ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Παπαστεφάνου Φλώρα

«Υλοποίηση προσωμοιωτή διαδικτυακού συστήματος ομότιμων κόμβων με χρήση κεντρικοποιημένου καταλόγου»

"Implementation of a peer-to-peer Internet file sharing system with a centralized directory service"

Επιβλέπων καθηγητής

κ.Αντώνιος Αργυρίου

Βόλος, 10 Οκτωβρίου 2013

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.

*Στο **Α' μέρος** της παρούσας εργασίας θα περιγράψουμε δομημένα και αδόμητα συστήματα,θα δώσουμε κάποια παραδείγματα τέτοιων συστημάτων και θα αναλύσουμε τα χαρακτηριστικά και τον τρόπο λειτουργίας τους, καθώς και τις σημαντικότερες διαφορές μεταξύ τους.*
*Στο **Β' μέρος** της εργασίας θα παρουσιάσουμε μια εφαρμογή προσωμοιωτή διαδικτυακού*
συστήματος ομότιμων κόμβων με χρήση κεντρικοποιημένου καταλόγου,η οποία είναι γραμμένη σε γλώσσα Java.

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 first** part of this study, we will describe structured and unstructured systems, give some examples of such systems and analyze their characteristics and how they work, and the major differences between them.*
*In **the second** part of this study, we will present an implementation of a peer-to-peer Internet file sharing system with a centralized directory service which is written in the programming language Java.*

Ευχαριστίες

Με την ευκαιρία της ολοκλήρωσης της διπλωματικής μου εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Αντώνιο Αργυρίου για την πολύτιμη καθοδήγησή του καθ' όλη τη διάρκειά της. Επίσης, ένα μεγάλο ευχαριστώ στην οικογένεια και τους φίλους μου, οι οποίοι βοήθησαν με τον τρόπο τους όλον αυτόν τον καιρό της εκπόνησης της διπλωματικής εργασίας, αλλά και των σπουδών μου γενικότερα.

Περιεχόμενα

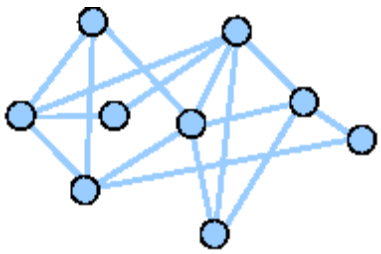
Β’ ΜΕΡΟΣ- ΠΡΟΣΟΜΟΙΩΣΗ ΣΥΣΤΗΜΑΤΟΣ NAPSTER

Α' ΜΕΡΟΣ
ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

1.ΕΙΣΑΓΩΓΗ

Όπως είναι γνωστό, στο διαδίκτυο αρχικά χρησιμοποιήθηκε το αρχιτεκτονικό μοντέλο του **εξυπηρετούμενου – εξυπηρέτη (client - server)**. Σε αυτό, τα δεδομένα αποθηκεύονται στους εξυπηρέτες. Για την εύρεση κάποιας πληροφορίας ο χρήστης (εξυπηρετούμενος) εντοπίζει τον εξυπηρέτη που κατέχει την επιθυμητή πληροφορία και κάνει μια αίτηση για να την αποκτήσει. Κατόπιν, ο εξυπηρέτης ανταποκρίνεται παρέχοντας, συνήθως, την ίδια την πληροφορία. Τις περισσότερες φορές υπάρχει μικρός αριθμός εξυπηρετών που παρέχουν υπηρεσίες στους υπόλοιπους χρήστες. Οι εξυπηρέτες αυτοί είναι συνήθως διαθέσιμοι για μεγάλο χρονικό διάστημα, ενώ οι εξυπηρετούμενοι παραμένουν συνδεδεμένοι για λίγο.

Στα **P2P συστήματα** η παραπάνω αρχιτεκτονική εγκαταλείπεται. Οι χρήστες δρουν και ως εξυπηρετούμενοι και ως εξυπηρέτες, δηλαδή αποθηκεύουν, ανακτούν και συνεισφέρουν στην εύρεση πληροφοριών που αναζητούνται σε ένα δυναμικό σύστημα. Επίσης, η αποθήκευση και διαχείριση των δεδομένων γίνεται με διαφορετικό τρόπο. Κάθε φορά που ένας νέος κόμβος εισέρχεται στο σύστημα, τα αρχεία που επιθυμεί να διαμοιραστεί είναι αμέσως διαθέσιμα στους υπόλοιπους χρήστες. Η πληροφορία του συστήματος, λοιπόν, αποτελείται
από τις επιμέρους πληροφορίες που βρίσκονται αποθηκευμένες σε κάθε κόμβο.



Σχήμα 1: P2P δίκτυο

Το μοντέλο αυτό, όμως, γεννά δύο σημαντικά ζητήματα: ποιος μηχανισμός αναζήτησης θα εφαρμοστεί προκειμένου να είναι δυνατή η αποτελεσματική εύρεση των επιθυμητών πληροφοριών και πώς θα γίνει τελικά η μεταφορά των δεδομένων. Στα P2P συστήματα προτείνονται δύο λύσεις: κεντρικοποιημένη και αποκεντρικοποιημένη αναζήτηση.

Τα **κεντρικοποιημένα** συστήματα (Napster), αποδεικνύονται ιδιαίτερα ευάλωτα καθώς υπάρχει μοναδικό σημείο αποτυχίας, κατάρρευση του οποίου μπορεί να οδηγήσει σε κατάρρευση ολόκληρου του συστήματος. Επίσης, η απόδοσή τους είναι μικρή όταν το πλήθος των κόμβων αυξηθεί πολύ.

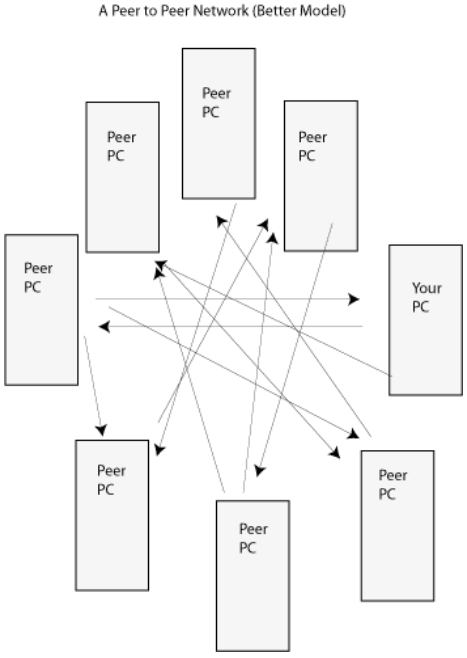
Τα μειονεκτήματα αυτά έρχονται να καλύψουν **αποκεντρικοποιημένα** συστήματα, όπως είναι τα δίκτυα Gnutella, Freenet (*αδόμητα*, όπου η σύνδεση των κόμβων ακολουθεί χαλαρούς κανόνες), τα Chord, CAN(*δομημένα*, όπου οι κόμβοι συνδέονται με αυστηρό τρόπο και δημιουργείται ένα ιδεατό δίκτυο για την πραγματοποίηση της δρομολόγησης) και άλλα. Φυσικά, σε κάθε ένα από αυτά χρησιμοποιούνται διαφορετικές τεχνικές αποθήκευσης και διαχείρισης των δεδομένων.

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.

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.

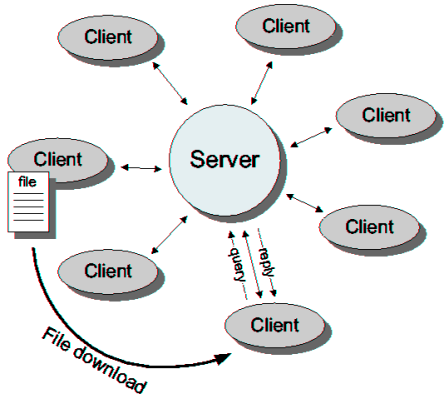
Τα **Peer-to-Peer** (Σχ. 2.1) δίκτυα χρησιμοποιούνται χαρακτηριστικά για τη σύνδεση των κόμβων μέσω των κατά ένα μεγάλο μέρος ειδικών συνδέσεων. Τέτοια δίκτυα είναι χρήσιμα για πολλούς λόγους. Μοιράζουν τα αρχεία που περιέχουν τους ήχους, τα βίντεο, τα δεδομένα ή οτιδήποτε είναι σε ψηφιακή μορφή κάτι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



Σχήμα 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.



Σχήμα 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 Peerto-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.

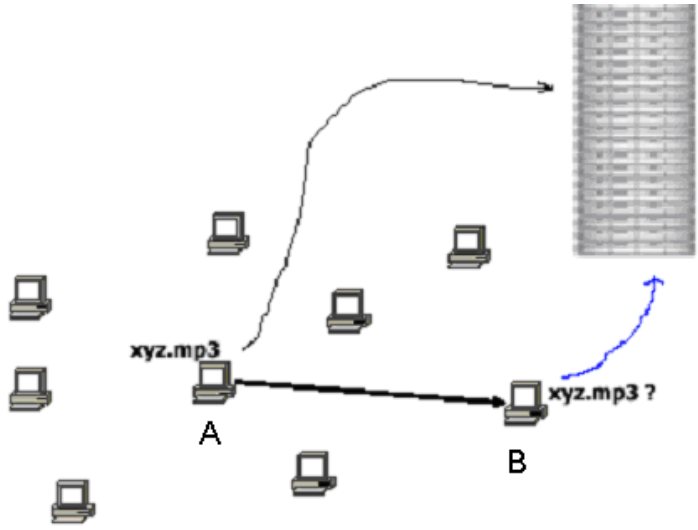
•Ιεραρχική αρχιτεκτονική: Η τρίτη αρχιτεκτονική είναι ένας συνδυασμός των δύο παραπάνω. Υπάρχουν κάποιοι κόμβοι που είναι περισσότερο σημαντικοί από τους υπόλοιπους (υπερκόμβοι). Το **Kazaa** χρησιμοποιεί υπερκόμβους στο δίκτυο του.

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

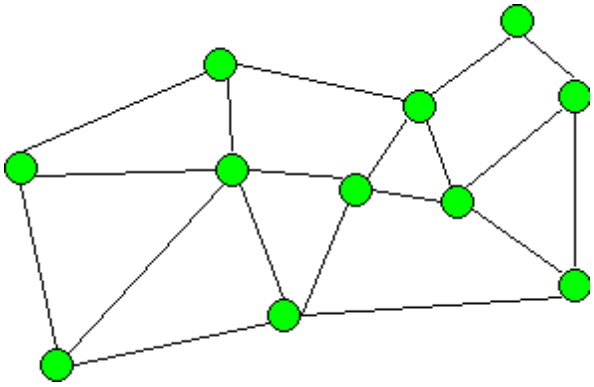
Το **Napster** βοήθησε πολύ ώστε τα **P2P** δίκτυα να γίνουν ευρέως γνωστά και να αποτελέσουν μία από τις πιο γρήγορα εξελισσόμενες εφαρμογές του διαδικτύου, νομικά προβλήματα όμως ήταν η αιτία να εγκαταλειφθούν όλα τα δίκτυα αυτής της κατηγορίας.
Στα πλεονεκτήματά τους συμπεριλαμβάνονται η αποδοτικότητα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

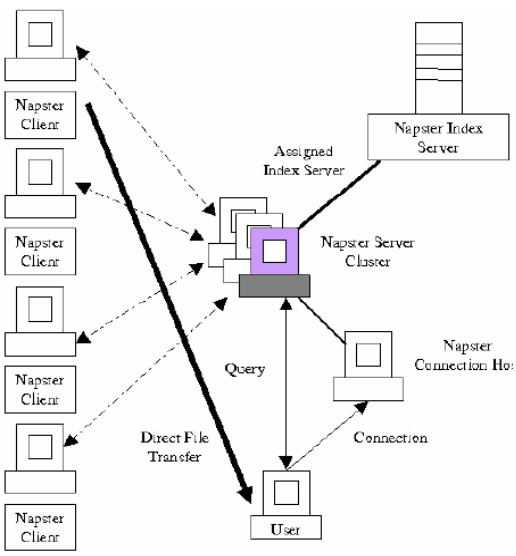
Η ιστορία των ομότιμων (P2P) δικτύων αρχίζει συχνά με μια αναφορά στο Napster. Είναι το δημοφιλέστερο παράδειγμα για τη δυνατότητα πραγματοποίησης επικοινωνίας με χρήση υπολογιστών και ενός προς έναν και ενός προς πολλούς.
Το Napster είναι μια P2P εφαρμογή διαμοιρασμού αρχείων που επιτρέπει στους χρήστες να ψάξουν και να μοιραστούν MP3 αρχεία μουσικής μέσω του Διαδικτύου. Δημιουργήθηκε από έναν έφηβο ονόματος Shawn Fanning (Tyson, 2000, Shirky, 2001). Όχι μόνο ανέπτυξε την εφαρμογή αλλά καινοτόμησε σχεδιάζοντας πρωτόκολλο που επέτρεπε στους peer υπολογιστές να επικοινωνήσουν ο ένας με τον άλλον άμεσα. Αυτό προετοίμασε το έδαφος για αποδοτικότερα και σύνθετα P2P πρωτόκολλα από άλλους οργανισμούς και ομάδες.

Η πρώτη έκδοση κυκλοφόρησε τον Ιανουάριο του 1999 και το 2000 το χρησιμοποιούσαν 80 εκατομμύρια χρήστες. Στα μέσα του 2001 βγήκε εκτός λειτουργίας λόγω δικαστικών αγωγών από δισκογραφικές εταιρίες, ενώ το 2003 επανιδρύθηκε ως υπηρεσία με πληρωμή.

Η αρχιτεκτονική του Napster, που παρουσιάζεται στο σχήμα, είναι βασισμένη στο κεντρικοποιημένο πρότυπο του P2P διαμοιρασμού αρχείων. Έχει δομή πελάτη-εξυπηρετητή όπου υπάρχει ένα κεντρικό σύστημα εξυπηρετητών που κατευθύνει την κυκλοφορία μεταξύ των μεμονωμένων εγγεγραμμένων χρηστών. Οι εξυπηρετητές διατηρούν καταλόγους των κοινών αρχείων, που αποθηκεύονται στα αντίστοιχα PC των εγγεγραμμένων χρηστών του δικτύου. Αυτοί οι κατάλογοι ενημερώνονται κάθε φορά που συνδέεται ή αποσυνδέεται ένας χρήστης στο δίκτυο του εξυπηρετητή του Napster.

Οι πελάτες συνδέονται αυτόματα σε έναν εσωτερικό «μεταεξυπηρετητής» (metaserver), που λειτουργεί ως κοινός κριτής σύνδεσης. Αυτός ο μεταεξυπηρετητής ορίζει τυχαία έναν διαθέσιμο, ελαφριά φορτωμένο εξυπηρετητή από τις συστάδες. Οι κεντρικοί εξυπηρετητές εμφανίζονται να είναι κατηγοριοποιημένοι σε πεντάδες σε μια γεωγραφικές περιοχή και ικανοί να χειριστούν μέχρι 15.000 χρήστες ο καθένας. Ο πελάτης καταχωρείται έπειτα στον ορισμένο εξυπηρετητή, παρέχοντας την ταυτότητα και πληροφορίες των διαμοιραζόμενων αρχείων για την τοπική βάση δεδομένων του εξυπηρετητή. Στη συνέχεια, ο πελάτης λαμβάνει πληροφορίες για τους συνδεδεμένους χρήστες και τα διαθέσιμα αρχεία από τον εξυπηρετητή. Αν και οργανώνεται τυπικά γύρω από μία σχεδίαση καταλόγου χρηστών, η εφαρμογή Napster είναι πολύ στοιχείο-κεντρική. Ο αρχικός κατάλογος των χρηστών που συνδέεται σε έναν ιδιαίτερο εξυπηρετητή χρησιμοποιείται μόνο έμμεσα, για να δημιουργήσει τις λίστες αρχείων περιεχομένου αναφερόμενα ως διαμοιραζόμενα από τον κάθε κόμβο (Barkai, 2001).

Οι χρήστες είναι σχεδόν πάντα ανώνυμοι ο ένας με τον άλλο: ο κατάλογος χρηστών ποτέ δεν ζητείται άμεσα. Το μόνο που γίνεται είναι να αναζητηθεί για το περιεχόμενο και να καθορίσει για έναν κόμβο από που θα λάβει αυτό που ζήτησε. Κάθε φορά που ένας χρήστης ενός P2P συστήματος κεντρικής οντότητας διαμοιρασμού αρχείων υποβάλλει ένα αίτημα ή μια αναζήτηση ενός συγκεκριμένου αρχείου, ο κεντρικός εξυπηρετητής δημιουργεί έναν κατάλογο αρχείων ταιριάζοντας το αίτημα αναζήτησης, με επαλήθευση του αιτήματος, με τη βάση δεδομένων του εξυπηρετητή των αρχείων που ανήκουν στους χρήστες που είναι συνδεδεμένοι αυτήν την στιγμή στο δίκτυο. Ο κεντρικός εξυπηρετητής επιδεικνύει έπειτα εκείνο τον κατάλογο στο χρήστη που το ζητάει. Αυτός ο χρήστης μπορεί έπειτα να επιλέξει το επιθυμητό αρχείο από την λίστα και να ανοίξει μια άμεση σύνδεση HTTP με το μεμονωμένο υπολογιστή που κατέχει αυτήν τη στιγμή εκείνο το αρχείο. Η λήψη του πραγματικού αρχείου πραγματοποιείται άμεσα, από τον έναν χρήστη στον άλλο, χωρίς την επέμβαση του κεντρικού εξυπηρετητή. Το πραγματικό MP3 αρχείο δεν αποθηκεύεται ποτέ στον κεντρικό υπολογιστή ή σε οποιοδήποτε ενδιάμεσο σημείο του δικτύου.



Απεικόνιση αρχιτεκτονικής Napster

Το πρωτόκολλο Napster

Εξαιτίας του γεγονότος ότι το Napster δεν είναι μια ανοικτού κώδικα εφαρμογή, ήταν μόνο δυνατό να δημιουργηθεί μια παρόμοια εφαρμογή για να αποκαλυφθεί το πρωτόκολλο Napster, μέσω της αντίστροφης εφαρμοσμένης μηχανικής (Turcan, 2002). Με άλλα λόγια, κανένας δεν θα είναι ποτέ ολοκληρωτικά βέβαιος για το ποιες είναι οι προδιαγραφές του πρωτοκόλλου Napster, εκτός από τον ίδιο το δημιουργό του.
Το πρόγραμμα OpenNap κατέστησε δυνατή την εκτέλεση ενός κεντρικού εξυπηρετητή Napster σε πολλές πλατφόρμες χωρίς χρησιμοποίηση της αρχικής εφαρμογής Napster και του κεντρικού εξυπηρετητή δεικτών. Τα ακόλουθα είναι προδιαγραφές πρωτοκόλλου για το Napster σε σχέση με τον Turcan (2002).

Το Napster λειτουργεί με έναν κεντρικό εξυπηρετητή που διατηρεί ένα ευρετήριο όλων των MP3 αρχείων των peers. Για να πάρει κάποιος ένα αρχείο, πρέπει να στείλει μια ερώτηση σε αυτόν τον κεντρικό εξυπηρετητή, ο οποίος θα στείλει πίσω την θύρα (port) και τη διεύθυνση IP ενός πελάτη, που μοιράζεται το ζητούμενο αρχείο. Με την εφαρμογή Napster, είναι τώρα δυνατό να εγκατασταθεί μια άμεση σύνδεση με τον host (οικοδεσπότη) και να ληφθεί ένα αρχείο.

Το πρωτόκολλο Napster χρησιμοποιεί επίσης πάρα πολλούς διαφορετικούς τύπους μηνυμάτων. Κάθε κατάσταση των hosts, που ενεργούν όπως οι πελάτες με τον εξυπηρετητή, συσχετίζεται με τον κεντρικό εξυπηρετητή Napster. Το πρωτόκολλο Napster ήταν κλειστό, έτσι ώστε κανείς να μην ξέρει σίγουρα πως η αναζήτηση και η μεταφορά του αρχείου γινόταν. Έτσι όταν το Napster εισήχθη αρχικά, υπήρξε μόνο μια εφαρμογή πελατών, η οποία κλήθηκε Napster.Το Napster εστιάζει

αποκλειστικά στα MP3-κωδικοποιημένα αρχεία μουσικής. Αν και κανένας άλλος τύπος αρχείου δεν υποστηρίχθηκε, μια υποομάδα κλώνων πελατών και εργαλείων προέκυψε σύντομα, κατασκευασμένη από τους πελάτες του κλειστού-κώδικα Napster.

Η πρόθεση πίσω από την ανάπτυξη επρόκειτο να έχει μεγαλύτερο έλεγχο χρηστών. Για παράδειγμα, μια υποκρινόμενη μορφή MP3 από εργαλεία όπως το Wrapster εφαρμόστηκε στους κεντρικούς εξυπηρετητές του Napster, και μπορούσε να τυλίξει ένα αυθαίρετο αρχείο με ένα είδος περιτυλίγματος που το έκανε να μοιάζει με ένα MP3 αρχείο. Θα εμφανιζόταν έπειτα στις βάσεις δεδομένων των κεντρικών εξυπηρετητών και θα ήταν εξερευνήσιμο από άλλους πελάτες που επιθυμούσαν να λάβουν κι άλλης μορφής αρχεία εκτός από μουσικά.

Ένα εμπόδιο σε αυτήν την προσπάθεια ήταν ότι το πεδίο περιγραφής καλλιτέχνη-τίτλου επέτρεψε λίγες πληροφορίες για το μη-μουσικό αρχείο. Αυτό, η χαμηλή αναλογία μη- μουσικών και μουσικών αρχείων και η κανονική τυχαία κατανομή των συνδεδεμένων κόμβων συνωμότησαν στο να καταστήσουν τις περιορισμένες αναζητήσεις του Napster απίθανο το να βρουν ειδικό περιεχόμενο. Εργαλεία όπως το Napigator, αναπτύχθηκαν για να επιτρέψουν στους χρήστες να συνδεθούν με τους συγκεκριμένους κεντρικούς εξυπηρετητές, παρακάμπτοντας τη διαιτησία του metaserver. Κατά αυτόν τον τρόπο, ορισμένοι κεντρικοί εξυπηρετητές έγιναν γνωστοί ως Wrapster-hangouts για το μη-μουσικό περιεχόμενο. Οι χρήστες που έψαχναν αυτό το είδος περιεχομένου ήταν πιθανότερο να το βρουν.

Οι ανταλλαγές μη μουσικών αρχείων, πέρα από Napster, δεν ήταν ποτέ περισσότερο από οριακές, τουλάχιστον συγκρινόμενες με εναλλακτικά, αγνώστου περιεχομένου συστήματα όπως το Gnutella. Μερικοί εναλλακτικοί εξυπηρετητές Napster, όπως το OpenNap που άρχισε ως «ασφαλή καταφύγιο» για τους χρήστες του Napster, όταν το αυτό άρχισε να φιλτράρει το περιεχόμενο, για μια στιγμή άρχισαν να καλύπτουν το κενό που ενώνει τους προηγούμενους πελάτες του Napster, τους κλώνους και τις παραλλαγές, με ένα νέο είδος κεντρικού εξυπηρετητή που επέκτεινε το αρχικό πρωτόκολλο Napster σε όλους τους τύπους αρχείου.

Παρά την αναπροσαρμογή του Napster, η θεμελιώδης απαίτηση ενός συμβατού κεντρικού εξυπηρετητή με το Napster παρέμεινε ένας σοβαρός περιορισμός για ένα δίκτυο βασισμένο σε αυτήν την τεχνολογία ή σε οποιονδήποτε από τους κλώνους του. Για να ξεπεραστεί αυτός ο περιορισμός, χρειάσθηκαν άλλα πρωτόκολλα και πρότυπα αρχιτεκτονικής, παραδείγματος χάριν, δίκτυα μικρότερης εξάρτησης από εξυπηρετητή, του τύπου Gnutella.

Gnutella

Αρχές Μαρτίου του 2000, το Gnutella δημιουργήθηκε από τους Justin Frankel και Tom Peper, οι οποίοι και οι δύο δουλεύουν το Gnullsoft, το οποίο είναι ένα από τα υποκαταστήματα της AOL.
Η ανάπτυξη του Gnutella σταμάτησε αργότερα από τη AOL, αμέσως μόλις δημοσιεύθηκε, αλλά η σύντομη διάρκεια όπου το Gnutella λειτούργησε, ήταν αρκετή για να επιτρέψει σε περίεργους προγραμματιστές να το «κατεβάσουν» και στη συνέχεια να επεξεργαστούν το πρωτόκολλο επικοινωνίας του Gnutella. Κατά συνέπεια, διάφοροι κλώνοι του Gnutella με βελτιώσεις εισήχθησαν (παραδείγματος χάριν, LimeWire, BearShear, Gnucleus, XoloX, και Shareaza) (Ding, Nutanong, Buyya 2005).

Η αρχιτεκτονική του Gnutella

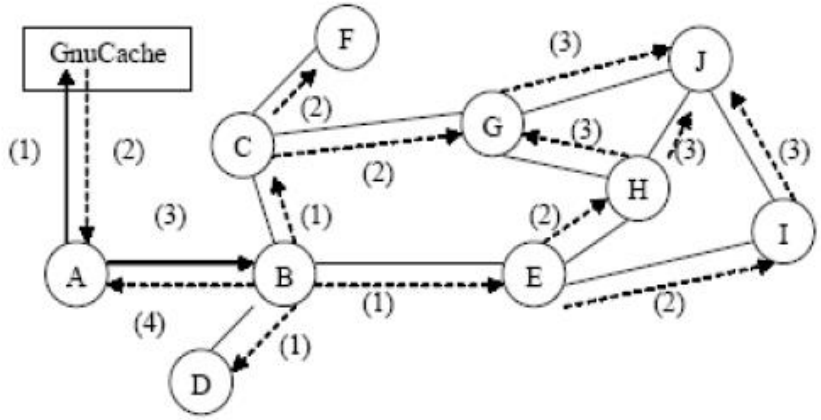
Αντί της διατήρησης ενός κεντρικού καταλόγου δεικτών (ευρετήριο) όπως κάνει το Napster, το Gnutella χρησιμοποιεί ένα επίπεδο δίκτυο από peers που καλούνται servents, για να διατηρήσουν τον κατάλογο δεικτών όλου του περιεχομένου στο σύστημα.
Σε ένα δίκτυο Gnutella, οι servents συνδέονται ο ένας με τον άλλον σε μια επίπεδη ad hoc τοπολογία. Ο servent λειτουργεί και ως πελάτης και ως εξυπηρετητής. Ως εξυπηρετητής, αποκρίνεται στις ερωτήσεις ενός άλλου peer servent. Ως πελάτης, προωθεί τις ερωτήσεις σε άλλους peer servents.

Ένας servent για να ενωθεί στο δίκτυο Gnutella, πρέπει να βρει τη διεύθυνση ενός servent που είναι ήδη συνδεδεμένος στο δίκτυο. Αυτό μπορεί να γίνει με τη χρησιμοποίηση των hosts caches (κρυφές μνήμες οικοδεσποτών), όπως τη GnuCache, οι οποίες αποθηκεύουν τους Gnutella servents (hosts) που είναι πάντα συνδεδεμένοι στο δίκτυο Gnutella. Αφότου βρεθεί μια διεύθυνση, στέλνεται έπειτα ένα μήνυμα αιτήματος GNUTELLA ΣΥΝΔΕΣΗ στον ήδη- συνδεμένο servent. Ο ζητούμενος servent μπορεί είτε να δεχτεί το αίτημα και να στείλει ένα μήνυμα απάντησης GNUTELLA OK, είτε να απορρίψει το μήνυμα αιτήματος με την αποστολή οποιασδήποτε άλλης απάντησης πίσω στον αιτών servent. Μια απόρριψη μπορεί να συμβεί εξαιτίας διαφορετικών λόγων, όπως εξαιτίας κενών σύνδεσης που έχουν διαφορετικές εκδόσεις πρωτοκόλλου, και ούτω καθεξής (Limewire). Κάποιος συνδεμένος με το δίκτυο servent θορυβεί (ping) περιοδικά τους γείτονές του για να ανακαλύψει άλλους servents. Χαρακτηριστικά, κάθε ένας servent πρέπει να συνδεθεί με περισσότερους από έναν servent, δεδομένου ότι το δίκτυο Gnutella είναι δυναμικό, κάτι που σημαίνει ότι ο οποιοσδήποτε servent μπορεί να αποσυνδεθεί οποτεδήποτε.
Είναι έτσι σημαντικό να μείνει σε επαφή με διάφορους servents συγχρόνως, αποτρέποντας, έτσι, την αποσύνδεση από το δίκτυο. Μόλις ένας εξυπηρετητής λάβει το μήνυμα θορύβου- ping, στέλνει πίσω ένα πίσω μήνυμα pong στον

εξυπηρετητή που δημιούργησε το μήνυμα ping χρησιμοποιώντας την ίδια πορεία του μηνύματος ping. Ένα μήνυμα pong περιέχει τις πληροφορίες για τον servent όπως τη θύρα, τη διεύθυνση IP, τον αριθμό διαμοιραζόμενων αρχείων, και τον αριθμό διαμοιραζόμενων kilobytes. Το Gnutella δεν χρησιμοποιεί οποιουσδήποτε εξυπηρετητές καταλόγου, δεδομένου ότι κάθε servent διατηρεί τοπικά τον κατάλογο δεικτών του. Για να ψάξει για ένα αρχείο, ένας κόμβος στέλνει ένα ερώτημα σε όλους τους γείτονές του που συνδέονται άμεσα με αυτόν. Μόλις ληφθεί μια ερώτηση από έναν γείτονα, το κριτήριο ερώτησης ελέγχεται σε σχέση με τον τοπικό κατάλογο δεικτών και το μήνυμα διαδίδεται στη συνέχεια σε όλους τους γείτονές του και ούτω καθεξής.

Εάν ο έλεγχος ταιριάζει με τα τοπικά δεδομένα του servent, ο servent στέλνει πίσω ένα μήνυμα queryHit στο servent που άρχισε την ερώτηση, κατά μήκος της ίδιας πορείας που έφερε το μήνυμα ερώτησης. Εντούτοις, όταν ο servent παράγει το μήνυμα queryHit όντας πίσω από ένα τείχος προστασίας, ο αιτών servent δεν μπορεί να δημιουργήσει μια σύνδεση σε αυτόν. Σε αυτήν την περίπτωση, το μήνυμα προώθησης στέλνεται από τον αιτών servent στο servent που παράγει το μήνυμα queryHit και μένει πίσω από το τείχος προστασίας για να αρχίσει τη σύνδεση αντ' αυτού. Σημειώστε ότι η μεταφορά αρχείων γίνεται χρησιμοποιώντας το πρωτόκολλο HTTP, και όχι το πρωτόκολλο Gnutella.

Δεδομένου ότι το Gnutella εκπέμπει τα μηνύματά του, προκειμένου να αποτραπεί η πλημμύρα του δικτύου με μηνύματα, το πεδίο TTL συμπεριλαμβάνεται στην επιγραφή κάθε μηνύματος Gnutella. Το πεδίο TTL μειώνεται κάθε φορά που το μήνυμα περνάει μέσω ενός servent. Όταν ο servent συναντήσει TTL με τιμή ίση με μηδέν, θα πετάξει εκείνο το μήνυμα. Κάθε servent επίσης πρέπει να διατηρεί μία λίστα πρόσφατα ιδωμένων μηνυμάτων αποθηκεύοντας τον περιγραφέα ταυτότητας (Descriptor ID) και τον περιγραφέα φορτίου (Payload Descriptor) κάθε εισερχόμενου μηνύματος, για να αποτρέψει να διαβιβάσει το ίδιο μήνυμα επανειλημμένα. Μια λεπτομερής περιγραφή του μηνύματος Gnutella και του πρωτοκόλλου θα εξηγηθεί σε επόμενο τμήμα.



Όπως φαίνεται στο σχήμα, υποτίθεται ότι ο χρήστης A που συνδέεται με το δίκτυο θέλει να ψάξει για μερικά αρχεία. Στέλνει ένα μήνυμα ερώτησης στο γείτονά του/της, χρήστη B. Ο χρήστης B πρώτα ελέγχει ότι το μήνυμα δεν είναι παλιό. Ελέγχει έπειτα για αντιστοιχία με τα τοπικά δεδομένα του/της. Εάν υπάρχει αντιστοιχία, στέλνει το μήνυμα queryHit πίσω στο χρήστη A. Ο χρήστης B έπειτα μειώνει το TTL κατά 1 και διαβιβάζει το μήνυμα ερώτησης στους χρήστες C, D, και E. Οι χρήστες C, D, και E εκτελούν τα ίδια βήματα με το χρήστη B και διαβιβάζουν το μήνυμα ερώτησης περαιτέρω στους χρήστες F, G, H, και I, που θα επαναλάβουν πάλι την ίδια διαδικασία.

Υποθέστε ότι ο χρήστης H είναι ο πρώτος που διαβιβάζει το μήνυμα ερώτησης στο χρήστη J. Οποιαδήποτε επόμενα μηνύματα ερώτησης που διαβιβάζονται από τους χρήστες G και I στο χρήστη J θα απορριφθούν δεδομένου ότι οι έλεγχοι στον τοπικό πίνακά του δείχνουν ότι έχει λάβει ήδη εκείνα τα μηνύματα. Αυτό ισχύει και για το χρήστη G επίσης, όταν του διαβιβάσει ο χρήστης H το μήνυμα ερώτησης. Ας υποθέσουμε τώρα ότι ο χρήστης J βρίσκει μια αντιστοιχία στα τοπικά δεδομένα του/της. Τότε αποκρίνεται με την αποστολή ενός μηνύματος queryHit πίσω στο χρήστη A, ακολουθώντας την ίδια πορεία του μηνύματος ερώτησης: από το J, στο H, στο E, στο B και στο A. Τώρα ο χρήστης A μπορεί να αρχίσει τη λήψη του αρχείου άμεσα με το χρήστη J χρησιμοποιώντας το πρωτόκολλο HTTP.

Το πρωτόκολλο Gnutella

Το μήνυμα που χρησιμοποιείται για την επικοινωνία μεταξύ των servents καλείται Gnutella Περιγραφέας (Descriptor). Οι Gnutella descriptors αποτελούνται από τα εξής: Περιγραφέας Κεφαλίδας και Περιγραφέας Φορτίου. Υπάρχουν πέντε τύποι descriptors Gnutella: Ping, Pong, Query (ερώτημα), QueryHit, και Push (ώθηση), όπως αναφέρονται στον πίνακα

Descriptor (Περιγραφέας)	Περιγραφή
Ping	Χρησιμοποιείται για να ανακαλύψει ενεργούς hosts στο δίκτυο. Ένας servent λαμβάνοντας έναν Ping Descriptor αναμένεται να αποκριθεί με έναν ή περισσότερους Pong Descriptor.
Pong	Η απάντηση σε ένα Ping. Περιλαμβάνει τη διεύθυνση ενός συνδεδεμένου Gnutella servent και των πληροφοριών σχετικά με το ποσό δεδομένων που θέτει στην διάθεση του δικτύου.
Query	Ο αρχικός μηχανισμός για αναζήτηση στο δίκτυο. Ένας servent λαμβάνοντας ένα Query Descriptor θα αποκριθεί με ένα QueryHit, εάν βρεθεί αντιστοιχία στο τοπικό σύνολο δεδομένων της.
QueryHit	Η απάντηση σε μια ερώτηση. Αυτός ο περιγραφέας παρέχει στον παραλήπτη αρκετές πληροφορίες για να αποκτήσει τα δεδομένα που ταιριάζουν με την αντίστοιχη ερώτηση.
Push	Ο μηχανισμός που επιτρέπει σ' έναν τειχισμένο servent να συμβάλει στα δεδομένα στο δίκτυο.

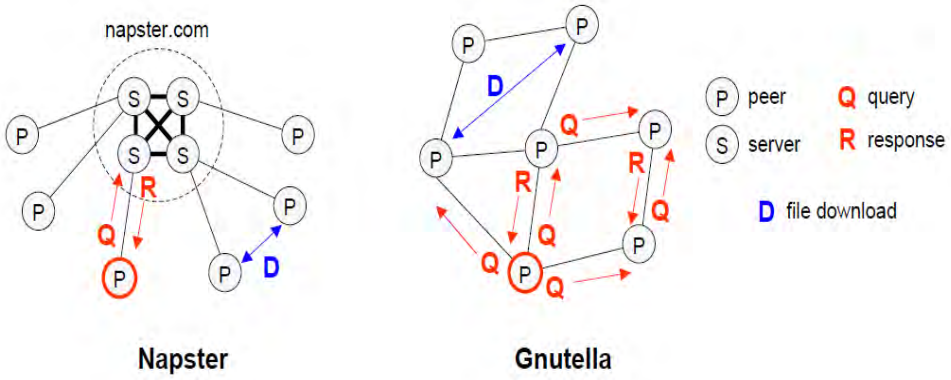


Figure 1: File location in Napster and Gnutella

Συγκριση file Sharing εφαρμογών

Name	Network[1]	Anonymous P2P	Link system compatibility	Price	Platform	License	Programming language	Spyware-adware free	Latest release version	Extra information
Acquisition	gnutella, BitTorrent	No	magnet	Nagware or US$17.99	OS X	GPL	Objective-C, Java	Yes	(November 19, 2010)	Alternative to LimeWire for Mac users; uses Cocoa GUI
aMule	eDonkey, Kad	No	eD2k	Free	Linux, OS X, Windows	GPL	C++	Yes	(November 11, 2011[2])	
ANts P2P	ANts	Yes	ed2k	Free	Linux, OS X, Windows	GPL	Java	Yes	2010	Proxies other users' IP addresses making it hard to determine who is downloading
Ares Galaxy	Ares, BitTorrent	No	arisk	Free	Windows	GPL	Object Pascal (Delphi)	Yes	(February 2, 2012[3])	
BearShare (pre v6.0)	gnutella	No	ed2k, magnet	Free or US$3.29 monthly	Windows	Proprietary	C++	No	(January 11, 2011)	Since version 6: clone of the commercial iMesh software; does not connect to gnutella and restricts downloads to media formats
BitComet	BitTorrent, eDonkey, Kad[4]	No	ed2k,[4] magnet	Free	Windows	Proprietary	C++	Yes	(September 18, 2012)[5]	Exploits SuperSeeding and thus is often banned by trackers and peers. Compatible with eD2K links through an eMule plugin which gives access to eDonkey and Kad networks and cross-network sharing abilities with BitTorrent network.
BitTornado	BitTorrent	No	No	Free	Linux, OS X, Windows	Open source	Python	Yes	2006	
BitTorrent	BitTorrent	No	No	Free	Windows, Mac OS X	Proprietary	C++	Yes	2010	µTorrent rebrand

brootz	Direct Connect	No	No	Free or US$59.95	Windows, OS X, Linux	Proprietary	Java	Yes	2011	Internal p2p protocols that allow only secondary users authorised by primary user to share
Cabos	gnutella	No	magnet	Free	OS X, Windows	GPL	Java, REALbasic	Yes	(February 9, 2010)	
DC++	Direct Connect	No	magnet	Free	Windows	GPL	C++	Yes	0.799 May 5, 2012	Many modifications exist, for example LinuxDC++ [6]
Deluge	BitTorrent	Unknown	magnet	Free	Linux, OS X, Windows	GPL	Python	Yes	(April 9, 2012)	
eDonkey2000	eDonkey, Overnet	No	ed2k	Free or US$19.95	Linux, OS X, Windows	Proprietary	C++	Yes	2005	Discontinued
eMule	eDonkey, Kad	No	ed2k	Free	Windows	GPL	C++	Yes	(April 7, 2010 [7])	Supports protocol obfuscation
ExoSee	?	No	No	Free	Windows	Proprietary	?	Yes	2007	ExoSee appears to have no business model
Free Download Manager	BitTorrent	No	metalink	Free	Windows	GPL	C++	Yes	(May 28, 2012) [8]	
Freenet's FProxy	Freenet	Yes	No	Free	Linux, OS X, Windows	GPL	Java	Yes	(October 30, 2012 [9])	
Frost [10]	Freenet	Yes	No	Free	Linux, OS X, Windows	GPL	Java	Yes	2009	
FrostWire	gnutella, BitTorrent	No	magnet	Free	Linux, OS X, Windows	GPL	Java	Yes	(September 20, 2012)	Fork of LimeWire, with no copyrighted-material blocker planned
giFT	OpenFT and, with plugins, Ares, gnutella, FastTrack	No	No	Free	Linux, OS X, Windows	GPL	C	Yes	(2004-11-27)	
Gnucleus	gnutella, Gnutella2	No	No	Free	Windows	GPL	C++	Yes	(2005-06-17)	
GNUnet	GNUnet	Yes	No	Free	Linux, OS X, Windows	GPL	C	Yes	(November 5, 2012)	
gtk-gnutella	gnutella	No	magnet	Free	Linux, Windows	GPL	C	Yes	(June 4, 2012)	
iMesh (pre v6.0)	FastTrack, gnutella, Gnutella2	No	No	Free	Windows	Proprietary	C++	Yes	2008	

I2Phex	I2P (protocol gnutella)	Yes	magnet	Free	Linux, OS X, Windows	GPL	Java	Yes	2011	need I2P to run
I2PSnark	I2P (protocol BitTorrent)	Yes	magnet	Free	Linux, OS X, Windows	GPL	Java	Yes	2010	Integrated into I2P distribution
iMule	I2P (protocol Kad)	Yes	ed2k	Free	Linux, OS X, Windows	GPL	C++	Yes	(July 2, 2012)	internal I2P is old and buggy, use separate I2P installation
Kazaa	FastTrack	No	No	Free	Windows	Proprietary	C++	adware & spyware	2006	Illegal in Australia.
Kazaa Lite	FastTrack	No	No	Free	Windows	Proprietary	?	Yes	2007	Adware-disabled version of Kazaa (fake versions in circulation)
KCeasy	Ares, gnutella, FastTrack, OpenFT	No	No	Free	Windows	GPL	C++, Object Pascal (Delphi)	Yes	2008	Uses integrated giFT file sharing daemon as back-end
KTorrent	BitTorrent	No	magnet	Free	Linux, OS X, Windows	GPL	C++	Yes	(September 4, 2012[11])	Part of the KDE desktop, can be installed separately
LimeWire	gnutella, BitTorrent	No	magnet	Free or US$18.88	Linux, OS X, Windows	GPL	Java	No	2010	Shutdown permanently due to injunction from MGM v. Grokster in which Limewire was ordered to disable all functionality of the software.[12]
Manolito	MP2P	No	MP2P	Free or US$19.95	Windows	Proprietary	?	No	2008	
MLDonkey	BitTorrent, eDonkey, FastTrack, (gnutella, Gnutella2), Kad	No	ed2k, magnet, sig2dat	Free	Linux, OS X, Windows	GPL	OCaml	Yes	(August 5, 2012[13])	P2P application, telnet / web-interface / GUI interface, complete and complex remote usage. Sancho GUI makes automatic SSH tunnels.
MonoTorrent (client library)	BitTorrent	No	magnet	Free	Linux, OS X, Windows	MIT/X11	C#	Yes	(May 22, 2010)	Requires either Mono or .NET Framework to be installed
Morpheus	NEOnet, gnutella, Gnutella2, BitTorrent	No	No	Free or US$19.95	Windows	Proprietary	?	Yes	(November 15, 2007)	

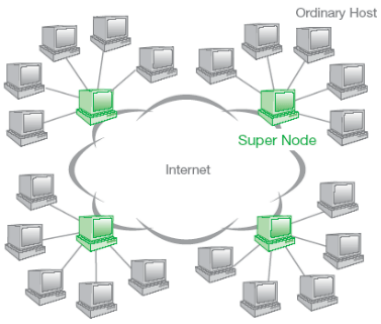
Algorithm taxonomy	Unstructured P2P Overlay Network Comparisons				
	Freenet	Gnutella	FastTrack/KaZaA	BitTorrent	Overnet/eDonkey 2000
Decentralization	Loosely DHT functionality.	Topology is flat with equal peers.	No explicit central server. Peers are connected to their Super-Peers.	Centralized model with a Tracker keeping track of peers.	Hybrid two-layer network composed of clients and servers.
Architecture	Keywords and descriptive text strings to identify data objects.	Flat and ad-hoc network of servents (peers). Flooding request and peers download directly.	Two-level hierarchical network of Super-Peers and peers.	Peers request information from a central Tracker.	Servers provide the locations of files to requesting clients for download directly.
Lookup protocol	Keys, Descriptive Text String search from peer to peer.	Query flooding.	Super-Peers.	Tracker.	Client-server peers.
System parameters	None.	None.	None.	*.torrent* file.	None.
Routing performance	Guarantee to locate data using *Key* search until the requests exceeded the Hops-To-Live (HTL) limits.	No guarantee to locate data; improvements made in adapting ultrapeer-client topologies; good performance for popular content.	Some degree of guarantee to locate data, since queries are routed to the Super-Peers, which has better scaling; good performance for popular content.	Guarantee to locate data and guarantee performance for popular content.	Guarantee to locate data and guarantee performance for popular content.
Routing state	Constant.	Constant.	Constant.	Constant but choking (temporary refusal to upload) may occur.	Constant.
Peers join/leave	Constant.	Constant.	Constant.	Constant.	Constant with bootstrapping from other peers and connect to server to register files being shared.
Security	Low; suffers from man-in-middle and Trojan attacks.	Low; threats: flooding, malicious content, virus spreading, attack on queries, and denial of service attacks.	Low; threats: flooding, malicious or fake content, viruses, etc. Spyware monitors the activities of peers in the background.	Moderate; centralized Tracker manages file transfer and allows more control, which makes it much harder to fake IP addresses, port numbers, etc.	Moderate; threats similar to those in FastTrack and Bit-Torrent.
Reliability/ fault resiliency	No hierarchy or central point of failure exists.	Degradation of the performance; peers receive multiple copies of replies from peers that have the data; requester peers can retry.	The ordinary peers are reassigned to other Super-Peers.	The Tracker keeps track of the peers and availability of the pieces of files; avoid choking by fibrillation by changing the peer that is choked once every ten seconds.	Reconnecting to another server; will not receive multiple replies from peers with available data.

■ Table 2. *A comparison of various unstructured P2P overlay network schemes.*

KaZaA

Το KaZaA είναι το πρώτο σύστημα P2P που μοιράζει τραγούδια με νόμιμη έγκριση, δημιουργήθηκε το 2001 από τον Niklas Zennstrom. Συνδυάζει το Napster και το Gnutella. Η χρήση του είναι δωρεάν και αν κάποιος χρήστης επιθυμεί μπορεί να κάνει δωρεά. Το KaZaA χρησιμοποιεί το πρωτόκολλο TCP επειδή χρειάζεται μία απ' άκρο εις άκρο αξιόπιστη ροή byte. Μοιάζει με το Gnutella στην δομή όσον αφορά ότι δεν χρησιμοποιεί έναν συγκεκριμένο server για τον εντοπισμό των αρχείων (αδόμητο), έχει όμως δύο ειδών κόμβους, τους κανονικούς και τους μεγάλους, σε αντίθεση με το Gnutella. Είναι υβριδικό δίκτυο, όπως δηλαδή και το FastTrack, που μοιάζουν πάρα πολύ. Σε αντίθεση με το Napster, το KaZaA δεν τερματίζεται απλά κλείνοντας τον κεντρικό server του συστήματος.

Έχει ιεραρχική οργάνωση των peers. Χρησιμοποιεί κατανεμημένο σχέδιο, προσπαθεί να έχει ισορροπία στην κίνηση των κόμβων και χρησιμοποιεί τοπικότητα σε γειτονικές επιλογές, έτσι ώστε να μειώνεται η καθυστέρηση αντίδρασης. Η τοπολογία του KaZaA είναι ισχυρά δυναμική -ανακατεύει τις συνδέσεις- , δηλαδή αν και οι ταυτόχρονες συνδέσεις ταλαντεύονται σ' ένα κατώτατο όριο, οι ανεξάρτητες αλλάζουν συχνά (Jian Liang, Rakesh Kuma,Keith W. Ross (2005)).
Για να μπορούν τα αρχεία να μοιράζονται πρέπει να μπορούν να παρακάμπτουν τα τοίχοι προστασίας και το NAT, τα οποία και είναι κυρίαρχα στο διαδίκτυο. Έτσι το KaZaA χρησιμοποιεί δυναμικές εισόδους σε συνδυασμό με την ιεραρχημένη της δομή για να αποφεύγει τα τείχη προστασίας και χρησιμοποιεί αντιστροφή σύνδεσης για να επιτρέπει τους peer με NAT να μοιράζονται αρχεία (Jian Liang, Rakesh Kuma, Keith W. Ross (2004), The KaZaA overlay: a measurement Study). Η ασφάλεια του KaZaA, παρόλα αυτά, είναι αρκετά ανεπαρκής καθώς απειλείται εύκολα από πλημμύρες, κακό ή ψεύτικο περιεχόμενο και ιούς. Το Spyware παρακολουθεί τις δραστηριότες των Peers στο παρασκήνιο (Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim (2005)).



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

BitTorrent

Το BitTorrent είναι ένα πρωτόκολλο σχεδιασμένο για μοίρασμα μεγάλων αρχείων σε κομμάτια χρησιμοποιώντας ένα αμοιβαίο μοίρασμα των κομματιών μεταξύ των peers (swarm). Η χρήση του είναι δωρεάν και είναι δομημένο δίκτυο. Δεν χρησιμοποιεί μηχανή αναζήτησης, την δουλεία αυτή την κάνουν κάποιες ιστοσελίδες όπως supernova.org ή btjunkie.org. Στην μεταφορά των δεδομένων χρησιμοποιεί την τεχνική pipeline προκειμένου να έχει καλύτερα αποτελέσματα (J.A.Pouwelse, P.Garbacki, D.H.J. Epema, H.J. Sips (2007)).

Το BitTorrent χρησιμοποιεί το πρωτόκολλο TCP επειδή είναι απαραίτητη η αξιοπιστία του και η αρχιτεκτονική του μπορεί να αναλυθεί σε πέντε βασικά συστατικά:

- ένα αρχείο που περιέχει όλες τις απαραίτητες πληροφορίες για την λειτουργικότητα του(metainfo file),
- έναν server που βοηθάει την διαχείριση του (tracker),
- χρήστες που ανταλλάζουν δεδομένα μέσω του BitTorrent (peers),
- τα δεδομένα που μοιράζονται (data) και
- το πρόγραμμα που χρησιμοποιεί ο χρήστης προκειμένου να χρησιμοποιήσει το BitTorrent(client).

Όσον αφορά την ασφάλεια, κινείται σε μέτρια επίπεδα

- κάποιος δεν θέλει να υπάρχει διαθέσιμο κάποιο αρχείο και έτσι δέχεται αυτός όλα τα συγκεκριμένα αρχεία τερματίζοντας έτσι το εύρος ζώνης ανεβάσματος του (Pollution),
- άρνησης διανομής της υπηρεσίας (DDOS) και βλάβες στην μεταφορά δεδομένων (bandwidth shaping) (Abhishek Sharma(2007)).
- Όμως, είναι κεντροποιημένο δίκτυο άρα υπάρχει περισσότερος έλεγχος και έτσι γίνεται δυσκολότερο να υπάρξουν ψεύτικες IP διευθύνσεις ή αριθμοί διόδου (Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim (2005)).

Skype

Το Skype είναι μια εφαρμογή P2P που δημιουργήθηκε το 2003 στο Λουξεμβούργο, όπου άτομα μπορούν να συνομιλούν μεταξύ τους γραπτά, με ήχο ή με βίντεο αλλά και να ανταλλάζουν αρχεία. Δημιουργήθηκε από την ίδια ομάδα που δημιούργησε και το KaZaA, με το οποίο έχει και παρόμοια αρχιτεκτονική.
Στο Skype ελέγχονται από έναν κεντρικό server οι συνδέσεις και οι λογαριασμοί και χρησιμοποιεί κόμβους και μεγάλους κόμβους, έχει δηλαδή ιεραρχημένη αρχιτεκτονική ως υβριδικό δίκτυο. Ο πηγαίος του κώδικας είναι κρυφός για τους χρήστες (David Tacconi (2008)).

Μετά την σύνδεση όλες οι επικοινωνίες γίνονται χωρίς να επικοινωνούν με τον κεντρικό server. Αυτό που γίνεται είναι κάποιοι κόμβοι αρκετά δυνατοί να προωθούνται σε μεγάλους κόμβους που δρουν σαν δρομολογητές μεταξύ των απλών χρηστών. Για να είναι λειτουργικό αυτό το σύστημα πρέπει οι μεγάλοι κόμβοι να γνωρίζουν την πλειοψηφία των υπόλοιπων. Οι μεγάλοι κόμβοι παρέχουν λίστα με τους συνδεδεμένους χρήστες στους υπόλοιπους χρήστες, αλλά παίζουν και σπουδαίο ρόλο στις μεθόδους διαπέρασης του τείχους προστασίας του Skype (Pascal Wibmann (2008)).

Η χρήση του είναι δωρεάν για την απλή χρήση του σε επικοινωνία υπολογιστή με υπολογιστή, όμως εάν ο χρήστης θέλει να μπορεί να καλεί σε κινητά νούμερα πρέπει να πληρώνει 6,89€τον μήνα για κλήσεις στην Ευρώπη ή 11,49€ ανά μήνα εάν θέλει να επικοινωνεί με ολόκληρο τον κόσμο. Για λόγους ασφαλείας οι εργασίες στο Skype -όπως επικοινωνία μεταξύ χρηστών- γίνονται με κρυπτογράφηση χρησιμοποιώντας ισχυρούς κρυπτογραφημένους αλγόριθμους. Σε μερικές περιπτώσεις, η επικοινωνία μπορεί να δρομολογείται μέσω άλλων χρηστών στο P2P δίκτυο (https://support.skype.com/en/faq/FA143/Is-Skype-secure).

Η αναζήτηση στο Skype είναι αρκετά εύκολη και χρησιμοποιεί την παγκόσμια τεχνολογία καταλόγου (Global Index Technology). Τα αποτελέσματα της αναζήτησης αποθηκεύονται σε κόμβους.
Για τις κλήσεις χρησιμοποιείται το πρωτόκολλο TCP, ενώ για την συνομιλία χρησιμοποιείται το πρωτόκολλο UDP επειδή δεν πειράζει ιδιαίτερα το αποτέλεσμα μία μικρή απώλεια δεδομένων, σε αντίθεση με το πρωτόκολλο TCP όπου και μία πιθανή καθυστέρηση θα ήταν καταστροφική.

Για την παράκαμψη του NAT χρησιμοποιεί STUN και TURN πρωτόκολλα (Luca De Cicco, Saverio Mascolo, Vittorio Palmisano (2008)).

FastTrack

Το FastTrack είναι άλλο ένα αδόμητο P2P που εμφανίστηκε σχεδόν μαζί με το Gnutella και χρησιμοποιούνταν από clients που μοιράζονταν αρχεία όπως Grokster και Imesh. Είναι ένα ιδιωτικό σύστημα που χρησιμοποιεί κωδικοποιημένο πρωτόκολλο. Μπορεί να χρησιμοποιηθεί δωρεάν από τους χρήστες. Το FastTrack που είναι υβριδικό δίκτυο, χρησιμοποιεί αρχιτεκτονική με superpeers στην οποία οι peers με μεγάλη χωρητικότητα είναι μεγάλοι κόμβοι και αυτά με μικρή χωρητικότητα είναι κανονικοί κόμβοι. Σε ένα σύνολο 3.000.000, οι μεγάλοι κόμβοι κυμαίνονται από 25.000 έως 40.000. Κάθε απλός κόμβος συνδέεται με έναν μεγάλο. Ο μεγάλος κόμβος παρέχει στους απλούς μία λίστα με τους υπόλοιπους μεγάλους που ο απλός μπορεί να συνδεθεί. Όταν ο απλός κόμβος κάνει μια ερώτηση στον μεγάλο και λαμβάνει τις απαντήσεις, αποσυνδέεται από τον εκάστοτε μεγάλο κόμβο και συνδέεται με κάποιον άλλον της λίστας. Με την επανασύνδεση λαμβάνει νέα λίστα με τους μεγάλους κόμβους που μπορεί να συνδεθεί.

Ένας απλός κόμβος διατηρεί περίπου 40 με 50 συνδέσεις με άλλους μεγάλους κόμβους. Ενώ, ένας μεγάλος κόμβος με σύνδεση μεγάλου εύρος ζώνης διατηρεί σύνδεση με περίπου 50-80 απλούς κόμβους.

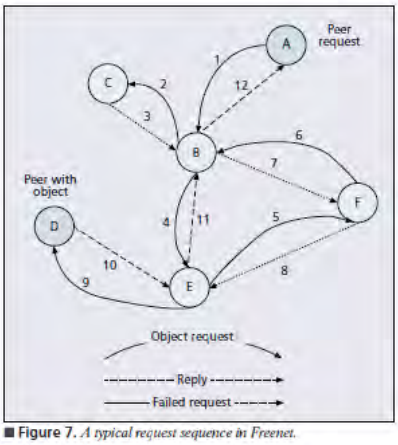
Οι συνδέσεις μεγάλου με απλό κόμβο και μεγάλο με μεγάλο φαίνεται να έχουν πολλαπλούς σκοπούς, συμπεριλαμβάνοντας τη διανομή φορτίων μεταξύ των μεγάλων κόμβων, βελτιώνοντας την τοπικότητα των συνδέσεων και ανακατεύοντας τις συνδέσεις από τους απλούς κόμβους για να αυξηθεί η κάλυψη στις μεγάλες αναζητήσεις πάνω στην επικάλυψη. Μεγάλη εντροπία σύνδεσης κάνει τον εντοπισμό των μεταφορών μέσω peers πιο δύσκολο, μία πιθανή παρακίνηση λαμβάνοντας υπ’ όψιν τις δικαστικές διαμάχες με άλλα συστήματα διαμοιρασμού αρχείων (John F. Buford, Heather Yu, Eng Keong Lua (2009), p55).
Αν και υβριδικό δίκτυο όπου μπορεί να υπάρχει έλεγχος μέσω των μεγάλων κόμβων, η ασφάλεια του FastTrack είναι αρκετά ανεπαρκής καθώς απειλείται εύκολα από πλημμύρες, κακό ή ψεύτικο περιεχόμενο και ιούς. Το Spyware παρακολουθεί τις δραστηριότητες των Peers στο παρασκήνιο (Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim (2005)).

Freenet

Το Freenet δημιουργήθηκε το 1999 από τον Ian Clarke και εκδόθηκε στο κοινό το 2000 ως ένας διανεμημένος και χαλαρά δομημένος P2P μηχανισμός διαμοιρασμού αρχείων με ασφάλεια και ανωνυμία. Παρέχει ραδιοφωνικές υπηρεσίες, υπηρεσίες φωνητικών συνομιλιών και διάφορα άρθρα όλο το εικοσιτετράωρο. Η χρήση του είναι δωρεάν και αν κάποιος χρήστης επιθυμεί μπορεί να κάνει δωρεά (www.freenet.com).
Τόσο τα αντικείμενα όσο και οι peers έχουν αναγνωριστικά ταυτότητας. Ταυτοποιητές δημιουργούνται με τη χρήση SHA-1 μονόδρομης συνάρτησης κατακερματισμού. Οι peer ταυτοποιητές αποκαλούνται κλειδιά δρομολόγησης. Κάθε peer έχει ορισμένο μέγεθος πίνακα δρομολόγησης που αποθηκεύει τους συνδέσμους προς άλλους peers. Κάθε είσοδος περιέχει το κλειδί δρομολόγησης των peers. Το ελεύθερο δίκτυο χρησιμοποιεί διαδρομές που βασίζονται σε κλειδιά για την εισαγωγή και ανάκτηση των αντικειμένων στο πλέγμα. Οι αιτήσεις διαβιβάζονται στα peers με το πλησιέστερο κλειδί χάραξης.

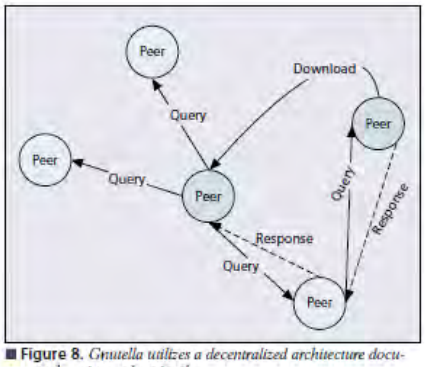
Αν ένα αίτημα κατά μήκος του άλματος αποτύχει, ο peer θα δοκιμάσει το επόμενο κοντινότερο κλειδί στον πίνακα δρομολόγησης. Ο αλγόριθμος δρομολόγησης είναι σαν απότομη άνοδος λόφου με δίαυλο οπισθοδρόμησης έως ότου το αίτημα TTL υπερβεί. Συνεπώς, εξαρτώμενο από την οργάνωση των συνδέσμων και τη διαθεσιμότητα των peers, είναι πιθανό ότι οι αιτήσεις μπορεί να αποτύχουν. Το ελεύθερο διαδίκτυο αντιδρά σ’ αυτό με απόκρυψη αντικειμένων κατά μήκος του δρόμου επιστροφής της αναζήτησης και εισαγωγή αιτήσεων. Ένα αντικείμενο αποθηκεύεται σε έναν peer μέχρις ότου δεν υπάρχει πια διαθέσιμος χώρος και είναι το πιο πρόσφατο χρησιμοποιημένο αντικείμενο σε αυτόν τον peer (John F. Buford, Heather Yu, Eng Keong Lua (2009), p56-60).

Ακολουθία αιτήματος στο Freenet



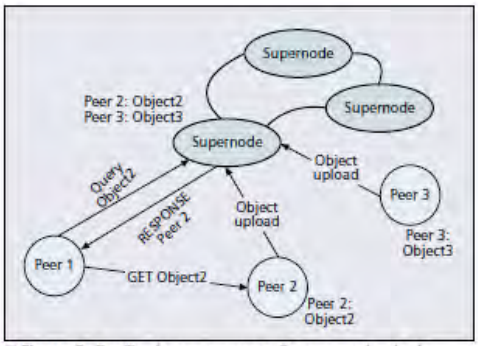
■ **Figure 7.** *A typical request sequence in Freenet.*

Το Gnutella χρησιμοποιεί μια αποκεντρωμένη αρχιτεκτονική για την ανάκτηση και την τοποθέτηση των εγγραφων.



■ **Figure 8.** *Gnutella utilizes a decentralized architecture document location and retrieval.*

Οι peers του FastTrack συνδέονται με Superpeers,η αναζήτηση δρομολογείται μεσω των Superpeers και οι λήψεις από τους peers.



■ **Figure 9.** *FastTrack peers connect to Superpeers whereby the search is routed through the Superpeers and downloads are done from the peer peers.*

Η αρχιτεκτονική του BitTorrent αποτελείται από ένα κεντρικό Tracker και ένα αρχείο .torrent

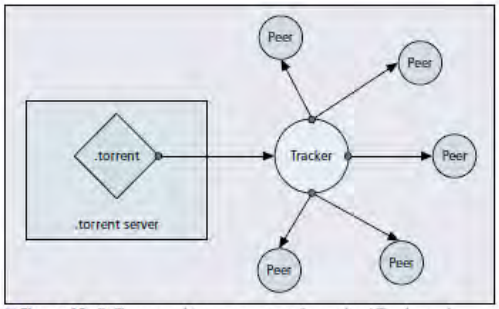


Figure 10. *BitTorrent architecture consists of centralized Tracker and .torrent file.*

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

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;

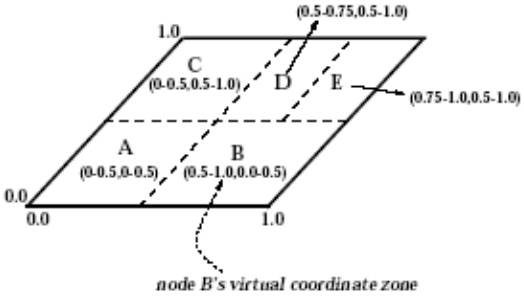
Οι Κατανεμημένοι Πίνακες Κατακερματισμού (**Distributed Hash Tables,DHT**) χρησιμοποιούνται κυρίως στα μη κεντρικοποιημένα δομημένα συστήματα (**decentralized structured systems**) οπως το **CAN** και το **Chord**. Αρχικά αναπτύχθηκαν σε ερευνητικό επίπεδο με σκοπό την εύκολη καταχώρηση και διαγραφή 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

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.

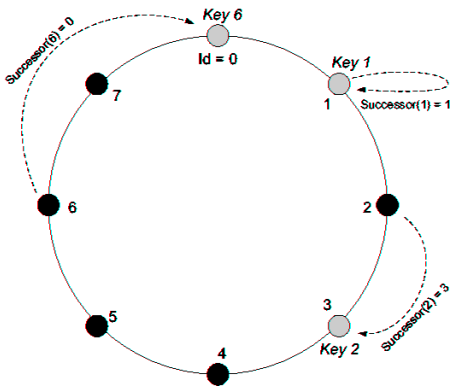
Στη συνέχεια, από τον κόμβο στον οποίο ανήκει το P, ανακτούμε το δεδομένο V. Αν το σημείο P δεν ανήκει στον κόμβο που έκανε την αίτηση για το (**K,V**) και δεν ανήκει ούτε στους κόμβους γειτονικών ζωνών, τότε η αίτηση δρομολογείται στην κοντινότερη γειτονική ζώνη του P.



Σχήμα 1 : Δομή του CAN

5.2.3 CHORD

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 2m***. Ένα κλειδί Κ 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 - 2m-1. Σε αυτήν την περίπτωση ο ***successor(K)*** είναι ο επόμενος του Κ 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).

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

6. Χαρακτηριστικά απόδοσης δικτύου

Τα πιο κοινώς χρησιμοποιούμενα, χαρακτηριστικά απόδοσης δικτύου, είναι τα ακόλουθα :

1)Χρόνος απόκρισης(Response time)

2)Καθυστέρηση (Latency)

3)Ρυθμός διεκπεραίωσης(Throughput)

4)Ρυθμός μετάδοσης(Bandwidth)

5)Ρυθμός απώλειας πακέτων(Loss)

6)Βέλτιστη δρομολόγηση πακέτων(Routing)

7) Αξιοπιστία μετάδοσης(Reliability)

8)Αξιοποίηση(Utilization)

Χρόνος απόκρισης(Response time) Σαν χρόνο απόκρισης ορίζουμε το χρόνο που χρειάζεται το σύστημα να ανταποκριθεί σε συγκεκριμένο αίτημα. Είναι επιθυμητό ο χρόνος απόκρισης να είναι πολύ μικρός αλλά όσο γίνεται μικρότερος τόσο αυξάνει το κόστος. Η αύξηση του κόστους οφείλεται σε δύο λόγους: στην ισχύ επεξεργασίας του συστήματος στον ανταγωνισμό των αιτημάτων – μειώνοντας το χρόνο για ένα αίτημα, αυξάνει ο χρόνος για κάποιο άλλο.

Καθυστέρηση (Latency) Η μετάδοση πληροφορίας σε ένα κανάλι, χαρακτηρίζεται από καθυστέρηση(delay). Αυτή η συνολική καθυστέρηση ονομάζεται και λανθάνων χρόνος(latency) και είναι το άθροισμα της καθυστέρησης διάδοσης(λόγω της πεπερασμένης ταχύτητας διάδοσης του σήματος μέσα στο κανάλι) και της καθυστέρησης μετάδοσης(λόγω της πεπερασμένης ταχύτητας μετάδοσης της πληροφορίας).

Ρυθμός διεκπεραίωσης(Throughput) Ορίζεται ως η συνολική ποσότητα της εργασίας που γίνεται σε δεδομένο χρόνο(συνήθως στη μονάδα του χρόνου sec). Ουσιαστικά σε ένα δίκτυο ,είναι ο μέσος ρυθμός επιτυχούς μετάδοσης μηνυμάτων/πακέτων πάνω από ένα κανάλι επικοινωνίας. Το throughput συνολικά του συστήματος, είναι το άθροισμα ,των ρυθμών δεδομένων, τα οποία παραδίδονται σε όλα τα τερματικά (κόμβους) ενός δικτύου. Μονάδες μέτρησης του throughput είναι συνήθως bits/ second, ή data packets/second ,ή data packets/ time slot.

Ρυθμός μετάδοσης(Bandwidth) Για κάθε κανάλι μετάδοσης στο δίκτυο, υπάρχει ένα όριο ,στο ρυθμό με τον οποίο μπορεί να μεταδόσει δεδομένα. Το όριο αυτό λέγεται μέγιστη ταχύτητα μετάδοσης ή εύρος ζώνης (bandwidth) του καναλιού.

Ρυθμός απώλειας πακέτων(Loss) Η απώλεια πακέτων είναι η απόρριψη των πακέτων σε ένα δίκτυο, όταν μια συσκευή δικτύου δρομολογητή ή άλλη συσκευή δικτύου, είναι υπερφορτωμένη, και δεν μπορεί να δεχθεί επιπλέον πακέτα σε μια δεδομένη στιγμή. Οι απώλειες είναι συνήθως λόγω της συμφόρησης στο δίκτυο και των υπερχειλίσεων buffer στα τερματικά συστήματα.

Βέλτιστη δρομολόγηση πακέτων (Routing) Ένα πρωτόκολλο δρομολόγησης ,καθορίζει τον τρόπο με τον οποίο οι δρομολογητές επικοινωνούν μεταξύ τους για τη διάδοση των πληροφοριών, δίνοντάς τους τη δυνατότητα να επιλέξουν διαδρομή για τη διάδοσή της,μεταξύ οποιωνδήποτε δύο κόμβων σε ένα δίκτυο υπολογιστών. Ο αλγόριθμος δρομολόγησης ,προσδιορίζει καποια συγκεκριμένη επιλογή της διαδρομής. Κάθε δρομολογητής έχει αρχικά γνώση μόνο των δικτύων με τα οποία συνδέεται άμεσα.Το πρωτόκολλο δρομολόγησης διαδίδει αυτή την πληροφορία ,πρώτα μεταξύ των άμεσων κομβων-γειτόνων, και στη συνέχεια σε όλο το δίκτυο. Με αυτό τον τρόπο, όλοι οι δρομολογητές αποκτούν γνώση της τοπολογίας του δικτύου. Στόχος είναι να επιλεχθεί η διαδρομή, που συνεπάγεται τι μικρότερη καθυστέρηση μετάδοσης.

Αξιοπιστία μετάδοσης(Reliability) Λόγω των περιορισμένων πόρων ενός δικτύου,(όπως η ενέργεια, υπολογιστική ικανότητα και χώρος αποθήκευσης των κόμβων) ,αλλά και εξαιτίας της ταχείας μεταβολής των χαρακτηριστικών των δυναμικών δικτύων(εισαγωγή/αποχώρηση κόμβων),υπάρχει η ανάγκη για μια αξιόπιστη μετάδοση δεδομένων στο δίκτυο. Σε γενικές γραμμές, υπάρχουν διάφορες προσεγγίσεις για εξασφάλιση της αξιοπιστίας όπως π.χ., αυτόματη αίτηση επανάληψης, πολλαπλές διαδρομές δρομολόγησης και κωδικοποίηση πηγής, που χρησιμοποιούνται, για την παροχή αξιόπιστης μεταφοράς δεδομένων.

Αξιοποίηση(Utilization) Η αξιοποίηση αναφέρεται στον προσδιορισμό του χρονικού ποσοστού που ένας πόρος του δικτύου βρίσκεται σε χρήση για μια συγκεκριμένη χρονική περίοδο. Η πιο σημαντική ίσως χρήση της αξιοποίησης των πόρων είναι η έρευνα για πιθανά σημεία μποτιλιαρίσματος και κυκλοφοριακής συμφόρησης. Έχει αποδειχθεί ότι ο χρόνος απόκρισης αυξάνεται εκθετικά με την αύξηση της αξιοποίησης ενός πόρου,με αποτέλεσμα πολύ γρήγορα να δημιουργηθεί συμφόρηση. Η διαχείριση ,παρακολουθώντας την αξιοποίηση πόρων ,μπορεί να βρει πόρους που υπολειτουργούν και άλλους που υπερλειτουργούν και να ρυθμίσει το δίκτυο ανάλογα.

Β' ΜΕΡΟΣ- ΠΡΟΣΟΜΟΙΩΣΗ ΣΥΣΤΗΜΑΤΟΣ NAPSTER

7. ΠΑΡΟΥΣΙΑΣΗ ΕΦΑΡΜΟΓΗΣ

Εισαγωγή

Έχοντας μελετήσει τους τρόπους μετάδοσης αρχείων, προχώρησα στην υλοποίηση του στόχου της διπλωματικής, δηλαδή τη δημιουργία ενός δικτύου Peer-to-Peer

Στην ανάλυση που ακολουθεί, θα περιγράψω τις προδιαγραφές της εφαρμογής, τον τρόπο λειτουργίας της, τις επιλογές των διαφόρων παραμέτρων και τους τρόπους αντιμετώπισης των προβλημάτων που παρουσιάστηκαν κατά την υλοποίηση.

Προδιαγραφές Εφαρμογής

Η εφαρμογή αναπτύχθηκε με τη γλώσσα προγραμματισμού Java, για το server και peer κομμάτι της. Χρησιμοποιήθηκε η πλατφόρμα NETBEANS IDE 7.2.1.

Γιατί Java

Οι λόγοι για τους οποίους επέλεξα τη Java για την ανάπτυξη της εφαρμογής είναι οι εξής: Είναι μια γλώσσα που προσφέρει αυξημένες δυνατότητες για την επικοινωνία των υπολογιστών, χωρίς να απαιτεί την χρήση εξειδικευμένων και πολύπλοκων εντολών. Η Java καταφέρνει μέσα από το package java.net να κρύψει όλη την low-level δυσκολία του προγραμματισμού των sockets και να δώσει ένα καθαρό αντικειμενοστραφές περιβάλλον προγραμματισμού.
Η εγγενής υποστήριξη πολυνηματικών εφαρμογών (multi-threading). Κάτι απολύτως απαραίτητο για την υλοποίηση της εφαρμογής, τόσο από την πλευρά του στησίματος του P2P δικτύου, όσο και στο κομμάτι της αναπαραγωγής.

Το τελικό πρόγραμμα μας μπορεί να τρέξει πρακτικά σε οποιοδήποτε γνωστό λειτουργικό σύστημα, καθώς η Java είναι ανεξάρτητη πλατφόρμας. Ένα πρόγραμμα Java δεν τρέχει ποτέ natively σε κάποιο μηχάνημα, αλλά ένα ειδικό native πρόγραμμα, ο Java Interpreter, διαβάζει το byte code του προγράμματος και το μεταφράζει στις αντίστοιχες εντολές της εκάστοτε μηχανής. Αυτό δοκιμάστηκε και στην πράξη μιας και η εφαρμογή μας γράφτηκε τόσο κάτω από περιβάλλον Windows 7, ενώ δοκιμάστηκε επιπλέον σε Ubuntu Linux 8.04 και Windows Vista χωρίς το παραμικρό πρόβλημα.

Ελάχιστες απαιτήσεις

Οι απαιτήσεις της εφαρμογής τόσο σε hardware, όσο και σε software είναι μικρές για τα σημερινά δεδομένα.
Software requirements:
Java Runtime Environment (JRE) 1.5 εγκατεστημένο

Hardware requirements:
Ένα σχετικά σύγχρονο μηχάνημα (5ετίας)
Port forwarding στις απαραίτητες θύρες (5555-5556 για τον server, 5557-5560 για τον peer)

ΕΦΑΡΜΟΓΗ NAPSTER

Η εφαρμογή χωρίζεται σε Client και Server. Οι ονομασίες που δόθηκαν είναι NapsterClient και NapsterServer αντίστοιχα.

Το Server κομμάτι της εφαρμογής τρέχει σταθερά σε μια πόρτα, και αναμένει για νέες συνδέσεις από clients. Για κάθε νέα σύνδεση που εμφανίζεται δημιουργείται ένα νέο thread το οποίο περιμένει για αιτήσεις από τον client και τις εξυπηρετεί ανάλογα.

Οι κλάσεις που έχουν φτιαχτεί για το server κομμάτι της εφαρμογής είναι:

- NapsterServer.java
- NapsterServerThread.java
- ClientInformation.java
- SharedFile.java
- Ip.java

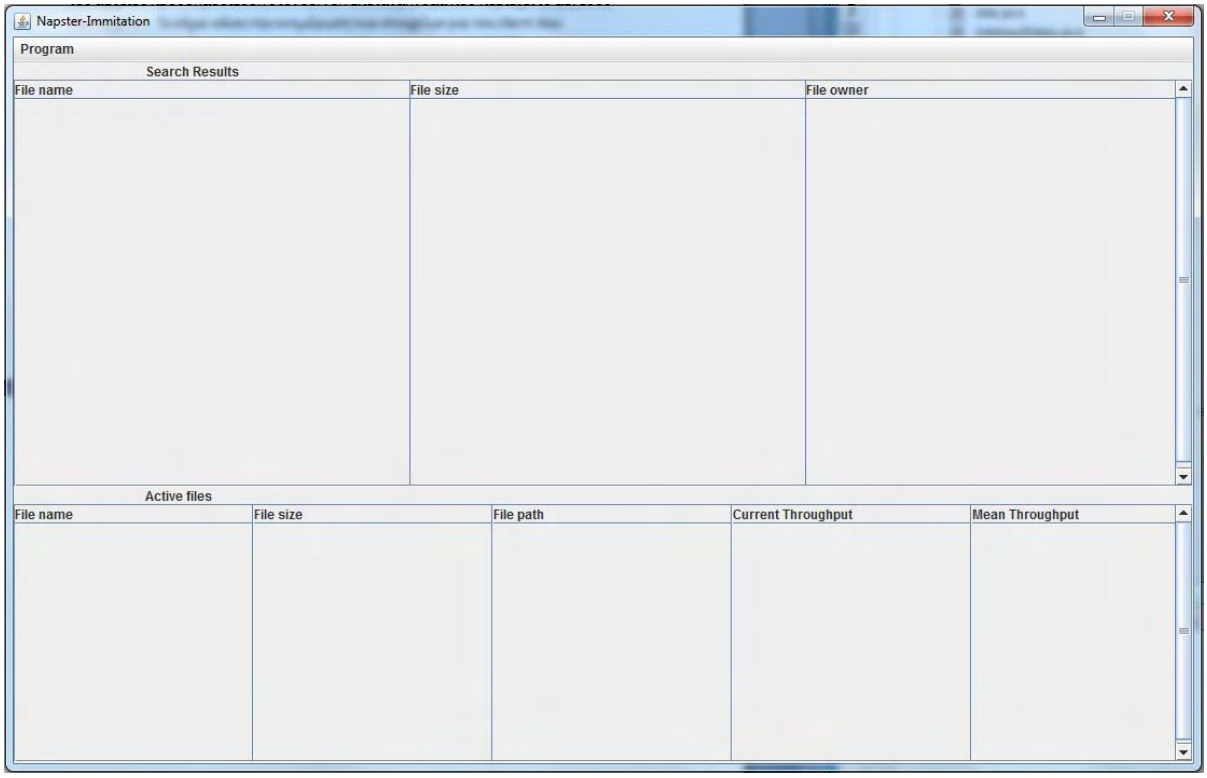
NapsterServer.java: Η κύρια κλάση της εφαρμογής. Ξεκινά την λειτουργία του server. Ανοίγει ένα ServerSocket σε μια προκαθορισμένη πόρτα. Κάνει listen στο Socket αυτό και αναμένει για συνδέσεις από χρήστες. Για κάθε νέα σύνδεση δημιουργεί μια νέα καταχώρηση σε μια λίστα με τους εξυπηρετούμενους χρήστες και ξεκινάει ένα νέο νήμα για την εξυπηρέτηση του νέου χρήστη.

NapsterServerThread.java: Το νήμα είναι επιφορτισμένο με την εξυπηρέτηση ενός client. Περιμένει για εντολές από τον εξυπηρετούμενο client. Τα μηνύματα που μπορεί να λάβει μπορεί να έχουνε ως header τα :

1. **ALIAS**: Μορφή αίτησης **"ALIAS username".** Όπου username ένα όνομα χρήστη, που έχει επιλέξει ο client. Έπονται μηνύματα που περιέχουν την IP address και το port, στο οποίο ακούει ο client. Το νήμα ελέγχει για την πιθανότητα ύπαρξης ενός χρήστη με το ίδιο username ήδη συνδεδεμένο, σε αυτήν την περίπτωση αλλάζει το όνομα του χρήστη σε κάτι παρεμφερές. Μορφή απάντησης **"OK USER username"**.
2. **PUBLISH**: Μορφή αίτησης **"PUBLISH filename".** Όπου το filename είναι το όνομα του αρχείου προς δημοσίευση στον server. Έπεται μήνυμα που περιέχει το μέγεθος του αρχείου. Το νήμα κάνει την ενημέρωση των στοιχείων για τον client που εξυπηρετεί και στέλνει πίσω την απάντηση. Μορφή απάντησης **"OK".**
3. **SEARCH**: Μορφή αίτησης **"SEARCH filename".** Όπου filename μια έκφραση προς αναζήτηση. Το νήμα ψάχνει για κοντινά ονόματα με το filename στη λίστα με τους εξυπηρετούμενους clients. Κάθε αποτέλεσμα αποθηκεύεται σε μια προσωρινή λίστα. Μορφή απάντησης **"result size",** όπου result size το σύνολο των αποτελεσμάτων που ταιριάζουνε στην έκφραση προς αναζήτηση. Για κάθε αρχείο που βρέθηκε έπονται τα μηνύματα : "file name", "owner", "file size", "owner's IP", "owner's port", με το ακριβές όνομα του αρχείου, το alias του ιδιοκτήτη, το μέγεθος του αρχείου, την διεύθυνση IP και το port του ιδιοκτήτη αντίστοιχα. Έτσι σε περίπτωση που ο client θέλει να κατεβάσει κάποιο αρχείο συνδέεται άμεσα με τον ιδιοκτήτη του αρχείου αυτού.

Οι κλάσεις ClientInformation.java, SharedFile.java, Ip.java είναι καθαρά βοηθητικές και δεν αναλύεται η λειτουργία τους.

Το client κομμάτι της εφαρμογής ξεκινάει με ένα γραφικό περιβάλλον, όπως φαίνεται παρακάτω:



Στο μενού ο χρήστης έχει τις επιλογές να ξεκινήσει μια σύνδεση με τον server, να δημοσιεύσει ένα αρχείο, να αναζητήσει ένα αρχείο και να τερματίσει την εφαρμογή. Καμία από τις υπόλοιπες λειτουργίες δεν είναι διαθέσιμες χωρίς να έχει γίνει πρώτα μια σύνδεση με τον server. Το παράθυρο χωρίζεται σε 2 τμήματα search results, όπου εμφανίζονται τα αποτελέσματα από μια αναζήτηση και active files, όπου εμφανίζονται τα αρχεία που έχουνε γίνει publish ή έχουνε γίνει download.

Οι κλάσεις που έχουνε φτιαχτεί για τον client είναι:

- NapsterClient.java
- TransferAcceptThread.java
- TransferThread.java
- NapConnectDialog.java
- SharedFile.java
- Ip.java

NapsterClient.java: Η κύρια κλάση της εφαρμογής υλοποιεί το γραφικό περιβάλλον της. Στον constructor γίνονται όλες οι αρχικοποιήσεις. Κατασκευάζεται το menu και το κυρίως μέρος του παραθύρου. Επίσης αρχικοποιείται ένα νήμα το οποίο δέχεται συνδέσεις από άλλους χρήστες σε περίπτωση που ζητηθεί ένα αρχείο από τον χρήστη. Οι βασικές λειτουργίες της εφαρμογής βρίσκονται στο menu(Connect to server, Publish file, Search file και Exit) και στις λίστες(Download πατώντας δεξί κλικ σε ένα αποτέλεσμα της searchList).

Στο menu διακρίνουμε τις λειτουργίες:

- **Connect to server**: Επιλέγοντας το εμφανίζεται ένα παράθυρο το οποίο απαιτεί από τον χρήστη να εισάγει τις απαραίτητες πληροφορίες ώστε να επιτευχθεί σύνδεση με server (IP, Port, Username). Αφού εισαχθούν τα απαιτούμενα δεδομένα πατώντας το πλήκτρο connect στέλνεται μήνυμα **"ALIAS username"** στον server. Και κατόπιν στέλνονται τα δεδομένα IP και Port του client. Τέλος αναμένεται απάντηση από τον server μορφής **"OK USER username"**.

- **Publish file**: Επιλέγοντάς το εμφανίζεται παράθυρο με την δυνατότητα επιλογής ενός αρχείου από το τοπικό σύστημα αρχείων του χρήστη. Για το αρχείο που επιλέγεται δημιουργείται ένα αντικείμενο της κλάσης SharedFile.java, όπου βρίσκονται οι απαραίτητες πληροφορίες για το αρχείο αυτό. Κατόπιν στέλνεται μήνυμα **"PUBLISH filename"** στον server. Ακολουθεί αποστολή του μεγέθους του αρχείου. Τέλος αναμένεται απάντηση μορφής **"OK"** και ακολουθεί εισαγωγή εγγραφής για το αρχείο στις λίστες active files στο κάτω μέρος του παραθύρου.

- **Search file**: Επιλέγοντας το εμφανίζεται παράθυρο που ζητά την εισαγωγή ενός String με το όνομα του αρχείου ή τμήματος αυτού. Διαγράφονται όλες οι εγγραφές από τις λίστες search results στο επάνω μέρος του παραθύρου. Στέλνεται μήνυμα στον server τύπου **"SEARCH filename"**. Ο client αναμένει απάντηση τύπου **"result size"**, ώστε να γνωρίζει πόσο είναι το πλήθος των αποτελεσμάτων που περιμένει. Κατόπιν ακολουθούν μηνύματα με τα απαραίτητα στοιχεία για κάθε αρχείο. Για κάθε αρχείο το οποίο λαμβάνει ο client ως αποτέλεσμα γίνεται εισαγωγή του στις λίστες Search results.

Στην λίστα Search Results, επιλέγοντας μια εγγραφή στην λίστα αυτή και πατώντας δεξί κλικ εμφανίζεται επιλογή το επιλεγμένο αρχείο να γίνει download.

- **Download**: Επιλέγοντας το εκκινείτε ένα νήμα της κλάσης TransferThread.java, το οποίο είναι υπεύθυνο για την λήψη του αρχείου από τον ιδιοκτήτη του. Ο client είναι γνώστης του Ip και του port, στο οποίο ο συγκεκριμένος client δέχεται αιτήσεις για λήψη αρχείων.

TransferAcceptThread.java: Ένα νήμα το οποίο εκκινείτε από την κύρια κλάση. Ξεκινάει ένα ServerSocket, το οποίο κάνει listen για νέες συνδέσεις προς τον client. Οι νέες αυτές συνδέσεις γίνονται μόνο όταν κάποιος άλλος client αιτείται την λήψη ενός αρχείου από αυτόν. Για κάθε νέα σύνδεση που γίνεται δημιουργείται ένα νήμα της κλάσης TransferThread.java, το οποίο αναλαμβάνει την αποστολή του αρχείου στον αιτούμενο.

TransferThread.java: Είναι το νήμα που αναλαμβάνει την αποστολή ή την παραλαβή ενός αρχείου. Έχει δύο constructors ανάλογα με τον ρόλο τον οποίο θα επιτελέσει(αποστολέας ή παραλήπτης).

- **Παραλήπτης**: Το νήμα στέλνει μήνυμα μορφής **"FILE filename"** και αναμένει απάντηση τύπου **"OK"**. Κατόπιν ανοίγει ένα αρχείο με το κατάλληλο όνομα στο τοπικό σύστημα αρχείων για την αποθήκευση του αρχείου που θα λάβει. Όσο διαβάζονται δεδομένα από το stream socket το νήμα τα αποθηκεύει στο τοπικό αρχείο. Μόλις λάβει όλα τα δεδομένα τότε κλείνει το αρχείο και την σύνδεση. Παράλληλα κρατά ενημερωμένες τις λίστες Active files με το λαμβανόμενο αρχείο.

- **Αποστολέας**: Το νήμα περιμένει αίτηση μορφής **"FILE filename"**. Κάνει αναζήτηση στο τοπικό σύστημα αρχείων για το αρχείο για το οποίο έγινε η αίτηση και στέλνει απάντηση **"OK"** αν το αρχείο βρεθεί. Κατόπιν ανοίγει το αρχείο και ξεκινάει να το στέλνει μέσω του stream socket. Αφού ολοκληρώσει την μεταφορά κλείνει την σύνδεση.

- **<u>Υπολογισμός Throughput</u>**: Κατά την διαδικασία λήψη ενός αρχείου γίνεται ένας υπολογισμός για το τρέχων throughput και για το μέσω throughput της μεταφοράς. Πιο συγκεκριμένα για κάθε 1000 αναγνώσεις από το socket γίνεται ένας υπολογισμός currentThroughput = (bytes read in elapsed time)/(elapsed time) και meanThroughput = (meanThroughput + currentThroughput)/2. Οι τιμές αυτές που υπολογίζονται ανανεώνονται στην λίστα των active files.

Οι κλάσεις SharedFile.java, NapConnectDialog.java, SharedFile.java και Ip.java έχουνε καθαρά βοηθητικό χαρακτήρα και δεν παρουσιάζουν κάποια ειδικότερη λειτουργικότητα.

8.ΚΩΔΙΚΑΣ JAVA

8.1 Για τον Server

Κλάση ClientInformations.java

```
package napsterserver;

import java.util.LinkedList;

public class ClientInformations {

    private String username;
    private LinkedList<SharedFile> clientShares;

    private String clientIp;
    private int clientAcceptPort;

    ClientInformations() {

        username = "";
        clientShares = new LinkedList();

    }

    ClientInformations(String username) {

        this.username = username;
        clientShares = new LinkedList();

    }



    ClientInformations(String username, String clientIp, int
clientAcceptPort) {

        this.username = username;
        clientShares = new LinkedList();

        this.clientIp = clientIp;
        this.clientAcceptPort = clientAcceptPort;

    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
```

```
    public String getClientIp() {
        return clientIp;
    }

    public void setClientIp(String clientIp) {
        this.clientIp = clientIp;
    }

    public int getClientAcceptPort() {
        return clientAcceptPort;
    }

    public void setClientAcceptPort(int clientAcceptPort) {
        this.clientAcceptPort = clientAcceptPort;
    }

    public LinkedList<SharedFile> getClientShares() {
        return clientShares;
    }

    public LinkedList<SharedFile> searchFile(String fileName) {

        LinkedList<SharedFile> temp = new LinkedList<>();

        for(int i=0; i<clientShares.size(); i++) {
            if(clientShares.get(i).getFileName().matches(fileName))
                temp.add(clientShares.get(i));
        }

        return temp;

    }

    public void addPublishedFile(SharedFile aFile) {
        clientShares.addLast(aFile);
    }

}
```

Κλάση Ip.java

```
package napsterserver;

import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.util.Enumeration;

class Ip {

      // boh8htikh klash gia na pairnoume to topiko Ip tou mixanimatos
    static InetAddress getLocalIp() {

        NetworkInterface iface = null;

        try {

            for(Enumeration ifaces =
NetworkInterface.getNetworkInterfaces(); ifaces.hasMoreElements();) {

                iface = (NetworkInterface)ifaces.nextElement();
                if(iface.isUp() && !iface.isLoopback()) {

                    InetAddress ia = null;
                    for(Enumeration ips = iface.getInetAddresses();
ips.hasMoreElements();) {

                        ia = (InetAddress)ips.nextElement();

                        if(ia.isSiteLocalAddress())
                            return ia;

                    }
                }
            }
        }catch(SocketException se) {
                se.printStackTrace();
        }

        return null;

    }

}
```

Κλάση NapsterServer.java

```
package napsterserver;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.LinkedList;


public class NapsterServer {

    public static LinkedList<ClientInformations> clients = new
LinkedList<>();

    public static void main(String[] args) {

        ServerSocket listeningSocket;

        // ksekinaei kai perimenei sundeseis (xristes)
        try {

            //Open socket
            listeningSocket = new ServerSocket(Integer.parseInt(args[0]),
0, Ip.getLocalIp());    // h porta dinetai apo to prompt
               System.out.println("Server running at " + Ip.getLocalIp()
+ " at port " + Integer.parseInt(args[0]));
            while(true) {
                //Wait for new connections
                Socket connectionSocket = listeningSocket.accept();

                System.out.println("new connection");
                //Create new entry for the client that just connected
                ClientInformations temp = new ClientInformations();
                clients.addLast(temp);

                //Start a service thread to service client
                NapsterServerThread tempThread = new
NapsterServerThread(connectionSocket, temp, clients);
                tempThread.start();

            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

Κλάση NapsterServerThread.java

```
package napsterserver;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.*;
import java.util.LinkedList;

//To nima pou einai epifortismeno me thn eksypiretisi enos xrhsth
public class NapsterServerThread extends Thread{

    //Ta streams gia thn epikoinwnia me ton client
    private BufferedReader inFromClient;
    private PrintWriter outToClient;
    private Socket clientSocket = null;

    //Flag gia na elegxoume an einai energh h syndesh
    private boolean connActive = false;
    //To alias to serviced client
    private String alias;

    //lista me olous tous clients pou eksyphretei o server
    private LinkedList<ClientInformations> clients;
    //Plhrofories gia ton client pou pou eksyphretei to nima
    private ClientInformations servicedClient;

    //Constructor
    NapsterServerThread(Socket clientSocket, ClientInformations
servicedClient, LinkedList<ClientInformations> clients) {

        try {
            inFromClient = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            outToClient = new PrintWriter(clientSocket.getOutputStream(),
true);
        } catch (IOException e) {
            e.printStackTrace();
        }

        this.servicedClient = servicedClient;
        this.clients = clients;
        this.clientSocket = clientSocket;

        connActive = true;

    }


    public void run() {
        //The thread has to wait for client's commands

        while(connActive) {
            try {
```

```
                String clientOption, reply = null;
                //Diavazoume ena aithma apo ton client
            clientOption = inFromClient.readLine();
            System.out.println(clientOption);

                //Aithma syndeshs
                if(clientOption.startsWith("ALIAS")) {


                    alias = clientOption.substring(6);
                    String user = new String(alias);
                    int temp = 1;

                    //Elegxos an yparxei hdh to onoma
                    for(int i = 0; i < clients.size(); i++) {
                        if(clients.get(i).getUsername().equals(alias)) {
                            user = alias + "" + temp;
                            temp++;
                            i = 0;
                        }
                    }

                    //Enhmerwsh twn stoixeiwn toy client
                    servicedClient.setUsername(user);

                    String clientIp =
inFromClient.readLine().substring(3);
                    servicedClient.setClientIp(clientIp);
                    System.out.println(clientIp);

                    int clientAcceptPort =
Integer.parseInt(inFromClient.readLine().substring(5));
                    servicedClient.setClientAcceptPort(clientAcceptPort);
                    System.out.println(clientAcceptPort);

                    //Apanthsh
                    reply = "OK USER " + user;
                    alias = user;

                    outToClient.println(reply);

                //Aithma dhmosieyshs
                } else if(clientOption.startsWith("PUBLISH")) {

                    String temp = clientOption.substring(8);
                    //Enhmerwsh twn stoixeiwn tou client gia to neo
arxeio pou dhmosieyse
                    servicedClient.addPublishedFile(new SharedFile(temp,
alias, Long.parseLong(inFromClient.readLine()),
servicedClient.getClientIp(), servicedClient.getClientAcceptPort()));

                    reply = "OK";

                    outToClient.println(reply);
                //Aithma anazhthshs
                } else if(clientOption.startsWith("SEARCH")) {

                    String temp = clientOption.substring(7);
                    //Anazhthsh gia arxeia pou teriazoune me thn ekfrash
                    LinkedList<SharedFile> results = new LinkedList<>();
```

```
                    for(int i = 0; i < clients.size(); i++) {

                        if(clients.get(i).getUsername().equals(alias))
                            continue;

                        results.addAll(clients.get(i).searchFile("(.*)" +
temp + "(.*)"));
                    }

                    outToClient.println(results.size());
                    System.out.println("Result size : " +
results.size());
                    //Apostolh olwn twn plhroforiwn gia ta apotelesmata
pou vre8ikan
                    for(int i = 0; i < results.size(); i++) {


outToClient.println(results.get(i).getFileName());
                        System.out.println("FileName    : " +
results.get(i).getFileName());
                        outToClient.println(results.get(i).getOwner());
                        System.out.println("Owner    : " +
results.get(i).getOwner());

outToClient.println(results.get(i).getFileSize());
                        System.out.println("FileSize    : " +
results.get(i).getFileSize());

outToClient.println(results.get(i).getOwnersIP());
                        System.out.println("Owner at IP : " +
results.get(i).getOwnersIP());

outToClient.println(results.get(i).getOwnersPort());
                        System.out.println("Owner at Port : " +
results.get(i).getOwnersPort());


                    }

                }

                System.out.println(reply + "\n");

            } catch (IOException e) {
                connActive = false;
                clients.remove(servicedClient);
                e.printStackTrace();
            }
        }
    }

}
```

Κλάση SharedFile.java

```
package napsterserver;

public class SharedFile {

    private String fileName;
    private String owner;
    private long fileSize;

    //Transef information in case someone wants to download the file
    private String ownersIP;
    private int ownersPort;

    SharedFile() {

        fileName = "";
        fileSize = 0;

    }

    SharedFile(String fileName, String owner, long fileSize) {

        this.fileName = fileName;
        this.owner = owner;
        this.fileSize = fileSize;

    }

    SharedFile(String fileName, String owner, long fileSize, String
ownersIP, int ownersPort) {

        this.fileName = fileName;
        this.owner = owner;
        this.fileSize = fileSize;

        this.ownersIP = ownersIP;
        this.ownersPort = ownersPort;

    }

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public long getFileSize() {
        return fileSize;
    }

    public void setFileSize(long fileSize) {
        this.fileSize = fileSize;
    }
```

```
    public String getOwner() {
        return owner;
    }

    public void setOwner(String owner) {
        this.owner = owner;
    }

    public String getOwnersIP() {
        return ownersIP;
    }

    public void setOwnersIP(String ownersIP) {
        this.ownersIP = ownersIP;
    }

    public int getOwnersPort() {
        return ownersPort;
    }

    public void setOwnersPort(int ownersPort) {
        this.ownersPort = ownersPort;
    }

}
```

8.2 Για τον Client

Κλάση Ip.java

```
package napsterclient;

import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.util.Enumeration;

class Ip {

      // boh8htikh klash gia na pairnoume to topiko Ip tou mixanimatos
    static InetAddress getLocalIp() {

        NetworkInterface iface = null;

        try {

            for(Enumeration ifaces =
NetworkInterface.getNetworkInterfaces(); ifaces.hasMoreElements();) {

                iface = (NetworkInterface)ifaces.nextElement();
                if(iface.isUp() && !iface.isLoopback()) {

                    InetAddress ia = null;
                    for(Enumeration ips = iface.getInetAddresses();
ips.hasMoreElements();) {

                        ia = (InetAddress)ips.nextElement();

                        if(ia.isSiteLocalAddress())
                            return ia;

                    }
                }
            }
        }catch(SocketException se) {
                se.printStackTrace();
        }

        return null;

    }

}
```

Κλάση NapConnectDialog.java

```
package napsterclient;

import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class NapConnectDialog extends JDialog{

    private String inputLabels[] = {"Server's IP", "Server's Port",
"Alias"};

    private JTextField input[] = new JTextField[3];

    public NapConnectDialog(JFrame owner) {
        //kalei ton constructor ths JDialog
        super(owner, "Connection info");

        this.add(getGridBagPanel());

        this.pack();

        this.setResizable(false);

        this.setVisible(true);
    }

    private void addComponent(JComponent cont, JComponent item,
GridBagLayout gridbag, GridBagConstraints c) {
        gridbag.setConstraints(item, c);
        cont.add(item);
    }

    protected boolean isAnInputEmpty(JTextField[] input) {

        for(int i=0; i<input.length; i++){
            if(input[i].getText().length() == 0)
                return true;
        }

        return false;
    }
    //Me8odos pou pairnei ws parametro en amhnyma kai to emfanizei sto
message dialog ws error
    protected void showErrorDialog(String msg) {
```

```
        JOptionPane.showMessageDialog(NapConnectDialog.this, msg,
"Error", JOptionPane.ERROR_MESSAGE, null);
    }

    private JPanel getGridBagPanel() {

        JPanel inputs = new JPanel();

        //ka8orizei thn emfanish tou dialog
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();

        inputs.setLayout(gridbag);
        inputs.setFont(new Font("Helvetica", Font.PLAIN, 14));

        //ti apostaseis 8a uparxoun anamesa sta antikeimena
        c.insets = new Insets(10, 20, 15, 30);
        for(int i=0; i<input.length; i++) {//gia thn aristerh sthlh
            //If the component is too big resize it horizontally
            c.gridwidth = GridBagConstraints.RELATIVE;
            c.fill = GridBagConstraints.HORIZONTAL;
            //take the extra width
            c.weightx = 3.0;
            //topo8etei sto JPanel inputs ena neo JLabel me titlo
inputLabel[i]
            addComponent(inputs, new JLabel(inputLabels[i]), gridbag, c);

            //End of Row
            c.gridwidth = GridBagConstraints.REMAINDER;
            c.weightx = 0.0;
            //pro8etoume ta JTextField me ta default values
            addComponent(inputs, (input[i]=new JTextField(15)), gridbag,
c);

        }

        //pros8etoume ta 2 koumpia connect kai cancel
        c.insets = new Insets(15, 40, 10, 50);
        c.gridwidth = GridBagConstraints.RELATIVE;
        c.weightx = 3.0;
        //Dhmiourgoume to koumpi accept
        JButton connect = new JButton("Connect");
        connect.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                //elegxoume an o pinakas exei adeio stoixeio aki
termatizei
                if(isAnInputEmpty(input)) {
                    showErrorDialog("Error: One of the fields is empty");
                    return;
                }

                ((NapsterClient)
getParent()).connectToServer(input[0].getText(), input[1].getText(),
input[2].getText());
                NapConnectDialog.this.dispose();

            }
        });
```

```
        //pro8etoume to koumpi accept sto Dialog
        addComponent(inputs, connect, gridbag, c);

        //dhmiourgia koumpiou cancel
        c.weightx = 2.0;
        JButton cancel = new JButton("Cancel");
        cancel.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                NapConnectDialog.this.dispose();
            }
        });

        //kanei add to koumpi cancel
        addComponent(inputs, cancel, gridbag, c);

        //epistrefei to Jpanel oloklhrwmeno
        return inputs;

    }

}
```

Κλάση NapsterClient.java

```
package napsterclient;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.HeadlessException;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.LinkedList;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.DefaultListModel;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.ScrollPaneConstants;


public class NapsterClient extends JFrame {

    private Socket clientSocket = null;
    private PrintWriter outToServer = null;
    private BufferedReader inFromServer = null;
    private boolean connActive = false;
    private String alias;

    private LinkedList<SharedFile> clientShares;
    private LinkedList<SharedFile> results;

    private JList[] searchLists;
    private DefaultListModel[] searchListModel;

    private JList[] activeLists;
```

```
    private DefaultListModel[] activeListModel;

    private TransferAcceptThread acceptConnectionsThread;

    private String acceptIp;
    private int acceptPort;

    private boolean search = false;

    public NapsterClient() {
        //kalei ton kataskeuastei tou Jframe
        super("Napster-Immitation");
        //8esh pou 8a emfanistei to para8uro
        int inset = 200;

      this.setBounds(inset, inset, 1100, 700);
        //mporei na to allaksoume giaauto vazoume false
        this.setResizable(false);
        //orizei th mpara tou menu
        this.setJMenuBar(createMenuBar());

        this.add(createView());

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.setVisible(true);

        clientShares = new LinkedList<>();
        //Ekinisi nimatos gia thn apodoxh aithsewn gia apostoli arxeiou
        Random portGen = new Random();
        acceptPort = portGen.nextInt(65535 - 1024) + 1024;

        try {
            acceptIp = InetAddress.getLocalHost().getHostAddress();
            acceptConnectionsThread = new TransferAcceptThread(acceptIp,
acceptPort, clientShares, activeListModel, alias);
        } catch (UnknownHostException ex) {

Logger.getLogger(NapsterClient.class.getName()).log(Level.SEVERE, null,
ex);
        }
        System.out.println("Ip : "+acceptIp + " Port : "+acceptPort);
        acceptConnectionsThread.start();

    }

    private void showErrorDialog(String msg) {
        JOptionPane.showMessageDialog(NapsterClient.this, msg, "Error",
JOptionPane.ERROR_MESSAGE, null);
    }

    private JMenuBar createMenuBar() {

        JMenuBar menuBar = new JMenuBar();

        JMenu menu = new JMenu("Program");

        JMenuItem item = new JMenuItem("Connect to server");
        item.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent e) {
                new NapConnectDialog(NapsterClient.this);
            }

        });
        //pros8etw sto menu to loaded image
        menu.add(item);

        item = new JMenuItem("Publish File");
        item.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                if(!connActive) {
                    showErrorDialog("No connection to server active");
                    return;
                }

                JFileChooser chooser = new JFileChooser();

                try {
                    chooser.showOpenDialog(NapsterClient.this);
                } catch(HeadlessException hle) { return; }

                //pernaei sto file pou dialexthke
                File publishFile = chooser.getSelectedFile();
                SharedFile selectedFile = new
SharedFile(publishFile.getName(), publishFile.getPath(),
publishFile.length());
                clientShares.addLast(selectedFile);

activeListModel[0].addElement(selectedFile.getFileName());

activeListModel[1].addElement(selectedFile.getFileSize());
                activeListModel[2].addElement(selectedFile.getPath());
                activeListModel[3].addElement("0");
                activeListModel[4].addElement("0");

                outToServer.println("PUBLISH " +
selectedFile.getFileName());
                outToServer.println(selectedFile.getFileSize());

                try {

                    String temp = inFromServer.readLine();

                    if(!temp.startsWith("OK")) {
                        clientShares.remove(selectedFile);
                        showErrorDialog("Error while publishing file");
                    }

                } catch(IOException ioe) {
                    ioe.printStackTrace();
                }

            }
        });

        menu.add(item);
```

```
        item = new JMenuItem("Search file");
        item.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                if(!connActive) {
                    showErrorDialog("No connection to server active");
                    return;
                }

                searchListModel[0].clear();
                searchListModel[1].clear();
                searchListModel[2].clear();

                String inputValue = JOptionPane.showInputDialog("Please
input a value");
                outToServer.println("SEARCH " + inputValue);

                try {
                    int resultsNumber =
Integer.parseInt(inFromServer.readLine());

                    results = new LinkedList<>();

                    for(int i=0; i<resultsNumber; i++) {

                        SharedFile cur = new
SharedFile(inFromServer.readLine(), inFromServer.readLine(),
Long.parseLong(inFromServer.readLine()), inFromServer.readLine(),
Integer.parseInt(inFromServer.readLine()));
                        results.add(cur);


searchListModel[0].addElement(results.getLast().getFileName());

searchListModel[1].addElement(results.getLast().getFileSize());

searchListModel[2].addElement(results.getLast().getPath());

                    }

                } catch(IOException ioe) {
                    ioe.printStackTrace();
                }
            }

        });

        menu.add(item);

        item = new JMenuItem("Exit");
        item.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }

        });
        menu.add(item);
```

```
        menuBar.add(menu);

        return menuBar;
    }

    private JList createJList(int width, int height, DefaultListModel
model) {

        final JList aList = new JList(model);
        aList.setSize(width, height);
        aList.setVisibleRowCount(15);

        final JPopupMenu popup = new JPopupMenu();
        JMenuItem item = new JMenuItem("Download");
        item.addActionListener(new ActionListener(){

            @Override
            public void actionPerformed(ActionEvent e) {

                if(search) {
                    SharedFile downloadFile =
results.get(searchLists[0].getSelectedIndex());

                    TransferThread temp = new
TransferThread(downloadFile.getOwnersIP(), downloadFile.getOwnersPort(),
downloadFile.getFileName(), clientShares, activeListModel, alias,
activeListModel[0].size());
                    temp.start();
                }

            }

        });

        popup.add(item);


        MouseListener mouseListener = new MouseAdapter() {

            public void mouseClicked(MouseEvent e) {

                    if(e.getButton() == MouseEvent.BUTTON1 ||
e.getButton() == MouseEvent.BUTTON2) {
                        int index = aList.locationToIndex(e.getPoint());
                        for(int i=0; i<3; i++) {
                            if(aList == searchLists[i])
                                search = true;
                        }
                        if(search) {
                            for(int i=0; i<3; i++) {
                                searchLists[i].setSelectedIndex(index);
                            }
                        } else {
                            for(int i=0; i<3; i++) {
                                activeLists[i].setSelectedIndex(index);
                            }
                        }
                    }

                }
```

```
            public void mousePressed(MouseEvent e) {
                maybeShowPopup(e);
            }

            public void mouseReleased(MouseEvent e) {
                maybeShowPopup(e);
            }

            private void maybeShowPopup(MouseEvent e) {
                if (e.isPopupTrigger()) {
                    popup.show(e.getComponent(), e.getX(), e.getY());
                }
            }

        };

        aList.addMouseListener(mouseListener);

        return aList;
    }

    private Box createBoxPanel(int width, int height, String labelString,
JList listVal) {

        Box aBox = Box.createVerticalBox();
        aBox.setMinimumSize(new Dimension(width, height));
        aBox.setPreferredSize(new Dimension(width, height));
        aBox.setMaximumSize(new Dimension(500, 500));
        aBox.setSize(width, height);
//        aBox.setBorder(BorderFactory.createEmptyBorder(4, 4, 4, 4));
        JLabel label = new JLabel(labelString);
        listVal.setAlignmentX(Component.TOP_ALIGNMENT);
        aBox.add(label);
        aBox.add(new JSeparator(JSeparator.HORIZONTAL));
        aBox.add(listVal);
        aBox.add(Box.createVerticalGlue());
//        aBox.setAlignmentY(Component.LEFT_ALIGNMENT);

        return aBox;
    }

    private JPanel createView() {

        searchLists = new JList[3];
        searchListModel = new DefaultListModel[3];

        activeLists = new JList[5];
        activeListModel = new DefaultListModel[5];

        JPanel searchResults = new JPanel();
        searchResults.setLayout(new BoxLayout(searchResults,
BoxLayout.LINE_AXIS));
        searchResults.setMinimumSize(new Dimension(this.getWidth(),
(int)(0.6 * this.getHeight()) - 40));
        searchResults.setPreferredSize(new Dimension(this.getWidth(),
(int)(0.6 * this.getHeight()) - 40));
        searchResults.setMaximumSize(new Dimension(1500, 500));
        searchResults.setSize(this.getWidth(), (int)(0.6 *
this.getHeight()) - 40);
```

```
        JPanel active = new JPanel();
        active.setLayout(new BoxLayout(active, BoxLayout.LINE_AXIS));
        active.setSize(this.getWidth(), (int)( 0.4 * this.getHeight()) -
40);

        for(int i = 0; i<3; i++) {

            searchListModel[i] = new DefaultListModel();
            searchLists[i] = createJList(searchResults.getWidth()/3,
searchResults.getHeight(), searchListModel[i]);

        }

        for(int i = 0; i < 5; i++) {
            activeListModel[i] = new DefaultListModel();
            activeLists[i] = createJList(active.getWidth() / 5,
active.getHeight(), activeListModel[i]);
        }

        searchResults.add(createBoxPanel(searchResults.getWidth()/3,
searchResults.getHeight(), "File name", searchLists[0]));
        searchResults.add(new JSeparator(JSeparator.VERTICAL));
        searchResults.add(Box.createHorizontalGlue());
        searchResults.add(createBoxPanel(searchResults.getWidth()/3,
searchResults.getHeight(), "File size", searchLists[1]));
        searchResults.add(new JSeparator(JSeparator.VERTICAL));
        searchResults.add(Box.createHorizontalGlue());
        searchResults.add(createBoxPanel(searchResults.getWidth()/3,
searchResults.getHeight(), "File owner", searchLists[2]));

//        searchResults.setBorder(BorderFactory.createEmptyBorder(4, 0,
4, 0));

        JScrollPane searchResultsScroll = new JScrollPane(searchResults);

searchResultsScroll.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORI
ZONTAL_SCROLLBAR_NEVER);

        active.add(createBoxPanel(active.getWidth()/5,
active.getHeight(), "File name", activeLists[0]));
        active.add(new JSeparator(JSeparator.VERTICAL));
        active.add(Box.createHorizontalGlue());
        active.add(createBoxPanel(active.getWidth()/5,
active.getHeight(), "File size", activeLists[1]));
        active.add(new JSeparator(JSeparator.VERTICAL));
        active.add(Box.createHorizontalGlue());
        active.add(createBoxPanel(active.getWidth()/5,
active.getHeight(), "File path", activeLists[2]));
        active.add(new JSeparator(JSeparator.VERTICAL));
        active.add(Box.createHorizontalGlue());
        active.add(createBoxPanel(active.getWidth()/5,
active.getHeight(), "Current Throughput", activeLists[3]));
        active.add(new JSeparator(JSeparator.VERTICAL));
        active.add(Box.createHorizontalGlue());
        active.add(createBoxPanel(active.getWidth()/5,
active.getHeight(), "Mean Throughput", activeLists[4]));

        JScrollPane activeScroll = new JScrollPane(active);
```

```
activeScroll.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_
SCROLLBAR_NEVER);


        JPanel panel = new JPanel();
        panel.setSize(this.getWidth(), this.getHeight());

//        panel.setBorder(BorderFactory.createEmptyBorder(4, 0, 4, 0));
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

        JLabel searchLabel = new JLabel("Search Results");
        searchLabel.setAlignmentY(Component.CENTER_ALIGNMENT);
        panel.add(searchLabel);
        panel.add(searchResultsScroll);

        JLabel activeFilesLabel = new JLabel("Active files");
        activeFilesLabel.setAlignmentY(Component.CENTER_ALIGNMENT);
        panel.add(activeFilesLabel);
        panel.add(activeScroll);

        return panel;
    }

    public void connectToServer(String ip, String port, String alias) {

        try {

            clientSocket = new Socket(ip, Integer.parseInt(port));

            outToServer = new PrintWriter(clientSocket.getOutputStream(),
true);
            inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            connActive = true;

            outToServer.println("ALIAS " + alias);
            outToServer.println("IP " + acceptIp);
            outToServer.println("PORT " + acceptPort);
            System.out.println("Ip : "+acceptIp + " Port : "+acceptPort);
            String temp = inFromServer.readLine();
            this.alias = temp.substring(8);

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    private static void createAndShowGUI() {

        //Create and set up the window.
        NapsterClient frame = new NapsterClient();

    }

    public static void main(String[] args) {
            //Schedule a job for the event-dispatching thread:a
            //creating and showing this application's GUI.
```

```
            javax.swing.SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                            createAndShowGUI();
                    }
            });
    }

}
```

Κλάση SharedFile.java

```
package napsterclient;

public class SharedFile {

    private String fileName;
    private String path;
    private long fileSize;

    //Transef information in case someone wants to download the file
    private String ownersIP;
    private int ownersPort;

    SharedFile() {

        fileName = "";
        path = "";
        fileSize = 0;

    }

    SharedFile(String fileName, String path, long fileSize) {

        this.fileName = fileName;
        this.path = path;
        this.fileSize = fileSize;

    }

    SharedFile(String fileName, String path, long fileSize, String
ownersIP, int ownersPort) {

        this.fileName = fileName;
        this.path = path;
        this.fileSize = fileSize;

        this.ownersIP = ownersIP;
        this.ownersPort = ownersPort;

    }

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public long getFileSize() {
        return fileSize;
    }

    public void setFileSize(long fileSize) {
        this.fileSize = fileSize;
```

```
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public String getOwnersIP() {
        return ownersIP;
    }

    public void setOwnersIP(String ownersIP) {
        this.ownersIP = ownersIP;
    }

    public int getOwnersPort() {
        return ownersPort;
    }

    public void setOwnersPort(int ownersPort) {
        this.ownersPort = ownersPort;
    }

}
```

Κλάση TransferAcceptThread.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package napsterclient;

import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.LinkedList;
import javax.swing.DefaultListModel;


public class TransferAcceptThread extends Thread{

    private ServerSocket listeningSocket;
    private String threadIP;
    private int threadPort;
    private LinkedList<SharedFile> clientShares;
    private DefaultListModel[] activeListModel;
    private String alias;

    public TransferAcceptThread(String threadIP, int threadPort,
LinkedList<SharedFile> clientShares, DefaultListModel[] activeListModel,
String alias) {

        super();

        this.threadIP = threadIP;
        this.threadPort =threadPort;
        this.clientShares = clientShares;
        this.activeListModel = activeListModel;
        this.alias = alias;

    }

    public void run() {

        try {

            listeningSocket = new ServerSocket(threadPort, 0,
InetAddress.getByName(threadIP));

            while(true) {
                Socket connectionSocket = listeningSocket.accept();
                TransferThread temp = new
TransferThread(connectionSocket, clientShares, activeListModel, alias);
                temp.start();
            }

        } catch(IOException e) {
            e.printStackTrace();
        }

    }
    }
```

Κλάση TransferThread.java

```
package napsterclient;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.LinkedList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.DefaultListModel;


public class TransferThread extends Thread{

    Socket transferSocket;
    private BufferedReader inFromClient;
    private PrintWriter outToClient;

    private boolean receiveThread;
    private String requestedFileName;
    private String alias;
    private int position;

    private LinkedList<SharedFile> clientShares;
    private DefaultListModel[] activeListModel;

    private final int BUFFER_SIZE = 5000000;

    public TransferThread(String ip, int port, String requestedFileName,
LinkedList<SharedFile> clientShares, DefaultListModel[] activeListModel,
String alias, int position) {

        try {

            transferSocket= new Socket(ip, port);
            outToClient = new
PrintWriter(transferSocket.getOutputStream(), true);
            inFromClient = new BufferedReader(new
InputStreamReader(transferSocket.getInputStream()));

        } catch (IOException e) {
            e.printStackTrace();
        }

        receiveThread = true;
        this.requestedFileName = requestedFileName;
        this.clientShares = clientShares;
        this.activeListModel = activeListModel;
```

```
        this.alias = alias;

        this.position = position;

    }

    public TransferThread(Socket transferSocket, LinkedList<SharedFile>
clientShares, DefaultListModel[] activeListModel, String alias) {

        this.transferSocket = transferSocket;

        try {

            inFromClient = new BufferedReader(new
InputStreamReader(transferSocket.getInputStream()));
            outToClient = new
PrintWriter(transferSocket.getOutputStream(), true);

        } catch (IOException e) {
            e.printStackTrace();
        }

        receiveThread = false;
        this.clientShares = clientShares;
        this.activeListModel = activeListModel;
        this.alias = alias;

    }

    public void run() {

        String reply;

        if(receiveThread) { // The thread has to request the file from
the other client and start reading it

            outToClient.println("FILE " + requestedFileName);
            try {
                reply = inFromClient.readLine();
                System.out.println(reply);
                if(reply.equals("OK")) {
                    File dir = new File(alias);
                    dir.mkdir();
                    File downloadFile = new File(alias + "\\" +
requestedFileName);
                    if(!downloadFile.exists()) {
                        downloadFile.createNewFile();
                    }

                    //Maybe inform server about changes in active files
                    clientShares.add(new
SharedFile(downloadFile.getName(), downloadFile.getAbsolutePath(),
downloadFile.length()));
                    activeListModel[0].add(position,
downloadFile.getName());
                    activeListModel[1].add(position, "0");
                    activeListModel[2].add(position,
downloadFile.getAbsolutePath());
                    activeListModel[3].add(position, "0");
                    activeListModel[4].add(position, "0");
```

```
                    try {
                        FileOutputStream outFile;
                        outFile = new FileOutputStream(downloadFile);

                        DataInputStream fileToReceive = new
DataInputStream(transferSocket.getInputStream());


                        int bytesRead = 1;
                        byte[] fileInBytes = new byte[BUFFER_SIZE];

                        double start, end, elapsedTime;
                        double currentThrou, meanThrou = 0;


                        while(bytesRead > 0) {
                            start = System.nanoTime();
                            int totalRead = 0;
                            for(int i = 0; (i < 1000) && (bytesRead =
fileToReceive.read(fileInBytes)) > 0; i++) {
                                totalRead += bytesRead;
                                outFile.write(fileInBytes, 0, bytesRead);
                            }

                            end = System.nanoTime();

                            elapsedTime = end - start;
                            currentThrou = (totalRead/elapsedTime) *
1000000000;
                            meanThrou = (meanThrou + currentThrou) / 2;

                            activeListModel[1].set(position,
downloadFile.length());
                            activeListModel[3].set(position,
(long)currentThrou);
                            activeListModel[4].set(position,
(long)meanThrou);

                        }

                        outFile.flush();
                        outFile.close();

                        fileToReceive.close();

                    } catch(FileNotFoundException fe) {
                        System.out.println("File cannot be saved.");
                    }

                    System.out.println("Transfer completed " +
downloadFile.getPath());

                }//Maybe an error message that the infos the client had
is invalid right now

                    transferSocket.close();
                } catch(IOException e) {
                    e.printStackTrace();
                }
```

```
        } else {

            try {

                reply = inFromClient.readLine();
                System.out.println(reply);
                if(reply.startsWith("FILE")) {
                    //Search if the file is availiable
                    String fileName = reply.substring(5);
                    System.out.println("Need to transfer a file " +
fileName);

                    int i;
                    boolean flag = false;
                    for(i=0; i < clientShares.size() &&
!clientShares.get(i).getFileName().equals(fileName); i++);

                    if(i == clientShares.size()) {
                        System.out.println("No file");
                        outToClient.println("NOFILE");
                    } else {
                        outToClient.println("OK");

                        try {


                            FileInputStream inFile;
                            inFile = new
FileInputStream(clientShares.get(i).getPath());

                            DataOutputStream fileToTransfer = new
DataOutputStream(transferSocket.getOutputStream());



                            byte[] fileInBytes = new byte[BUFFER_SIZE];
                            int bytesRead;
                            while((bytesRead = inFile.read(fileInBytes))
> 0) {

                                fileToTransfer.write(fileInBytes, 0,
bytesRead); // stelnoume to arxeio
                            }

                            inFile.close();
                            fileToTransfer.flush();
                            fileToTransfer.close();


                        } catch(FileNotFoundException fe) {
                            System.out.println("File not found.");
                        }

                    }

                }

                transferSocket.close();
```

```
            } catch (IOException ex) {
                outToClient.println("NOFILE");

Logger.getLogger(TransferThread.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }

    }

}
```

9. Πηγές

http://www.p2pfoundation.net/Main_Page
Η p2p Foundation διατηρεί ένα wiki-based site, blog και newsletter με σκοπό την πληροφόρηση και την ανταλλαγή απόψεων για διάφορες πτυχές των peer to peer δικτύων.

http://en.wikipedia.org/wiki/Peer_to_peer
Μια εισαγωγή στην έννοια των peer to peer δικτύων από την online εγκυκλοπαίδεια wikipedia με πλήθος σχετικών links και αναφορών.

http://www.techweb.com/encyclopedia/defineterm.jhtml?term=peer-to-peer+network
Από την εγκυκλοπαίδεια τεχνολογίας TechEncyclopedia ορισμένες πληροφορίες και διαγράμμα- τα peer to peer δικτύων.

http://www.cachelogic.com/
Η cachelogic είναι μια εταιρεία παροχής λύσεων peer to peer δικτύωσης. Στην ιστιοσελίδα της υπάρχουν χρήσιμες πληροφορίες και ενδιαφέρουσα έρευνα σε θέματα peer to peer.

http://www.sandvine.co.uk/solutions/p2p_policy_mngmt.asp
Η Sandvine είναι μια εταιρεία που αναπτύσσει και εμπορεύεται εξοπλισμό δικτύων. Στην ιστιοσελίδα της υπάρχουν μερικά white papers για τα peer to peer δίκτυα.

http://www.p2pscience.org/
Το p2p science είναι ένα site εστιασμένο στις εφαρμογές των peer to peer δικτύων για την επι- στήμη. Περιέχει αρκετά links για software, blogs και forums σχετικά με τα peer to peer.

http://www.firstauthor.org/Downloads/P2P.pdf
Άρθρο για την συμβολή των peer to peer δικτύων στην επιστημονική έρευνα με πλήθος παραπο- μπών.

http://www.gnutella.com/
Η επίσημη ιστιοσελίδα του Gnutella. Περιέχει γενικές πληροφορίες, οδηγίες για νέα μέλη ,links και νέα που αφορούν την κοινότητα των χρηστών του.

http://www.idea-group.com/downloads/excerpts/Subramanian01.pdf
Core Concepts in Peer-to-Peer Networking Detlef Schoder, Kai Fischbach,Christian Schmitt, University of Cologne, Germany