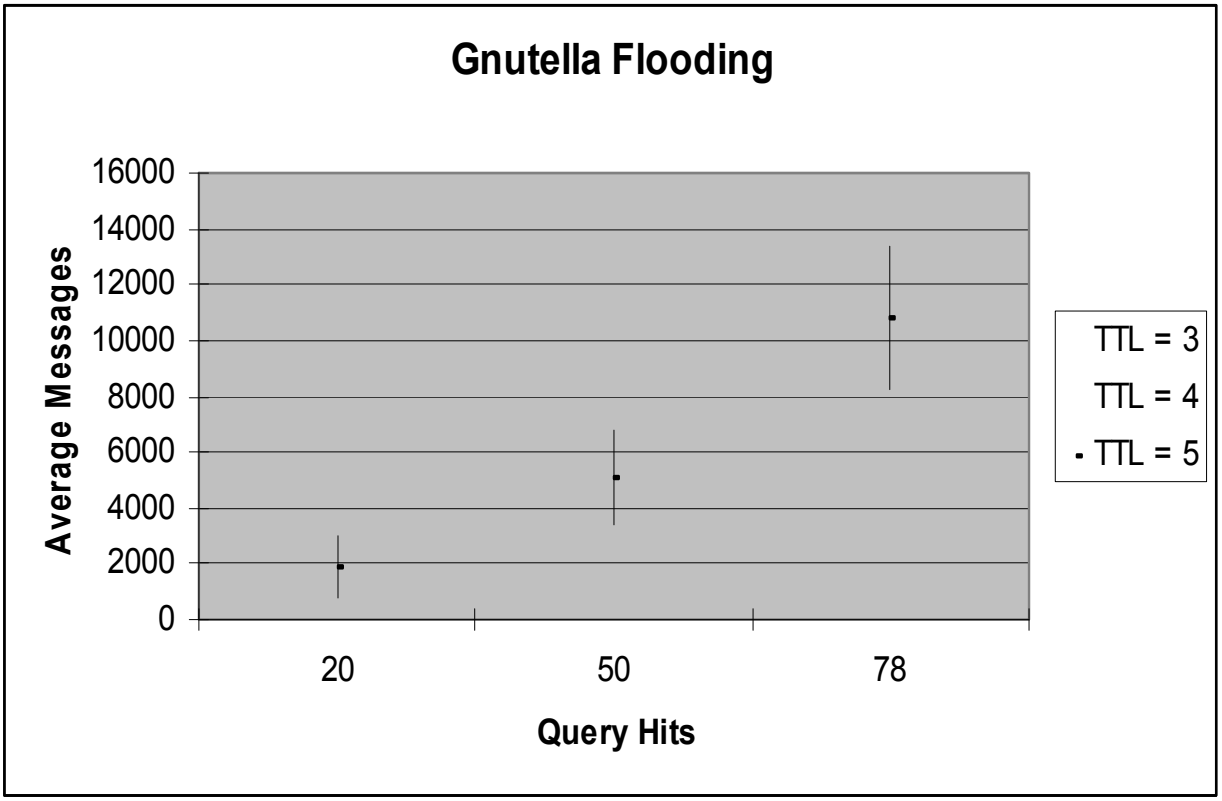


Στην παρακάτω γραφική παράσταση ζητούνται τα εξής δεδομένα:

HDDrive: 200 GByte και **OpS** : "XP"

Έχοντας εκτελέσει τον αλγόριθμο του **Gnutella Flooding** με την παράμετρο **TTL**, βλέπουμε τα αποτελέσματα των πειραμάτων για κάποια **TTL** που επιλέξαμε. Παρατηρούμε ότι, όσο αυξάνεται η διάρκεια ζωής του μηνύματος (**TTL**), τόσο περισσότερες είναι οι θετικές απαντήσεις άρα και περισσότερα τα μηνύματα που θα σταλούν. Όπως φαίνεται και πιο κάτω, για **TTL** = 3 βρίσκουμε μέσο όρο 20 θετικές απαντήσεις (**query hits**) και στέλνουμε μέσο όρο 2000 μηνύματα. Ακόμη για **TTL** = 4 βρίσκουμε 50 θετικές απαντήσεις στέλνοντας 5000 μηνύματα. Τέλος για **TTL** = 5 έχουμε μέσο όρο 78 **query hits** προωθώντας μέσο όρο 11000 μηνύματα.

Από τις μετρήσεις αυτές συμπεραίνουμε ότι, το **TTL** είναι ένας μηχανισμός που δεν αποφέρει καλά αποτελέσματα στα **P2P** συστήματα.

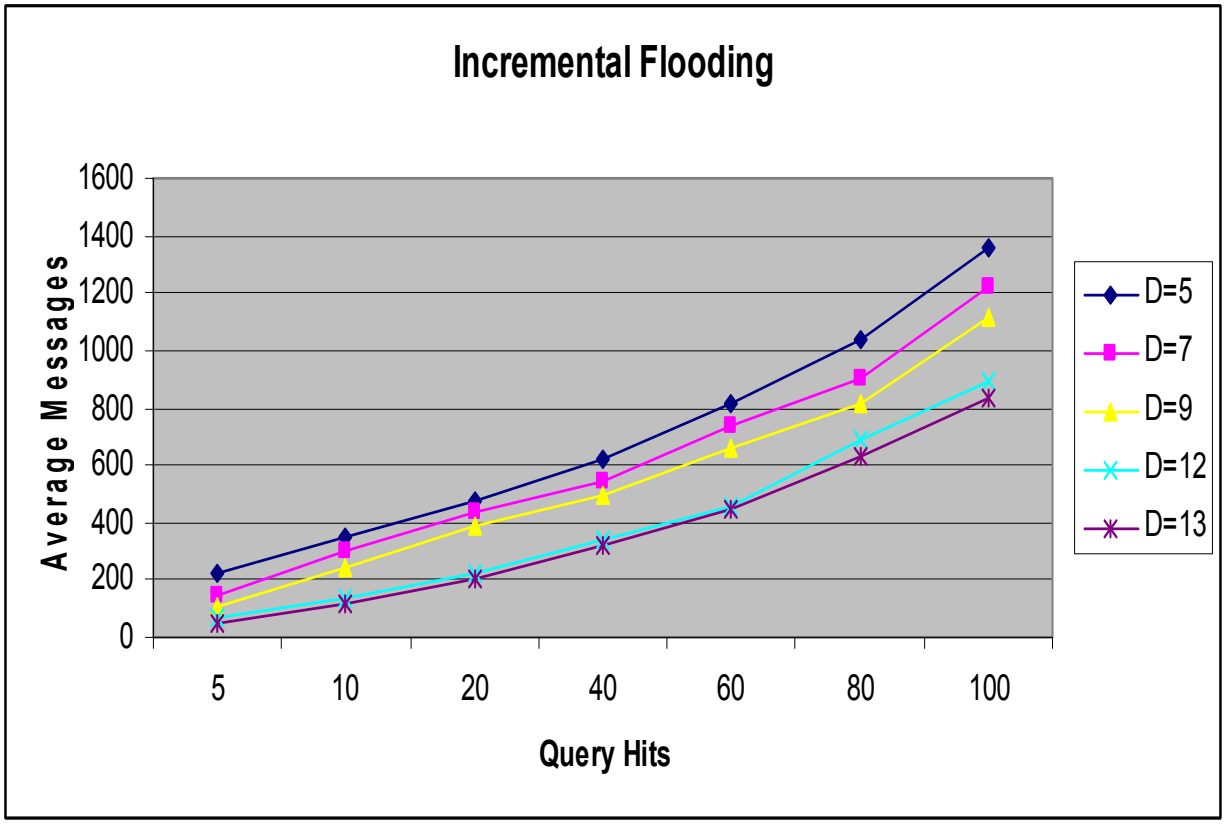


Στην παρακάτω γραφική παράσταση ζητούνται τα εξής δεδομένα:

HDD: 300 GByte **RAM** : 1024 MByte

Για τη μελέτη και αυτού του αλγόριθμου έγιναν μετρήσεις σε 50 κόμβους για να λάβουμε το μέσο όρο τον μηνυμάτων που διακινήθηκαν στο δίκτυο. Κάναμε πειράματα για 5 διαφορετικές τιμές της μεταβλητής **D**. Όπως φαίνεται και απο τη γραφική παράσταση, όσο μεγαλύτερη είναι η τιμή του **D** τόσο λιγότερο παράλληλη είναι η δρομολόγηση όπως και ο μέσος όρος των μηνυμάτων που στέλνονται, έως ότου το **D** γίνει ίσο με 13 όπου η διαδικασία γίνεται σειριακή και κάθε κόμβος ρωτάει απλώς τον επόμενο του στο δακτύλιο του **Chord**.

Για την εύρεση 100 κόμβων για παράδειγμα με τις απαιτούμενες προδιαγραφές (**query hit**), παρατηρούμε ότι στέλνονται στο δίκτυο 800 μηνύματα χρησιμοποιώντας **D = 13** ενω για **D = 5** χρειαζόμαστε περίπου 1400 μηνύματα.



9. Συμπερασματα Αλγορίθμων Αναζήτησης

Από τη μελέτη των παραπάνω αλγορίθμων, καθώς και από τα πειράματα που έγιναν σε αυτούς, προκύπτουν κάποια συμπεράσματα.

Τα συστήματα **P2P** παρουσιάζουν αρκετά πλεονεκτήματα σε σχέση με τα κεντρικοποιημένα συστήματα, όπως η ανοχή σε σφάλματα, ο διαμοιρασμός πόρων και η ανωνυμία.

Οι τεχνικές που υλοποιήθηκαν, αν και χρησιμοποιούνται κυρίως σε αδόμητα συστήματα **P2P**, συμπεραίνουμε ότι παρουσιάζουν καλά αποτελέσματα και κατά την εφαρμογή τους σε δομημένα συστήματα.

Ένας καλός αλγόριθμος δρομολόγησης, όπως φαίνεται και από τις γραφικές παραστάσεις, για δομημένα καθώς και για αδόμητα **P2P** συστήματα πρέπει να έχει τις εξής ιδιότητες:

- Η διαδικασία της αναζήτησης, πρέπει να τερματίζει με βάση τον αριθμό των θετικών απαντήσεων που αναζητά ο χρήστης, και όχι με βάση τη διάρκεια ζωής του μηνύματος (**TTL**).

- Ο διπλασιασμός των μηνυμάτων πρέπει να περιορίζεται όσο το δυνατόν περισσότερο. Κόμβοι οι οποίοι έχουν ερωτηθεί ήδη, δεν χρειάζεται να επεξεργάζονται ξανά τα ίδια μηνύματα. Αυτό μπορεί να επιτευχθεί με τη χρήση του περιοριστή μηνυμάτων (**limit argument**).

- Η δρομολόγηση πρέπει να είναι δυναμική και να καθορίζεται από το χρήστη ο βαθμός παραλληλίας του συστήματος, όπως γίνεται με τη χρήση του παράγοντα **D**.

10. Κώδικας Αλγορίθμου DHT

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 1600          // numbers of nodes in chord-ring right now
#define MAXID 8000
#define capacity (int)pow(2,Mbits)     // maximum chord capacity = 2^Mbits nodes
#define Mbits 13          // number of bits that describe each node in chord  8192
#define num 300
#define sNode 7556  //requester Node
#define on 1       //when node is online
#define off 0      //when node is offline


int i,j,datakey[num],pin[MAX];
char song[100]={"Aerosmith - I Don't want to miss a thing"};   //requested song

char data[][100]={{"Aerosmith - I Don't want to miss a thing"},{"Sweet
Emotion"},{"Air - Dead Bodies"},{"Kelly Watch The Stars"},{"Alanis Morissette -
Hand In My Pocket"},{"Head Over Feet"},{"Alanis Morissette - Joining
You"},{"Uninvited"},{"Alice Cooper - Bed Of Nails"},{"Billion Dollar
Babies"},{"Alice Cooper - Elected"},{"Halo Of Flies"},{"Alice Cooper - Hey
Stoopid"},{"House Of Fire"},{"Alice Cooper - I Never Cry"},{"No More Mr. Nice
Guy"},{"Alice Cooper - Schools Out For Summer"},{"Steven"},{"Alice Cooper -
Under My Wheels"},{"Smooth Criminal"},{"Anouk - it's a shame"},{"it's so
hard"},{"Anouk - together alone"},{"Nobody's_wife"},{"apollo440-charlies
angels"},{"Fuck U"},{"ARCHIVE - GET OUT"},{"your_time_has_come"},{"Avril
Lavigne - My Happy Ending"},{"Sk8er Boi Live at The Grammys 2003"},{"Avril
Lavine - Complicated"},{"American Jesus"},{"Beastie Boys - Sabotage"},{"fight
for your right to party"},{"Beautiful South - Living Thing"},{"Madonna"},{"beck-
diamond dogs"},{"Loser"},{"Big Sleep-Looking for a girl with a washing
machine"},{"Stay_Together_For_The_Kids"},{"blondie-maria"},{"The Roof Is On
Fire"},{"Bloodhound Gang - Discovery Channel"},{"Song 2"},{"BLUR-Song
2"},{"Fear Of The Dark"},{"BonJovi - It's my life"},{"childrenm of the
revolu"},{"Bregovic-Iggy Pop-In The Death Car"},{"I Love Rock'n'Roll"},
{"Calexico - Gypsy's Curse"},{"erase rewind"},{"Cardigans-favorit game"},{"top
of the world"},{"Chumbawamba--Tubthumping"},{"Don't Panic Me"},{"coldplay -
In my place"},{"just my imagination"},{"cranberries - promises me"},{"Strange
World"},{"cranberries-animal enstinct"},{"Zombie"},{"Cruel Intentions - Placebo
- Every You Every Me "},{"Lullaby"},{"cypress hill - rock superstar"},{"bohemian
like you"},{"dandy warhols-bohemian like you"},{"24000 Baci"},{"david bowie-
thursdays child"},{"WALK INTO THE SUN"},{"Duran Duran - Ordinary
World"},{"save tonight"},{"Elvis Presley vs. JXL - A Little Less
```

**Conversation"},{"Why Took You So Long"},{"Eurotrip Soundtrack - Lustra - Scotty Doesn't Know"},{"Bring Me To Life"},{"Evanescence - Going Under"},{"fire_escape"},{"FatBoy Slim - The Rockafeller Skank"},{"FEELING A MOMENT"},{"First Day Of My Life"},{"Five-We Will Rock You"},{"Livin' On My Own"},{"Garbage - Only Happy When It Rains"},{"i think im paranoid"},{"Gardigans-My_favorite_game"},{"i just wanna live"},{"Green Day - American Idiot"},{"basket case"},{"Green Day - Basket"},{"redundant"},{"Green Day - when i come around"},{"Boulevard Of Broken Dreams"},{"Highlander"},{"Him - Join Me"},{"Wicked game"},{"him - wings of a butterfly"},{"disappear"},{"Hoobastank - The Reason is You"},{"The Passenger"},{"james blunt - you're beautiful"},{"crush"},{"joanne osbourne-One of us"},{"Unchain My Heart"},{"JOE COKER - You can leave yor hat on"},{"L.S.F. LOST SOULS FOREVER"},{"Kittie - Brackish"},{"All In My Head"},{"Leanne Rymes - Can't Fight the Moonlight"},{"I want to fly away"},{"Lenny Kravitz - Are You Gonna Go My Way"},{"american woman"},{"lenny kravitz-fly on"},{"Take A Look Around"},{"Limp Bizkit-Getcha Groove On"},{"My Generation"},{"Limp Bizkit-My Way"},{"Rollin"},{"Limp Bizkit-The One"},{"Toxic Rmx"},{"LinkinPark - Numb"},{"Narcotic"},{"maroon 5 - sunday morning"},{"This Love"},{"maroon5-harder to breathe"},{"Angel"},{"Metallica - Enter Sandman"},{"Mission Impossible Theme"},{"Moby - Extreme Ways"},{"Jam For The Ladies"},{"Moby - James Bond Theme"},{"Lift Me Up"},{"moby-everlovin"},{"honey"},{"MOBY-IN MY HEART"},{"IN THIS WORLD"},{"MOBY-ONE OF THIS MORNINGS"},{"SIGNS OF LOVE"},{"Moby-We Are All Made Of Stars"},{"WHAT DO YOU WANT"},{"Neve-it's_over_now"},{"brutal"},{"nickel back - how you remind me"},{"Don't Speak"},{"oasis-do you know what i mean"},{"Come Out And Play"},{"Offspring - Gotta Get Away"},{"SELF ESTEEM"},{"Opus - Live Is Life"},{"belfast"},
{"orbital - The Box"},{"Blue Monday"},{"Paparoach - Blood Brothers"},{"Last Report"},{"Pink - Get This Party Started"},{"Bitter End"},{"Placebo - Daddy Cool"},{"Sleeping With Ghosts"},{"Placebo-Allergic"},{"Every you every me"},{"Placebo-Pure Morning"},{"Scared of girls"},{"Placebo-Without you I'm nothing"},{"You don't care about us"},{"Portishead - Numb"},{"Come with me"},{"Pulp - Common People"},{"Disco 2000"},{"pulp fiction"},{"Queen - Another One Bites the Dust"},{"Iwant it all"},{"queen - Kind of Magic"},{"karma police"},{"Rage_Against The Machine- bulls_on_parade"},{"dancing queens"},{"rainning pleasure-fake"},{"Amerika"},{"Rammstein-du_hast"},{"du hast"},{"rasmus -In The Shadows web edit"},{"Other side"},{"Redvhotvchilyvpeppers-Scar tissue"},{"Bad Day"},{"rem - Bang And Blame"},{"Don't Go Back To Rockv"},{"rem - end of the world"},{"Everybody Hearts"},{"rem - Everybody Hurts"},{"imitation of life"},{"rem - It's the End of the World as we Know it"},{"Losing My Religion"},{"rem - Man on the moon"},{"Radio Free Europe"},{"Radio Song"},{"rem - Shiny Happy People"},{"The One I Love"},{"The Sidewinder Sleeps"},{"What's The Frequency Kenneth"},{"08_ignoreland"},{"REM-The One I Love"},**

```
{"I'll Be There For You"},{"Robbie Williams - Let Me Entertain You"},{"Simply
Irresistable"},{"Robie Williams - Radio"},{"serenate"},{"rolling stones-Anybody
seen my baby"},{"Out_of_control"},{"roxette - she's got the look"},{"Wish I could
Fly"},{"Roy Orbinson - You Got It"},{"Smooth"},{"Savage Garden - 08 - Break
Me Shake Me"},{"In trance"},{"scorpions-hurricane2000"},{"sweet child o
mine"},{"Simple Minds - Don't You"},{"dont you"},{"Simply Red -
Stars"},{"Sunday"},{"soundgarden - Black Hole Sun"},{"Two Princes"},{"Stereo
MCs - Deep Down & Dirty"},{"French Disco"},{"Stereophonics - Just
Looking"},{"Dakota"},{"Sting - Mad About You"},{"SEND YOUR
LOVE"},{"talking heads - 500 Miles"},{"Our Lives"},{"The Killers - Somebody
Told Me"},{"Guilty"},{"The Rasmus - In The Shadows"},{"the roots"},{"The
Smiths - How Soon Is Now"},{"The Best"},{"Twoface - Revelations Day"},{"SLIP
AND SLIDE"},{"TWOFACE - YOU"},{"beautiful day"},{"u2 - Electrical
Storm"},{"everlasting love"},{"u2 - hold me thrill me kiss me kill me"},{"New
Year's Day"},{"sometimes you cant make it on your own"},{"Sunday Bloody
Sunday"},{"Vertigo"},{"u2 - With or without you"},{"Waitin' For A
Superman"},{"No Woman No Cry"},{"Boston - Amanda"},
{"Heartbreak Station"},{"Cutting Crew - Died In Your Arms Tonight"},{"All The
Fools Sailed Away"},{"Guns'N'Roses-civil_war"},{"yesterday"},{"Lou Reed -
Perfect Day"},{"believe_me"},{"Moist-into_everything"},{"Standing Here With
You"},{"queen - The Show must go on"},{"Who Wants To Live For
Ever"},{"REM-Everybody Hurts"},{"Broken"},{"Still Haven't Found What I'm
L"},{"Aerosmith - Dream On"},{"Dream On"},{"Aerosmith -
Spiderman"},{"Walk This Way"},{"alan parsons-eye in the sky"},{"kari
anne"},{"asia-only time will tell"},{"Hazy Shade Of Winter"},{"billy idol - rebel
yell"},{"white_wedding"},{"Black Sabbath - Paranoid"},{"Smoke on the
Water"},{"Blind Guardian - The Bard's Song"},{"dancing in the dark"},{"I drove
all night"},{"City - Am Fenster"},{"Cure - LoveSong"},{"Sultans Of
Swing"},{"Eric Clapton-Cocaine"},{"Layla"},{"Wonderful Tonight"},{"Foreigner-
Cold as Ice"},{"Housemartins - Bow Down"},{"Candy"},{"Iggy Pop - The
Passenger"},{"Fear of the dark"},{"Judas Priest - Caviar And Meths"},{"Dying To
Meet You"},{"Nothing Else Matters"},{"nirvana-Come_as_you_are"},{"Breaking
ice"},{"Police - Every Breath You Take"},{"Kill The King"},{"Rolling Stones -
Angie"},{"Scorpions -Love is blind"},{"Send Me An Angel"}};  // 300 songs

void create_chord_ring(void);
void create_fingers(void);
void successor(int *);
void profile(void);
void hash(char [][100],int *);
void search();


struct fing{
        struct komvos *table[Mbits];
        };
```

```
typedef struct komvos{
        int id;
        int mode;
        int key[15];
        struct fing finger;
        struct komvos *right;
        }NODE;

NODE *node_position[MAXID];      //struct pointer array that points to each node
NODE *head,*current,*tail,*point;

main()
{
  create_chord_ring();
  create_fingers();
  hash(data,datakey);
  successor(datakey);
  profile();
  search();
}

void create_chord_ring()  //dimiourgia tou chord
{
    int temp;
    FILE *fp1;
    head =(NODE *)malloc(sizeof(NODE));     //creating 1st node
    current = head;
    for(j=0;j<15;j++)
    {
       current->key[j] = 0;
    }
    current->id = 0;
    current->mode = off;
    current->right = NULL;
    node_position[0]=head;

    for(i=1; i<MAXID; i++)     //creating all MAX-1 nodes
    {
       current->right = (NODE*)malloc(sizeof(NODE));
       current = current->right;
       node_position[i]=current;
       current->id = i;
       current->mode = off;
       for(j=0;j<15;j++)
```

```
    {
        current->key[j] = 0;
    }
    current->right = NULL;
}

fp1=fopen("Create_ChordRing","w") ;

for(i=0; i<MAX; i++)
{
    pin[i] = rand()%MAXID;
}


for(i=0; i<MAX; i++)
{
    for(j=i+1; j<MAX; j++)
    {
        if(pin[i]>pin[j])
        {
            temp = pin[j];
            pin[j] = pin[i];
            pin[i] = temp;
        }
    }
}


for(i=0; i<MAX; i++)
{
    current=head;
    if(pin[i]==pin[i+1])
    {
        pin[i+1]+=1;
    }
    for(j=0; j<pin[i]; j++)
    {
      current = current->right;
    }
    current->mode = on;
}

current = head;

for(i=0; i<MAXID; i++)        //NODE COUNTER = 1590
{
```

```
    if(current->mode==0)
    {
    }

    else
    {
       fprintf(fp1,"Node[%d] is %d \n",current->id,current->mode);
    }
    current = current->right;
   }

fclose(fp1);
}



void create_fingers()       //creating fingers
{
   double y;
   int k,m;
   FILE *fp2;
   fp2=fopen("Create_Fingers","w");
   current = head;

   for(i=0; i<1590; i++)     //scaning all nodes from 0 to MAXID
   {
      while(current->mode==0)
      {
         current = current->right;
      }

      fprintf(fp2,"Node  id[%d]\n\n", current->id);

        for(k=0; k<Mbits; k++)      // each node has 'Mbits' fingers
        {
            y = pow(2,(k+1)-1);   // 2^(i-1)
            m = current->id+(int)y;        // node + 2^(i-1)

              if(m < MAXID)
              {
                    if(m <= 7998)
                    {
                       while(node_position[m]->mode==0)
                       {
                          m++;
                       }
```

```
                    current->finger.table[k] = node_position[m] ;
                }
                else
                current->finger.table[k] = node_position[21] ;

                fprintf(fp2,"finger[%d] ---> node id[%d]\n",k,current-
>finger.table[k]->id);
            }

            else if(m >= MAXID && m <= capacity)
            {
                current->finger.table[k] = node_position[21] ;
                fprintf(fp2,"finger[%d] ---> node id[%d]\n",k,current-
>finger.table[k]->id);
            }

            else if (m > capacity)
            {
                m = m - capacity;

                while(node_position[m]->mode==0)
                {
                    m++;
                }

                current->finger.table[k] = node_position[m] ;
                fprintf(fp2,"finger[%d] ---> node id[%d]\n",k,current-
>finger.table[k]->id);
            }

        }

       fprintf(fp2,"\n**************************\n\n");
       current=current->right;

   }


fclose(fp2);

}


void hash(char sdata[][100],int *sdatakey)   //metatropi olou tou pinaka data se key
{
```

```
int result=0;
 FILE *fp3;
 fp3=fopen("Hash","w");

 current = head;

 for(j=0; j<num; j++)
 {
    fprintf(fp3,"The song '");
    result = 0;
    for(i=0; i<100; i++)
    {
       if(sdata[j][i]!='\0')
       {
          fprintf(fp3,"%c",sdata[j][i]);
          result = (result+sdata[j][i]);
       }
    }

     sdatakey[j] = result;
     fprintf(fp3,"' has key: %d\n\n",sdatakey[j]);
 }

 fclose(fp3);
}


void successor(int *sdatakey)   //topothethsh ton key stous nodes
{
 FILE *fp6;
 fp6=fopen("Successor","w");

 i=0;
 for(j=0; j<num; j++)
 {
        current = head;
        while(current->id < sdatakey[j])
        {
           current = current->right;
        }
        while(current->mode!=1)
        {
           current = current->right;
        }
        if(current->key[i]==0)
        {
```

```
            current->key[i] = sdatakey[j];
            fprintf(fp6,"Key = %d --->   stored in --->  node id[%d]\n",current-
>key[i],current->id);
         }
         else
         {
            i++;
            current->key[i] = sdatakey[j];
            fprintf(fp6,"Key = %d --->   stored in --->  node id[%d]\n",current-
>key[i],current->id);
         }


  }

 fclose(fp6);
}



void profile()  //emfanizei ta key pou exei o kathe node
{

 FILE *fp7;
 fp7=fopen("Profile","w");
 int t=0;
 current = head;

   for (i=0; i<MAXID; i++)
   {
         if(current->mode==0)
         {
          current=current->right;
         }

         else
         {
            fprintf(fp7,"************\n\nNode[%d]\nId : %d\n",t,current->id);
            t++;

            for(j=0;j<10;j++)
            {
               if(current->key[j]!=0)
               fprintf(fp7,"Key : %d\n",current->key[j]);
            }
            current = current->right;
```

```
        }
    }

 fclose(fp7);
}



void search()  //psaxnei to song pou zitise kapoios node
{
 int k=0,s,v=0,result=0,m,n,b,a=1,d,min=0,max;
 FILE *fp8;
 fp8=fopen("Search","w");

 point = head;

 for(s=0; s<100; s++)
 {
    if(song[s]!='\0')
    result = result+song[s];
 }

 fprintf(fp8,"Node[%d] request the song '%s' with key
%d!\n\n",sNode,song,result);

 for(i=0; i<num; i++)  //elexgoume an iparxei to song sto chord
 {
    if(datakey[i]==result)
    {
       fprintf(fp8,"The song '%s' exists!\n\n",song);
       break ;
    }
    v++;
 }

 if(v==num)  //elexgoume an iparxei to song sto chord
 {
    fprintf(fp8,"The song '%s' does not exist!\n\n",song);
    fprintf(fp8,"Search again for another song!");
    return ;
 }

 while(point->id !=result) //pigainei ton point ston node->id = result
 {
    point = point->right;
 }
```

```
while(point->mode!=1)  //pigainei ton point ston node pou exei to song
{
   point = point->right;
}
fprintf(fp8,"The Node[%d] has the song!\n\n",point->id);


current = head;
while(current->id!=sNode)  //pigainei ton current ston node pou zitaei
{
   current = current->right;
}



while(current->id!=point->id)  //kanoume anazhthsh mexri na vroume ton node
pou psaxnoume
{
   for(b=0; b<Mbits; b++)
   {
      if(current->finger.table[b]->id == point->id)
      {
         current = current->finger.table[b];
         fprintf(fp8,"We are in Node[%d]!\n",current->id);
         fprintf(fp8,"We made %d steps to find the song!!!\n",a);
         return ;
      }
   }

   for(b=0; b<Mbits; b++)
   {
      if(b == 12)
      {
         if(point->id > current->finger.table[11]->id &&  point->id < current-
>finger.table[12]->id)
         {
               current = current->finger.table[11];
               fprintf(fp8,"We are in Node[%d]!\n",current->id);
               a++;
               break;
         }

         else
         {
               current = current->finger.table[12];
               fprintf(fp8,"We are in Node[%d]!\n",current->id);
```

```
                a++;
                break;
            }
        }

        else
        if(point->id > current->finger.table[b]->id &&  point->id < current-
>finger.table[b+1]->id)
        {
                current = current->finger.table[b];
                fprintf(fp8,"We are in Node[%d]!\n",current->id);
                a++;
                break;
        }



    }

 }

fclose(fp8);
}
```

11. Παράρτημα

1) Ευχαριστίες

Για την εκπόνηση της πτυχιακής μας εργασίας θα πρέπει να ευχαριστήσουμε θερμά την επιβλέπουσα καθηγήτρια κ.Π.Φραγκοπούλου για τη σημαντική βοήθεια που μας προσέφερε κατά τη διάρκεια της εργασίας αυτής.

2) Βιβλιογραφια

[1] How Napster Works : http://www.napster.com/faq/

[2] How Kazaa works : http://www.kazaa.com/us/help/index.htm

[3] An Improved Resource Discovery Algorithm for Gnutella Networks
Yu-Tang Guo, Wan-Li Lv , *Bin Luo*

[4] A Simple Fault Tolerant Distributed Hash Table
Moni Naor Udi Wieder

[5] Distributed k-ary System: Algorithms forDistributed Hash Tables
Ali Ghodsi Royal Institute of Technology

[6] Chord and Symphony: An Examination of Distributed Hash Tables
Monica Haladyna Braunisch
Harvard University June 2006

Chord Project Overview : http://pdos.csail.mit.edu/chord/

[7] Pastry: Scalable, decentralized object location and routing for large-scale p2p systems
Antony Rowstron1 and Peter Druschel

[8] Kademlia: A Peer-to-peer Information System Based on the XOR Metric
PetarMaymounkov and David Mazieres

[9] Chord: A Scalable Peertopeer Lookup Service for Internet Applications
Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan
In *Proceedings of the ACM SIGCOMM '01 Conference*,San Diego, California, August 2001.

Efficient Broadcast in P2P Grids
Peter Merz, Katja Gorunova
Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 1 - Volume 01

Looking Up Data in P2P Systems
Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica

Designing a Super-Peer Network
Beverly Yang and Hector Garcia-Molina