ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΛΟΠΟΙΗΣΗ ΠΡΟΣΟΜΟΙΩΤΗ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΔΙΑΔΙΚΤΥΑΚΟΥ ΚΑΤΑΛΟΓΟΥ ΓΙΑ ΤΟ ΔΙΑΜΟΙΡΑΣΜΟ ΑΡΧΕΙΩΝ ΣΕ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ ΣΥΣΤΗΜΑΤΑ ΟΜΟΤΙΜΩΝ ΚΟΜΒΩΝ

IMPLEMENTATION OF A DISTRIBUTED DIRECTORY SERVICE FOR PEER-TO-PEER FILE SHARING IN LARGE SCALE NETWORKS

ΕΛΕΝΗ ΚΟΡΔΟΛΕΜΗ

ΕΠΙΒΛΕΠΩΝ : ΑΝΤΩΝΙΟΣ ΑΡΓΥΡΙΟΥ Λέκτορας Πανεπιστημίου Θεσσαλίας

ΒΟΛΟΣ

ΟΚΤΩΒΡΙΟΣ 2013

ΠΕΡΙΛΗΨΗ

Τα ομότιμα (peer-to-peer) συστήματα αποτελούν,ένα δημοφιλές μέσο, διαμέσου του οποίου είναι δυνατόν να διαμοιραστούν μεγάλα ποσά δεδομένων.

Τα peer-to-peer συστήματα είναι κατανεμημένα συστήματα στα οποία οι κόμβοι μπορούν να διαμοιράσουν πόρους,πληροφορίες και υπηρεσίες, ενώ εξακολουθούν να παραμένουν πλήρως αυτόνομοι.

Επιπρόσθετα τα συστήματα αυτά συμπεριλαμβάνουν την εξισορρόπηση του φόρτου εργασίας, την προσαρμοστικότητα,και την ανοχή σε αποτυχίες.

Στην παρούσα εργασία,παρουσιάζουμε την σχεδίαση και υλοποίηση μιας εφαρμογής προσομοιωτή κατανεμημένου διαδικτιακού καταλόγου,για τη διαχείριση και διαμοιρασμό αρχείων, σε μεγάλης κλίμακας peer-to-peer συστήματα.

Επιπρόσθετα,στο θεωρητικό μέρος της εργασίας,γίνεται αναφορά στα διάφορα είδη peer-to-peer συστημάτων καθως και στα χαρακτηριστικά και τις ιδιότητες τους.Επίσης αναλύονται οι διάφοροι δείκτες μέτρησης της απόδοσης αυτών των συστημάτων.

ABSTRACT

Peer-to-peer systems are popular means for transferring large amounts of data from one peer to another.

Peer-to-peer systems are distributed systems in which nodes can share resources, information and services, while they continue to remain completely independent.

In addition,those systems include workload balancing ,adaptability, and tolerance to failures .

In the present study, we present the design and implementation of a distributed directory service, for peer-to peer file sharing,in large scale networks.

Additionally, in the theoretical part of this study, we refer to various kinds of peer-to-peer systems, their characteristics and their properties .Also we analyze several indicators measuring the performance of these systems.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου, κ. Αργυρίου Αντώνιο, για την δυνατότητα που μου έδωσε να ασχοληθώ με τον συγκεκριμένο θέμα, για τις εύστοχες παρατηρήσεις και υποδείξεις του, και τη βοήθειά του.

Επίσης, ευχαριστώ θερμά τους γονείς μου Σπύρο και Φωτεινή και τον αδερφό μου Κωνσταντίνο, που με στήριξαν με κάθε τρόπο, σε όλη την διάρκεια των σπουδών μου.

Τέλος, θα ήθελα να ευχαριστήσω το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας για την υλικοτεχνική υποστήριξη που μου παρείχαν.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΠΡΟΛΟΓΟΣ

Η παρουσα πτυχιακη εργασία,εκπονήθηκε κατά τη διάρκεια του εαρινού εξαμήνου,του ακαδημαϊκού έτους 2013,με επιβλέποντα καθηγητή τον κ.Αργυρίου Αντώνιο,Λέκτορα του Πανεπιστημίου Θεσσαλίας .

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Τα ομότιμα (peer-to-peer) συστήματα αποτελούν,ένα δημοφιλές μέσο, διαμέσου του οποίου είναι δυνατόν να διαμοιραστούν μεγάλα ποσά δεδομένων.

Τα peer-to-peer συστήματα είναι κατανεμημένα συστήματα στα οποία οι κόμβοι μπορούν να διαμοιράσουν πόρους,πληροφορίες και υπηρεσίες, ενώ εξακολουθούν να παραμένουν πλήρως αυτόνομοι.

Επιπρόσθετα τα συστήματα αυτά συμπεριλαμβάνουν την εξισορρόπηση του φόρτου εργασίας, την προσαρμοστικότητα,και την ανοχή σε αποτυχίες.

1.1 ΣΤΟΧΟΙ ΤΗΣ ΠΤΥΧΙΑΚΗΣ

Στόχος της παρούσας εργασίας,είναι στο πρώτο μέρος της, να παρουσιαστούν και να αναλυθούν τα διάφορα χαρακτηριστικά και οι ιδιότητες ορισμένων δημοφιλών ομότιμων συστημάτων.

Στο δεύτερο μέρος της,παρουσιάζεται η υλοποίηση σε Java,ενός τέτοιου peer-to-peer συστήματος ιεραρχικής δομής,με στόχο την καλύτερη ανάλυση και κατανόηση της λειτουργείας του.

1.2 ΔΙΑΡΘΡΩΣΗ ΤΗΣ ΠΤΥΧΙΑΚΗΣ

Στο κεφάλαιο 2 αναλύονται τα γενικά χαρακτηριστικά και οι ιδιότητες των peer-to-peer συστημάτων**.**

Στο κεφάλαιο 3 αναλύεται ιδιαιτέρως το σύστημα Kazaa,ως χαρακτηριστικό παράδειγμα υπηρεσίας κατανεμημένου διαδικτυακού καταλόγου.

Στο κεφάλαιο 4 αναλύεται το πρόβλημα του σύγχρονου διαμοιρασμού,στα συστήματα κατανεμημένου διαμοιρασμού.

Στο κεφάλαιο 5 αναλύονται τα γενικά χαρακτηριστικά διαφόρων αδόμητων και δομημένων peer-to peer συστηματων

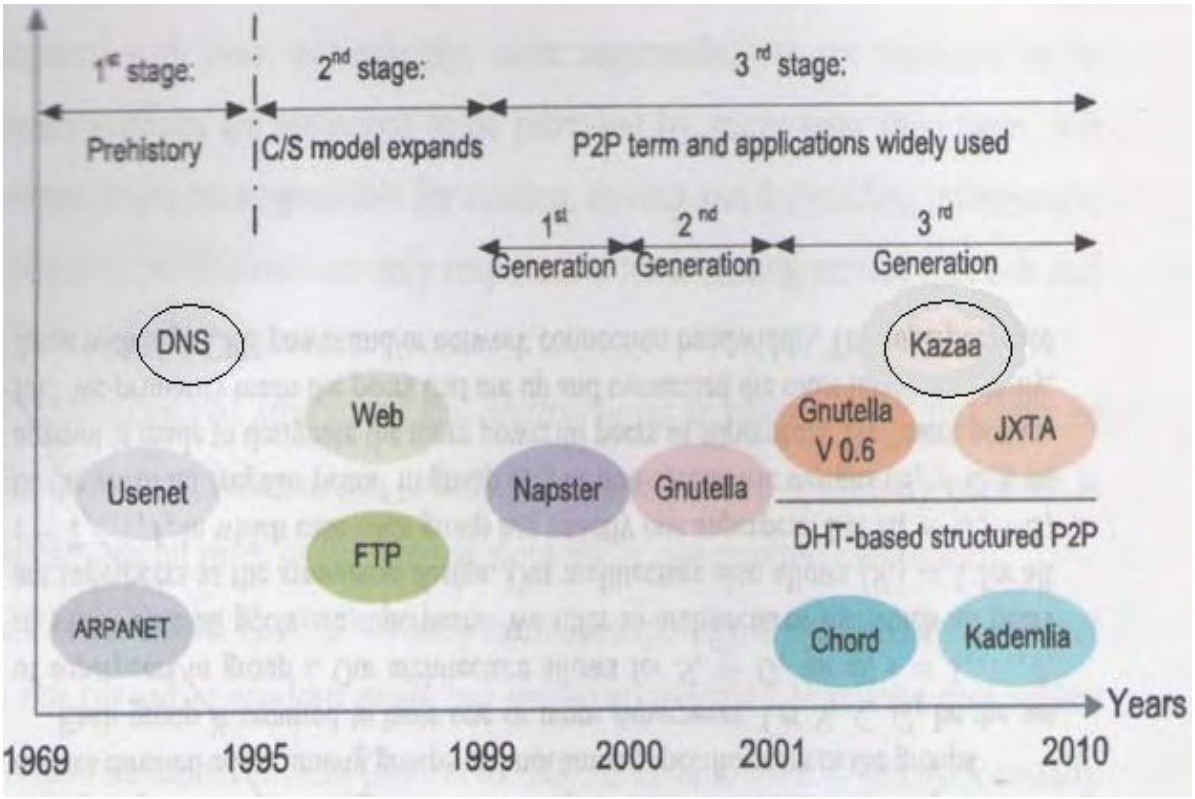
Στο κεφάλαιο 6 αναλύονται οι πιο βασικοί δείκτες μέτρησης της απόδοσης των peer-to peer συστηματων

Στο κεφάλαιο 7 δίνεται η υλοποίηση σε Java του κατανεμημένου διαδικτυακού καταλόγου**.**

ΚΕΦΑΛΑΙΟ 2

ΣΥΣΤΗΜΑΤΑ ΟΜΟΤΙΜΩΝ ΚΟΜΒΩΝ(PEER-TO-PEER NETWORKS)

2.1 ΙΣΤΟΡΙΚΗ ΕΞΕΛΙΞΗ ΤΩΝ ΣΥΣΤΗΜΑΤΩΝ ΟΜΟΤΙΜΩΝ ΚΟΜΒΩΝ (P2P NETWORKS)



ΣΧΗΜΑ 2.1

2.2 ΟΡΙΣΜΟΣ P2P ΔΙΚΤΥΟΥ (ΓΕΝΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ)

Σε γενικές γραμμές, ένα peer-to-peer (ή P2P) δίκτυο υπολογιστών,είναι κάθε δίκτυο που δεν έχει συγκεκριμένο αριθμό απο πελάτες και διακομιστές, αλλά ένα σύνολο από ομοτίμους κόμβους ,που λειτουργούν ταυτόχρονα ως πελάτες ή διακομιστές για τους άλλους κόμβους του δικτύου.

Αυτό το μοντέλο δικτύου αντιπαραβάλλεται με το μοντέλο client-server. Κάθε κόμβος είναι σε θέση να ξεκινήσει ή να ολοκληρώσει οποιαδήποτε συναλλαγή.

Οι ομότιμοι κόμβοι μπορεί να διαφέρουν όσον αφορά :

τοπική αναπαράσταση

ταχύτητα επεξεργασίας

εύρος ζώνης του δικτύου

αποθηκευτική ικανότητα

Τα Peer-to-Peer συστήματα,είναι κατανεμημένα συστήματα, αποτελούμενα από διασυνδεδεμένους κόμβους ,οι οποίοι είναι σε θέση να αυτο-οργανωθούν σε τοπολογίες δικτύων, με σκοπό το διαμοιρασμό πόρων, όπως: το περιεχόμενο τους, CPU κύκλους, αποθήκευση και εύρος ζώνης. Επίσης είναι σε θέση να προσαρμόζονται σε αποτυχίες και αλλαγές πληθυσμών των κόμβων, διατηρώντας παράλληλα τη συνδεσιμότητα και τη λειτουργία τους, χωρίς να απαιτείται η διαμεσολάβηση ή η υποστήριξη από κεντρικό διακομιστή .

Επειδή η πρόσβαση σε αποκεντρωμένους πόρους ,σημαίνει ότι λειτουργούν σε ένα περιβάλλον ασταθούς συνδεσιμότητας και απρόβλεπτων διευθύνσεων IP, οι peer-to-peer κόμβοι θα πρέπει να λειτουργούν εκτός του DNS και να έχουν σημαντική ή ολική αυτονομία.

2.3 ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ P2P ΔΙΚΤΥΩΝ

Τύποι Peer-to-Peer Αρχιτεκτονικών Δικτύου

1)Κεντρικοποιημένα (Centralized) P2P networks :μια βασική υπηρεσία, όπως αναζήτηση ή διαμοιρασμός των ID'S, παρέχεται από κάποιον κεντρικό server ο οποίος γίνεται single point of failure. (π.χ. Napster)

2)Κατανεμημένα (Distributed) P2P networks : όλοι οι κόμβοι είναι ομότιμοι, και κάθε κόμβος θα μπορούσε να αφήσει το δίκτυο χωρίς να υποβαθμίζεται το QoS του δικτύου.

Υπάρχουν δύο κύριες υποκατηγορίες:

Αδόμητα (Unstructured) P2P networks: δεν υπάρχει καμία δομή στο γράφο διασύνδεσης και τα περιεχόμενα τοποθετούνται σε κόμβους στο δίκτυο, χωρίς γνώση της τοπολογίας ή άλλης συσχέτισης με αυτό.(π.χ Gnutella,Bit torrent,Napster)

Δομημένα (Structured) P2P networks: οι κόμβοι οργανώνονται σε ένα δομημένο γράφο. Στα αντικείμενα που εισάγονται στο σύστημα αντιστοιχίζεται ένα κλειδί ,και η τοποθέτησή τους στους κόμβους γίνεται με προκαθορισμένο τρόπο ,έτσι ώστε να διευκολύνεται η αναζήτησή τους και να επιτυγχάνεται κλιμάκωση. (π.χ CAN, Chord, ,Pastry, Viceroy, Kademlia)

3)Υβριδικά (Hybrid) networks: μια μικτή αρχιτεκτονική μεταξύ Centralized και Distributed δίκτυων, που προσπαθεί να αποκτήσει τα πλεονεκτήματα και των δύο, χωρίς οποιοδήποτε από τα μειονεκτήματά τους. (π.χ FastTrack)

2.4 ΑΔΟΜΗΤΑ ΚΑΤΑΝΕΜΗΜΕΝΑ P2P ΔΙΚΤΥΑ

Unstructured P2P networks

Το P2P δίκτυο επικάλυψης, οργανώνει τους κόμβους σε τυχαίο γράφο με τρόπο **flat** ή **hierarchical** και χρησιμοποιεί το μηχανισμό της πλημμύρας (flooding),τυχαίες διαδρομές, ή expanding-ring Time-to-Live(TTL) search στο γράφο ,για να αναζητήσει περιεχόμενο, αποθηκευμένο στους κόμβους.

2.5 ΣΧΗΜΑΤΑ ΟΡΓΑΝΩΣΗΣ ΚΟΜΒΩΝ ΣΕ P2P ΔΙΚΤΥΟ

hierarchical P2P networks Χρησιμοποιούνται πολλαπλά (>=2) επίπεδα ιεραρχίας. Οι συμμετέχοντες σε ιεραρχικό δίκτυο P2P χωρίζονται σε διαφορετικούς ρόλους, για παράδειγμα, super-peer και peer με βάση τις δυνατότητες και την αξιοπιστία τους. Γενικότερα, η τοπολογία του δικτύου του κάθε επιπέδου μπορεί να είναι διαφορετική, πχ, το άνω επίπεδο μπορεί να είναι δομημένο P2P δίκτυο, ενώ το χαμηλότερο επίπεδο μπορεί να είναι αδόμητα P2P δίκτυο.(π.χ. KazaA)

flat P2P networks Οι συμμετέχοντες βρίσκονται σε ένα επίπεδο. Οι ρόλοι τους είναι ίσοι από την άποψη των αρμοδιοτήτων (π.χ.Gnutella)

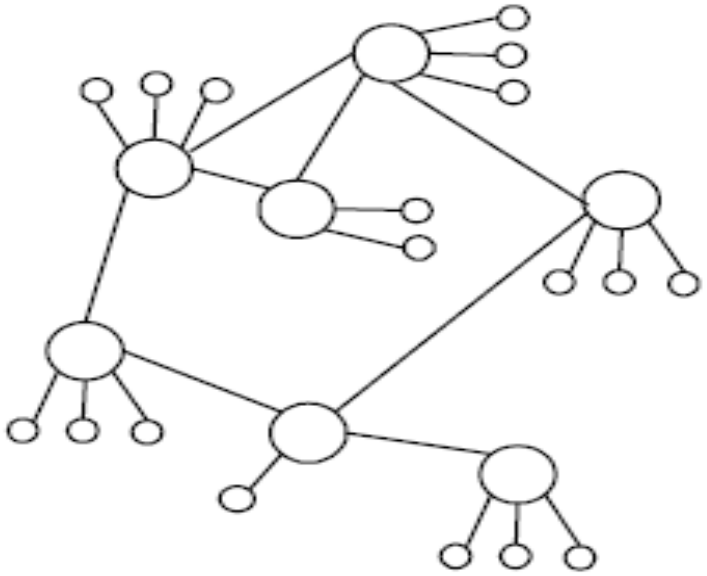
ΚΕΦΑΛΑΙΟ 3
ΕΝΑ ΠΑΡΑΔΕΙΓΜΑ ΥΠΗΡΕΣΙΑΣ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΔΙΑΔΙΚΤΥΑΚΟΥ ΚΑΤΑΛΟΓΟΥ:ΤΟ P2P ΔΙΚΤΥΟ KAZZA

Το KazaA ήταν μία υπηρεσία για P2P ανταλλαγή αρχείων μουσικής, βίντεο, εικόνας κλπ. Δημιουργήθηκε το 2001 από την Ολλανδική εταιρία Kazaa BV και χρησιμοποιούσε το πρωτόκολλο FastTrack. Τελικά το πρωτόκολλο αυτό άλλαξε ώστε οι clients να μην μπορούν πλέον να μιλούν με αυτό, κάτι που οδήγησε στο κλείσιμο της υπηρεσίας τον Αύγουστο του 2012.

3.1 ΤΟΠΟΛΟΓΙΑ ΤΟΥ KAZAA(ΥΒΡΙΔΙΚΟ P2P ΔΙΚΤΥΟ)

Σε αυτήν την τοπολογία, εισάγονται peers που λειτουργούν ως ηγέτες ομάδας. Αναφέρονται ως υπερκόμβοι (super nodes).

Οι υπερκόμβοι εκτελούν το ρόλο ενός κεντρικού εξυπηρετητή, όπως στην κεντρικοποιημένη τοπολογία, αλλά μόνο για ένα υποσύνολο των peers. Οι ίδιοι οι υπερκόμβοι είναι συνδεμένοι με έναν αποκεντρωμένο τρόπο. Επομένως, αυτή η υβριδική τοπολογία εισάγει δύο διαφορετικά επίπεδα ελέγχου. Στο πρώτο, οι συνηθισμένοι peers συνδέονται με τους υπερκόμβους σε μια τοπολογία κεντρικής οντότητας. Στο δεύτερο οι υπερκόμβοι συνδέονται ο ένας με τον άλλον σε μια αποκεντρωμένη τοπολογία.



ΣΧΗΜΑ 3.1

Οι υπερκόμβοι διατηρούν μια βάση δεδομένων που ταιριάζει τα ονόματα των αρχείων στις διευθύνσεις IP όλων των peers που συμμετέχουν.Η βάση δεδομένων του υπερκόμβου παρακολουθεί μόνο τους peers της ομάδα του. Αυτό μειώνει πολύ το σύνολο των peers που πρέπει να εξυπηρετήσει. Έτσι οποιοσδήποτε peer με μια σύνδεση υψηλής ταχύτητας θα είναι κατάλληλος για να γίνει ένας υπερκόμβος. Το καλύτερο παράδειγμα μιας P2P εφαρμογής που χρησιμοποιεί μια τέτοια τοπολογία είναι το **Kazaa/FastTrack.**

3.2 ΣΥΓΚΡΙΣΗ P2P ΔΙΚΤΥΩΝ ΩΣ ΠΡΟΣ ΤΟ ΒΑΘΜΟ ΚΕΝΤΡΙΚΟΠΟΙΗΣΗΣ ΤΟΥΣ

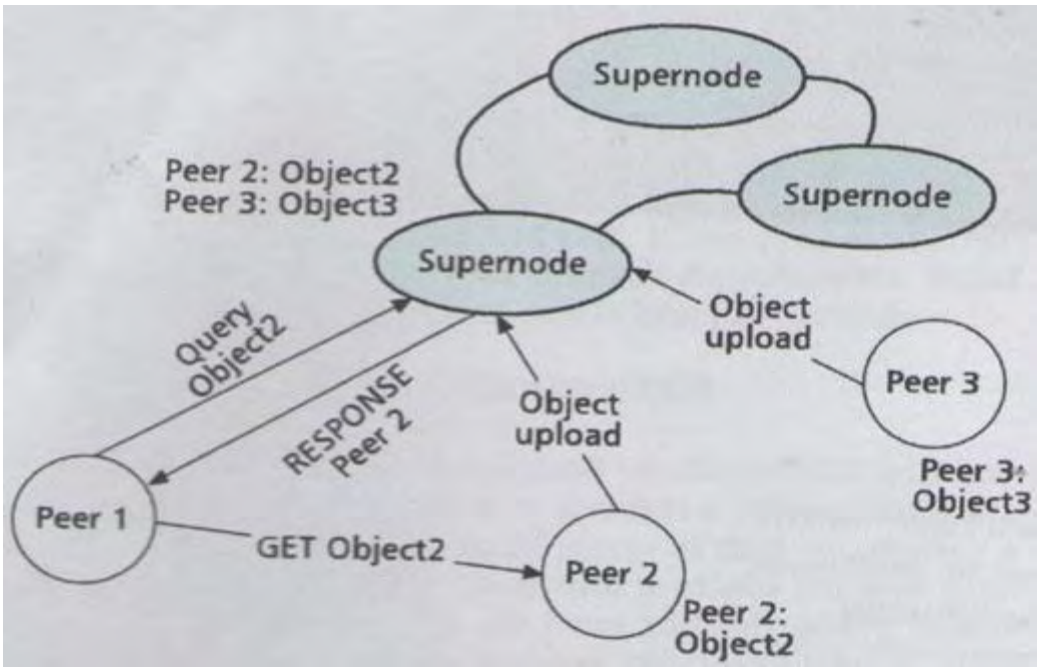
Σε ένα κεντρικοποιημένο σύστημα, όπως είναι ο Napster, η αναζήτηση πραγματοποιείται σε ένα κεντρικοποιημένο φάκελο, αλλά η μεταφορά αρχείων εξακολουθεί να συμβαίνει με ισότιμο τρόπο.

Στα «καθαρά» (pure) συστήματα, όπως είναι το Gnutella και το Freenet όλοι οι peers έχουν ισοδύναμους ρόλους και ευθύνες σε όλα τα θέματα.

Τα super-peer δίκτυα όπως είναι το **KaZaA** ,αποτελούν μια ενδιάμεση κατάσταση μεταξύ των κεντρικοποιημένων και καθαρών συστημάτων. Οι πελάτες αποστέλλουν ερωτήσεις στο super-peer και λαμβάνουν απαντήσεις από αυτόν, όπως και σε ένα κεντρικοποιημένο σύστημα. Όμως οι super-peers είναι επίσης συνδεδεμένοι μεταξύ τους όπως είναι οι peers σε ένα κατανεμημένο σύστημα. Επομένως οι super-peers αναλαμβάνουν να δρομολογήσουν τα μηνύματα, να υποβάλλουν και να απαντήσουν τις ερωτήσεις για λογαριασμό των πελατών τους.

3.3 ΤΟ ΧΡΗΣΙΜΟΠΟΙΟΥΜΕΝΟ ΠΡΩΤΟΚΟΛΛΟ FASTTRACK/KAZAA

Το FastTrack P2P, είναι ένα αποκεντρωμένο σύστημα διαμοιρασμού αρχείων(file-sharing)που υποστηρίζει αναζήτηση μετα-δεδομένων. Οι κόμβοι ,συνθέτουν μια δομημένη επικάλυψη super-peer αρχιτεκτονικών ώστε να γίνεται πιο αποτελεσματική η αναζήτηση.



ΣΧΗΜΑ 3.2 Το δίκτυο Kazaa

Super-peers/supernodes : κόμβοι με υψηλό εύρος ζώνης, μεγάλο χώρο στο δίσκο, μεγάλη επεξεργαστική ισχύ, και γρήγορη σύνδεση στο Internet ,οι οποίοι έχουν προσφερθεί εθελοντικά να «εκλεχθούν» ώστε να διευκολύνουν την αναζήτηση ,με caching των μετα-δεδομένων.

Ordinary peers : κόμβοι οι οποίοι διαβιβάζουν τα μετα-δεδομένα από τα αρχεία δεδομένων που μοιράζονται με τους super-peers (upload).Όλα τα αιτήματα/ερωτήσεις(queries),επίσης διαβιβάζονται στο super-peer.

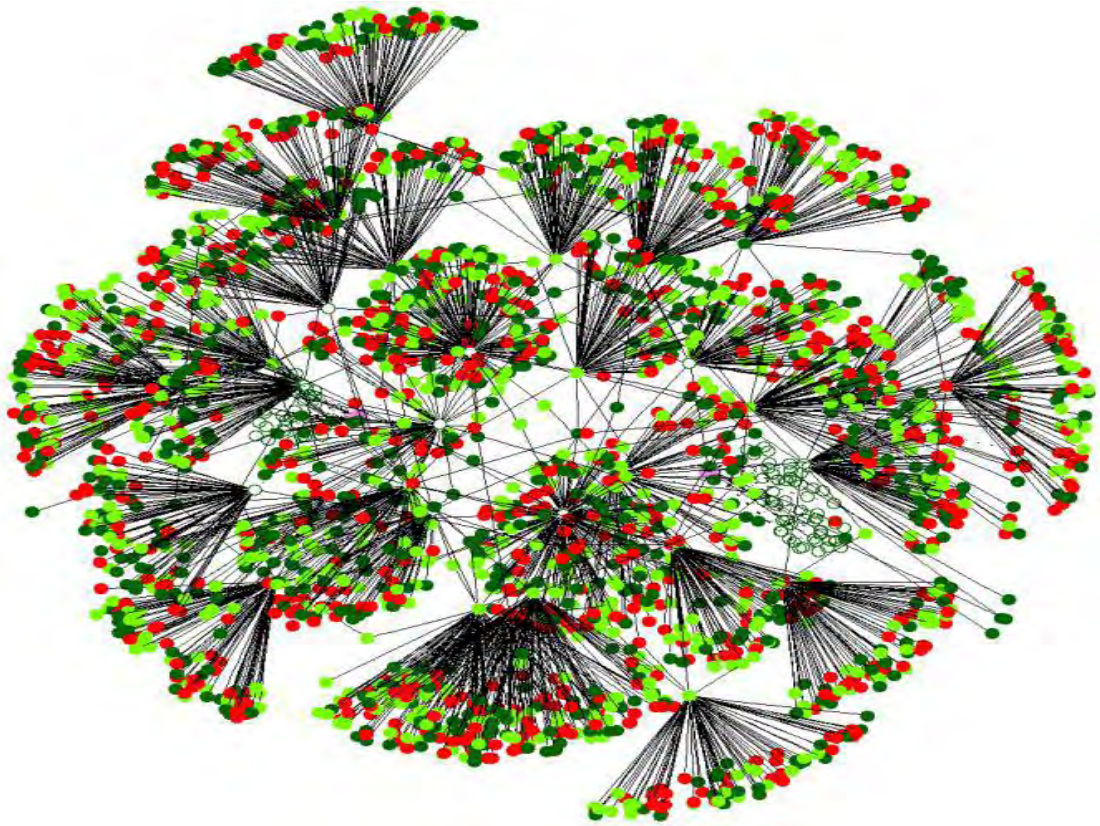
Στη συνέχεια αν ο άμεσος super-peer δε γνωρίζει την απάντηση, τότε Gnutella τύπου, βασισμένη στην εκπομπή, αναζήτηση (flood), εκτελείται σε ένα εξαιρετικά κλαδεμένο δίκτυο επικάλυψης των super-peers.

3.4 ΕΝΑΛΛΑΚΤΙΚΗ ΔΟΜΗ ΚΑΙ ΠΡΩΤΟΚΟΛΛΟ

Το σύστημα P2P μπορεί να υπάρξει χωρίς super-peers, αλλά αυτό θα μπορούσε να οδηγήσει σε χειρότερη καθυστέρηση της αναζήτησης(query). Ωστόσο, αυτή η προσέγγιση εξακολουθεί να καταναλώνει εύρος ζώνης ,τόσο όσο στην περίπτωση όπου διατηρείται πίνακας με δείκτες, στους super-peers,για λογαριασμό των peers που είναι συνδεδεμένοι σε αυτούς.

Οι super-peers εξακολουθούν να χρησιμοποιούν ένα πρωτόκολλο μετάδοσης για αναζήτηση, και τα ερωτήματα αναζήτησης(look-up queries) κατευθύνονται προς τους peers και τους super-peers, οι οποίοι έχουν τις σχετικές πληροφορίες για το ερώτημα.

Δύο κλασικά παραδείγματα εφαρμογών, που χρησιμοποιούν το πρωτόκολλο FastTrack ,είναι το **KaZaA** και το **Crokster .**



ΣΧΗΜΑ 3.3 Δομή ενός ιεραρχικού δικτυου

3.5 ΕΠΕΚΤΑΣΙΜΟΤΗΤΑ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ KAZAA

Ο αριθμός των peers που μπορούν να ορισθούν ως super-nodes, ποικίλει από δέκα έως αρκετές χιλιάδες. Αυτό συμβαίνει επειδή αυτοί οι super-nodes είναι οι ίδιοι συνηθισμένοι κόμβοι, που μπορούν να συμμετάσχουν ή να εγκαταλείψουν το δίκτυο, ανα πασα χρονική στιγμή. Επομένως, το δίκτυο είναι δυναμικό και πάντα μεταβαλλόμενο. Προκειμένου να εξασφαλιστεί η σταθερή διαθεσιμότητα του δικτύου, υπάρχει η ανάγκη ύπαρξης ενός αφοσιωμένου peer (ή αρκετών από αυτούς) που θα ελέγχει και θα παρακολουθεί το δίκτυο.Ένας τέτοιος peer καλείται κόμβος έναρξης και πρέπει πάντα να είναι on-line.

Όταν ένας πελάτης του FastTrack, παραδείγματος χάριν το Kazaa, εκτελείται σε έναν peer, αρχικά θα έρθει σε επαφή με τον κόμβο έναρξης. Ο κόμβος έναρξης θα καθορίσει εάν εκείνος ο ιδιαίτερος peer είναι κατάλληλος να είναι ένας super-node. Εάν κριθεί κατάλληλος, του παρέχονται, μερικές (εάν όχι όλες) διευθύνσεις IP άλλων super-nodes. Εάν είναι κατάλληλος να είναι μόνο συνηθισμένος peer, ο κόμβος έναρξης θα αποκριθεί με την παροχή της διεύθυνσης IP ενός από τους super-nodes.

Ορισμένοι πελάτες του FastTrack, όπως το Kazaa χρησιμοποιούν μια μέθοδο γνωστή ως «σύστημα φήμης (reputation system)», όπου η φήμη ενός ορισμένου χρήστη απεικονίζεται από το επίπεδο συμμετοχής τους στο δίκτυο (ένας αριθμός μεταξύ 0 και 1.000). Όσο πιο μακροχρόνιες είναι οι παραμονές των χρηστών που συνδέονται με το δίκτυο, τόσο υψηλότερο θα είναι το επίπεδο συμμετοχής του, το οποίο στη συνέχεια σημαίνει ότι θα ευνοηθεί περισσότερο από τις πολιτικές αναμονής και ως εκ τούτο θα λάβει καλύτερες υπηρεσίες. Αυτό πρόκειται κυρίως να ενθαρρύνει τους χρήστες να μοιραστούν τα αρχεία τους και έτσι αποτελεσματικά να μειώσει τον αριθμό «πελατών-επιβατών» στο δίκτυο.

Η ανακάλυψη των πόρων ολοκληρώνεται μέσω εκπομπής (broadcast)μεταξύ των super-nodes. Όταν ένας κόμβος από τη δεύτερη σειρά (ordinary peer) κάνει μια ερώτηση, κατευθύνεται αρχικά στον super-node του, ο οποίος θα εκπέμψει στη συνέχεια την ίδια ερώτηση σε όλους τους άλλους super-nodes, με τους οποίους συνδέεται αυτήν την στιγμή. Αυτό επαναλαμβάνεται ωσότου γίνει το TTL ,εκείνης της ερώτησης ,ίσο με μηδέν. Έτσι,

εάν παραδείγματος χάριν, το TTL μιας ερώτησης τίθεται στο 7, και ο μέσος αριθμός κόμβων ανά super-node είναι 10, ένας πελάτης του FastTrack, είναι σε θέση να ψάξει 11 φορές περισσότερους κόμβους σε ένα δίκτυο.

3.6 ΣΥΓΚΡΙΣΗ FastTrack/KAZZA ΜΕ Gnutella

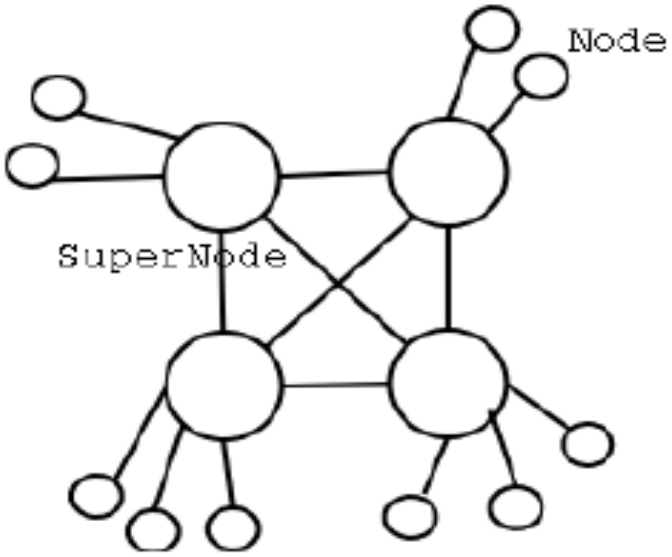
Το FastTrack, παρέχει στους πελάτες του,μεγαλύτερη κάλυψη και καλύτερα αποτελέσματα αναζήτησης.

Εντούτοις, υπάρχει ένα μειονέκτημα σε μια τέτοια μέθοδο εκπομπής (πλημμύρα), και αυτό είναι η περιττή ποσότητα δεδομένων που πρέπει να μεταφερθεί από super-node σε super-node.Αυτό είναι περίπου το ίδιο πρόβλημα που έχει και το δίκτυο Gnutella.

Παρόλα αυτά, δεν αποτελεί σοβαρό πρόβλημα για το Kazaa, σε αντιδιαστολή με το Gnutella, επομένως οι super-nodes είναι κόμβοι που είναι εγγυημένοι στο να έχουν γρήγορες συνδέσεις.

Κάθε ένας από τους super-nodes που έλαβαν την ερώτηση, εκτελεί μια αναζήτηση στη βάσης δεδομένων του, που περιέχει τις πληροφορίες όλων των κοινών αρχείων στους συνδεδεμένους κόμβους του. Μόλις βρεθεί ένα ταίριασμα, μια απάντηση θα σταλεί μέσω της ίδιας πορείας που η ερώτηση αναζήτησης διαδόθηκε από τον αρχικό κόμβο. Αυτή η μέθοδος δρομολόγησης είναι παρόμοια με του Gnutella, και διατρέχει τον κίνδυνο του ίδιου προβλήματος στο να χαθούν οι απαντήσεις όπως καθοδηγούνται πίσω ,μέσω του δικτύου.

Αυτό οφείλεται στο γεγονός ότι οι peers του δικτύου Gnutella, όπως αναφέρθηκε προηγουμένως, συνδέονται και αποσυνδέονται από το δίκτυο πολύ συχνά. Αυτό σημαίνει, ότι τα πακέτα απάντησης που καθοδηγούνται πίσω, μπορούν να χαθούν στην πορεία, επειδή ένας ή περισσότεροι από τους κόμβους που αποτελούν τη σύνδεση αποσυνδέθηκα ,και έτσι δεν είναι πλέον εκεί. Το προαναφερθέν πρόβλημα, εντούτοις, δεν είναι ιδιαίτερα σοβαρό για τους χρήστες του Kazaa ,καθώς οι peers του FastTrack ,έχουν μεγάλες συνδέσεις υψηλής ταχύτητας (super-nodes) και ως εκ τούτου οι πορείες επιστροφής μπορούν να θεωρηθούν πιο αξιόπιστες.



ΣΧΗΜΑ 3.4 Τοπολογία ενός δικτύου τύπου KAZAA

3.7 ΖΗΤΗΜΑΤΑ ΑΠΟΔΟΣΗΣ ΤΟΥ KAZAA

Όπως αναφέρθηκε το Kazaa, βασίζεται στο ιδιόκτητο πρωτόκολλο FastTrack, το οποίο χρησιμοποιεί ειδικά διαμορφωμένους super-nodes ,με μεγαλύτερη σύνδεση εύρους ζώνης(bandwidth).

Δείκτες στα δεδομένα του κάθε peer, αποθηκεύονται σε κάθε συσχετιζόμενο super-peer, και όλα τα queries υποβάλλονται στους super-peers.

Αν και αυτή η προσέγγιση φαίνεται να προσφέρει καλύτερες ιδιότητες κλιμάκωσης/διαβάθμισης σε σχέση με το Gnutella ,ο σχεδιασμός δεν έχει αναλυθεί.

Η μεταφορά αρχείων στο Kazaa,αποτελείται από μη κρυπτογραφημένες HTTP μεταφορές. Όλες οι μεταφορές περιλαμβάνουν KaZaA-specific HTTP κεφαλίδες. (π.χ., X-KaZaA-IP). Αυτές οι επικεφαλίδες βοηθούν να διακριθούν οι δραστηριότητες KaZaA από άλλες δραστηριότητες HTTP.

Μια power-law τοπολογία, που βρίσκεται συνήθως σε πολλά πρακτικά HTTP δίκτυα, όπως WWW ,έχει την ιδιότητα ότι ένα μικρό ποσοστό των peers, έχουν υψηλό βαθμό εξόδου ενώ η συντριπτική πλειοψηφία των peers,έχουν χαμηλό βαθμό εξόδου.

Τυπικά η συχνότητα fd των peers με βαθμό εξόδου d, περιγράφεται από τη σχέση $fd \propto d\text{^}a; a < 0$. (Ιδιότητα Zipf , με Zipf γραμμική κατανομή όταν σχεδιάζεται σε log-log κλίμακα. Οι τοπολογίες δρομολόγησης στο Internet ακολουθούν αυτή τη σχέση (power-law) με $a \approx -2$.)

Όμως το φορτίο εργασίας (workload) του διαμοιρασμού αρχείων(file sharing)του Kazaa δεν ακολουθεί την κατανομή Zipf. Αυτό συμβαίνει επειδή, η δημοτικότητα των αντικειμένων του Kazaa τείνει να είναι σύντομη, και τα δημοφιλή αντικείμενα είναι συνήθως εκείνα που δημιουργήθηκαν πρόσφατα. Υπάρχει επίσης σημαντική τοπικότητα στο workload.

3.8 ΛΕΙΤΟΥΡΓΙΕΣ ΤΟΥ KAZZA(ΣΥΝΟΠΤΙΚΑ)

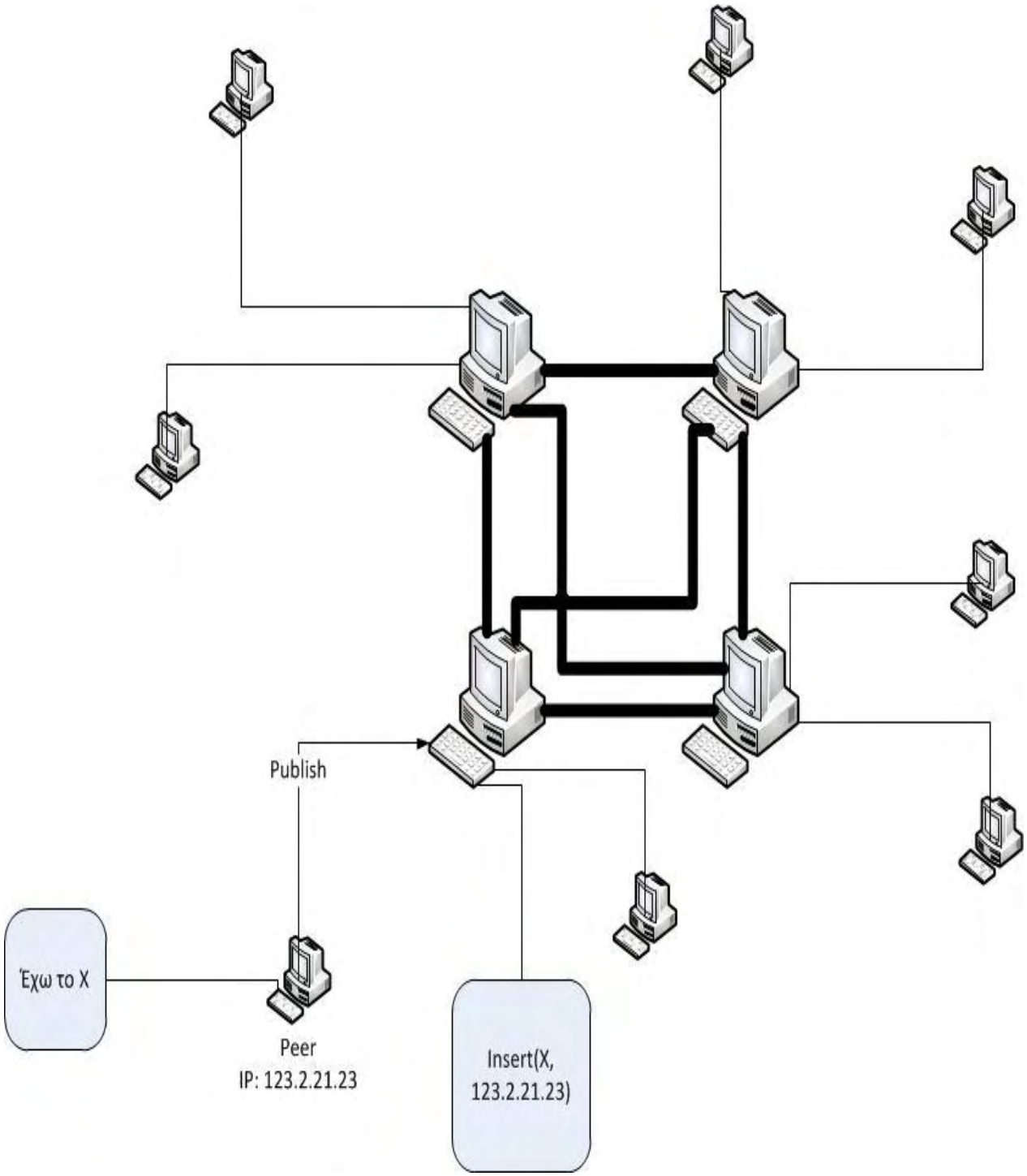
Το KazaA χρησιμοποιεί έξυπνο query flooding και εκτελεί τις ακόλουθες διεργασίες

- **Join**: ξεκινώντας ο client επικοινωνεί με ένα super-node (στη συνέχεια μπορεί να γίνει και ο ίδιος)

- **Publish**: ο peer αναφέρει τη λίστα των αρχείων που διαθέτει στον super-node του
- **Search**: ο client κάνει ερώτηση στον super-node για κάποιο αρχείο. Οι super-nodes «πλημμυρίζουν» μεταξύ τους τα ερωτήματα

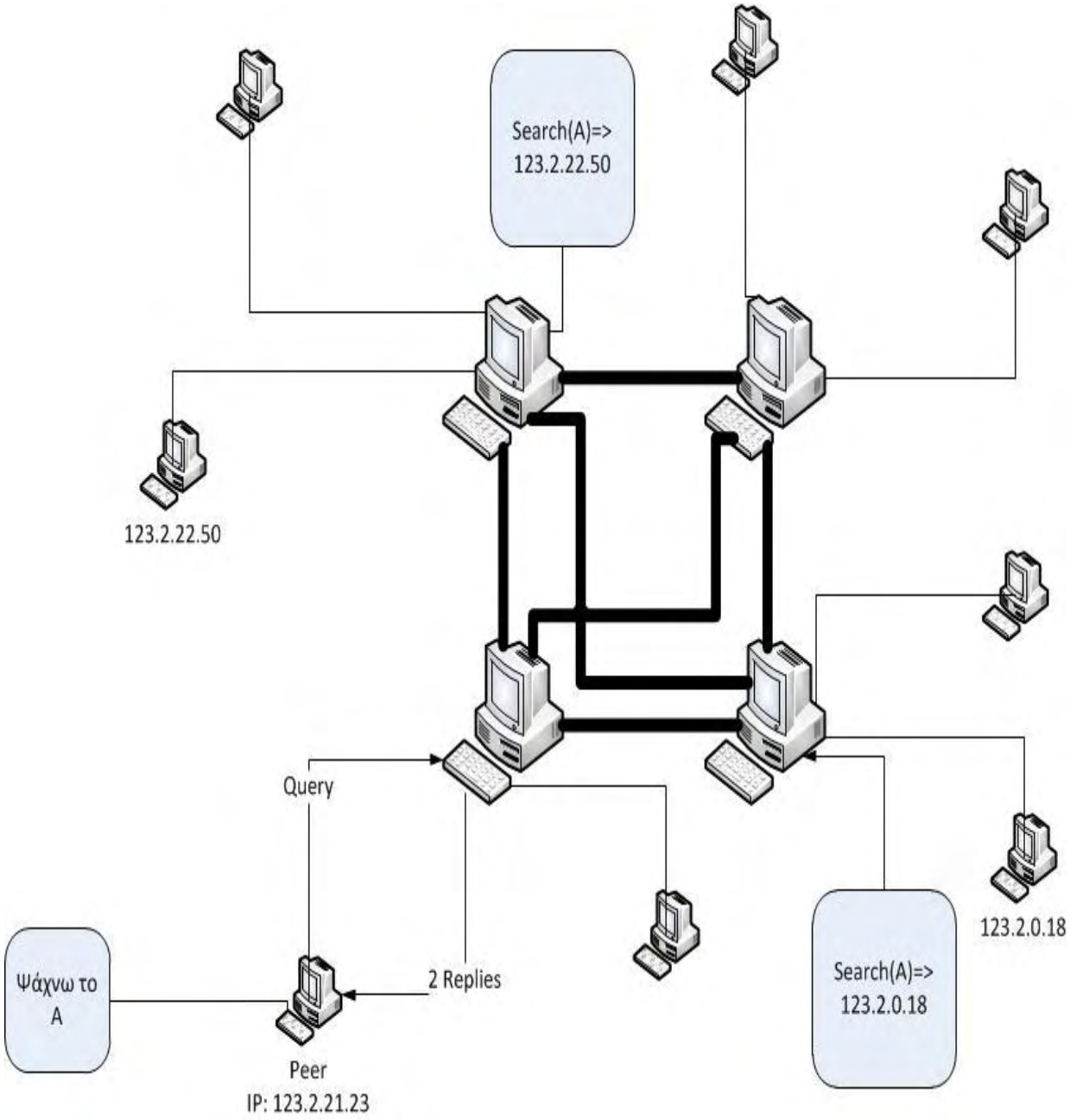
- **Fetch**: ο client λαμβάνει το αρχείο απευθείας από τον(τους) peer(s). Υπάρχει η δυνατότητα να παίρνει το αρχείο από πολλούς peers ταυτόχρονα.

Publish



ΣΧΗΜΑ 3.5

Search



ΣΧΗΜΑ 3.6

Ένας κόμβος στέλνει πρώτα query στον super-node του. Ο Super-node απαντά με όμοια αρχεία (matches). Αν βρεθούν x όμοια αρχεία, τελειώνει η αναζήτηση. Αλλιώς, ο super-node προωθεί το query σε υποσύνολο από super-nodes. Αν βρεθούν συνολικά x όμοια αρχεία, τελειώνει η αναζήτηση. Αλλιώς, το query προωθείται περαιτέρω, πιθανώς από τον αρχικό super-node, παρά διαδοχικά.

Fetching

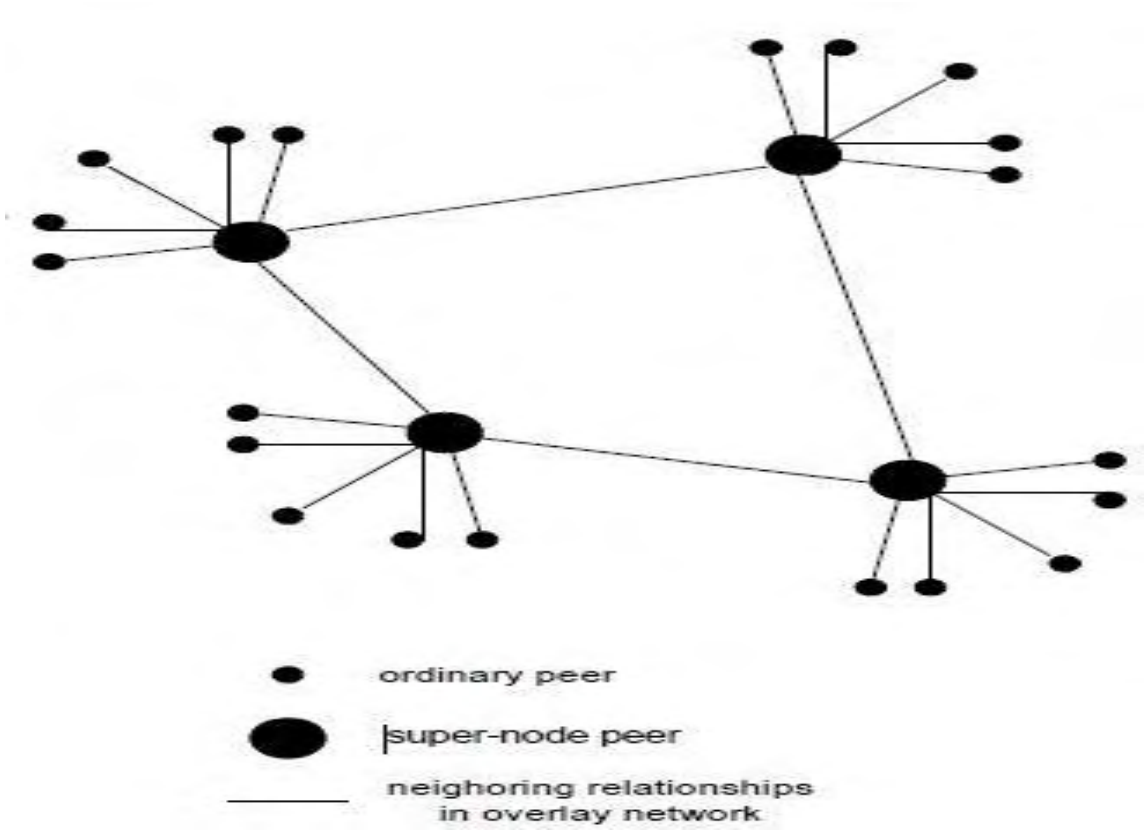
Αν το αρχείο βρεθεί σε πολλούς κόμβους, ο χρήστης μπορεί να επιλέξει παράλληλο κατέβασμα

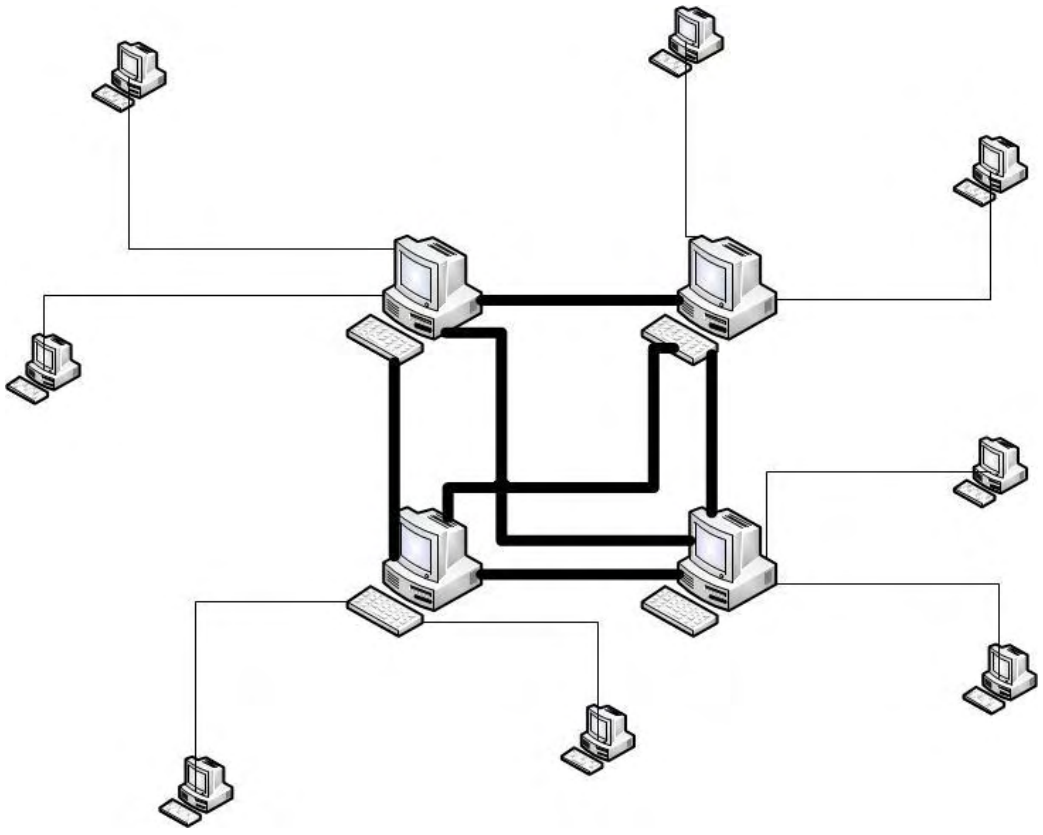
Αξιοποίηση της ετερογένειας

Η αρχιτεκτονική μεταξύ του κεντρικού καταλόγου και του query flooding αξιοποιεί την ετερογένεια μεταξύ των κόμβων (peers) του δικτύου. Κάθε peer είναι είτε super-node ή έχει ανατεθεί σε κάποιον. Κάθε peer συνδέεται με τον super-node του μέσω TCP και με τον ίδιο τρόπο συνδέονται και γειτονικά ζεύγη super-nodes. Κάθε super-node παρακολουθεί το περιεχόμενο των peers που είναι συνδεδεμένοι σε αυτόν.

Κίνητρα για να γίνει κάποιος peer super-node

- Συγχώνευση query
  - Πολλοί συνδεδεμένοι κόμβοι μπορούν να έχουν μόνο λίγα αρχεία.
  - Η διάδοση ενός ερωτήματος σε ένα κοινό κόμβο μπορεί να παίρνει περισσότερο χρόνο από το να απαντήσει ο ίδιος ο super-node.
- Ευστάθεια
  - Η επιλογή του super-node ευνοεί τους κόμβους που είναι πολύ ώρα ενεργοί.
  - Ο χρόνος στον οποίο ένας κόμβος είναι ενεργός είναι μία ικανοποιητική πρόβλεψη για το πόσο χρόνο θα είναι ενεργός στο μέλλον.





ΣΧΗΜΑ 3.7 Διαγραμματικη αναπαράσταση peers /super peers

3.8.1 Συντήρηση υπερκείμενου δικτύου(join νέου κόμβου)

Μία λίστα από τους εν δυνάμει super-nodes ,περιλαμβάνεται στο κατέβασμα του λογισμικού. Ο νέος peer διατρέχει τη λίστα αυτή μέχρι να βρει έναν ενεργό super-node. Στη συνέχεια στέλνει ping σε 5-6 super-nodes της λίστας και συνδέεται με τον πρώτο που θα απαντήσει. Κατά τη σύνδεση λαμβάνει μια πιο ενημερωμένη λίστα με super-nodes που βρίσκονται κοντά στον peer. Αν ο super-node για κάποιο λόγο πάψει να λειτουργεί, ο peer ανατρέχει στην ενημερωμένη λίστα που είχε κατεβάσει και βρίσκει νέο super-node.

3.8.2 Πλεονεκτήματα –Μειονεκτήματα του kazaa

Υπέρ

- Προσπάθεια για αξιοποίηση της ετερογένειας, όπως:
  - Του εύρου ζώνης
  - Της υπολογιστικής δυνατότητας των hosts
  - Της διαθεσιμότητας των hosts
- Αξιοποίηση της τοπικότητας

Κατά

- Εύκολα καταστρατηγούμενοι μηχανισμοί
- Δεν υπάρχει εξασφάλιση ως προς τον χώρο και το χώρο αναζήτησης

3.9 ΣΥΝΟΨΗ ΙΔΙΟΤΗΤΩΝ ΚΑΙ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΤΟΥ KAZZA

Το KaZaA είναι μια peer-to-peer εφαρμογή για το διαμοιρασμό των αρχείων βασισμένη στο πρωτόκολλο FastTrack. Το πρωτόκολλο FastTrack είναι βασισμένο στο Gnutella ,και το επεκτείνει με την προσθήκη των υπερκόμβων (supernodes), έτσι ώστε να βελτιώσει την κλιμακοσιμότητά του. Η λειτουργικότητα του supernode, είναι ενσωματωμένη σε κάθε κόμβο του δικτύου και επομένως κάθε κόμβος του δικτύου μπορεί να γίνει supernode κάποια στιγμή. Για να μπορεί να συνδεθεί ένας κόμβος στο δίκτυο, θα πρέπει αρχικά να έχει γνώση σχετικά με μια λίστα από γνωστούς supernodes. Αφού καταφέρει να συνδεθεί σε έναν από αυτούς, μπορεί στη συνέχεια να μάθει και για άλλους σε περίπτωση μελλοντικών συνδέσεων. Συνδεόμενος ένας κόμβος ,στέλνει στο supernode μια λίστα με τα αρχεία που θέλει να μοιραστεί. Όταν ένας κόμβος θέλει να στείλει αίτηση αναζήτησης, τότε αντίθετα με το Gnutella ,δε διαχειρίζεται ο ίδιος τη διαδικασία εύρεσης, αλλά αποστέλλει την αίτηση στο supernode. Ο supernode επικοινωνεί με τους υπόλοιπους supernodes, έτσι ώστε να ικανοποιήσει την αίτηση αναζήτησης. Στη συνέχεια, ο κόμβος που αρχικοποίησε την αίτηση αναζήτησης, συνδέεται απευθείας σε έναν peer για να λάβει τα δεδομένα. Η μεταφορά γίνεται με τη χρήση του πρωτοκόλλου HTTP.

Ο super-peer ,είναι ένας peer που λειτουργεί σαν ένας κεντρικοποιημένος εξυπηρετητής για ένα σύνολο από πελάτες. Οι super-peers συνδέονται μεταξύ τους σχηματίζοντας ένα «καθαρό» peer-to-peer σύστημα και αναλαμβάνουν να εξυπηρετήσουν τις αιτήσεις που στέλνουν οι πελάτες. Κάθε super-peer διατηρεί δύο πρωτόκολλα: ένα για να επικοινωνεί με τους υπόλοιπους super-peers, και ένα για να επικοινωνεί με τους πελάτες.

Ένα σύνολο από πελάτες μαζί με το super-peer σχηματίζουν ένα σύμπλεγμα (cluster). Η επικοινωνία μέσα σε ένα σύμπλεγμα, πραγματοποιείται με απευθείας συνδέσεις ανάμεσα στους πελάτες. Η επικοινωνία όμως ενός πελάτη του συμπλέγματος με έναν άλλο πελάτη ενός διαφορετικού συμπλέγματος ,είναι εφικτή διαμέσου του super-peer. Συγκεκριμένα, ο έλεγχος ροής για τους κανονικούς peers συμβαίνει διαμέσου του superpeer. Επειδή το super-peer δίκτυο συνδυάζει στοιχεία του υβριδικού συστήματος και του pure (distributed) συστήματος ,παρουσιάζει περισσότερα πλεονεκτήματα.

Συνδυάζει την απόδοση της κεντρικοποιημένης αναζήτησης με την αυτονομία, την ισορροπία φόρτου και την αντοχή σε επιθέσεις της κατανεμημένης αναζήτησης. Για παράδειγμα, επειδή οι super-peers λειτουργούν σαν κεντρικοποιημένοι εξυπηρετητές για τους πελάτες τους, μπορούν να χειριστούν τις ερωτήσεις πιο αποδοτικά από ότι θα έκανε ένας πελάτης. Επιπλέον όμως, επειδή υπάρχουν αρκετοί super-peers στο δίκτυο, κανένας super-peer δε χρειάζεται να αντιμετωπίσει ένα μεγάλο φόρτο εργασίας και επομένως να αποτελέσει σημείο αποτυχίας του συστήματος.

Η υλοποίηση του προσομοιωτή κατανεμημένου διαδικτυακού καταλόγου, για διαμοιρασμό αρχείων ,σε μεγάλης κλίμακας συστήματα ομότιμων κόμβων, που αποτελεί το προγραμματιστικό μέρος αυτής της εργασίας, βασίστηκε στα χαρακτηριστικά που παρουσιάζει το δίκτυο Kazza.

3.10 ΑΝΑΛΥΣΗ ΑΠΟΔΟΣΗΣ KAZAA ΔΙΚΤΥΟΥ P2P

N= #κόμβων επιπέδου

Χαμηλό Επίπεδο Ιεραρχίας(όπως κεντρικοποιημένο p2p NAPSTER):
O(1) ΑΝΑΖΗΤΗΣΗ, O(N) ΑΠΟΘΗΚΕΥΣΗ

Υψηλό Επίπεδο Ιεραρχίας(flood όπως στο Gnutella):
O(N) ΑΝΑΖΗΤΗΣΗ, O(1) ΑΠΟΘΗΚΕΥΣΗ

ΤΕΛΙΚΑ στο KaZaA:
O(ΑΝΑΖΗΤΗΣΗ)*O(ΑΠΟΘΗΚΕΥΣΗ_ΑΝΑ_ΚΟΜΒΟ)==O(N)
(trade-off μεταξύ των αποτελεσμάτων κάθε επιπέδου)

ΚΕΦΑΛΑΙΟ 4

ΣΥΣΤΗΜΑΤΑ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΔΙΑΜΟΙΡΑΣΜΟΥ ΚΑΙ ΤΟ ΠΡΟΒΛΗΜΑ ΤΟΥ ΣΥΓΧΡΟΝΟΥ ΔΙΑΜΟΙΡΑΣΜΟΥ(FILE DISTRIBUTION)

4.1 Διαμοιρασμός Αρχείων (File Sharing)

Ο διαμοιρασμός αρχείων (file sharing), είναι η διανομή ή η παροχή πρόσβασης σε ψηφιακά αποθηκευμένη πληροφορία, όπως προγράμματα Η/Υ, πολυμέσα (εικόνα, ήχος),έγγραφα, ή ηλεκτρονικά βιβλία (e-books). Υπάρχει ποικιλία μοντέλων διανομής, αποθήκευσης και μετάδοσης. Οι κοινές μέθοδοι είναι ο διαμοιρασμός με τη χρήση κινητών μέσων (όπως CD, DVD, flopy disk, μαγνητικές ταινίες, μνήμες flash), η αποθήκευση σε κεντρικούς servers, ‘hyperlinked’ έγγραφα στον παγκόσμιο ιστό, και η χρήση κατανεμημένων peer-to-peer (p2p) δικτύων.

Το πρόβλημα του διαμοιρασμού αρχείων απασχόλησε και παλιές, κατανεμημένες client-server εφαρμογές, ωστόσο αυτές παρουσίασαν δυσεπίλυτα προβλήματα (Sun RPC, NFS). Το κενό αυτό το κάλυψαν οι peer-to-peer εφαρμογές, οι οποίες στηρίζονται στην εξής, απλή ιδέα: κάθε χρήστης δημοσιοποιεί τα αρχεία που διαθέτει και πάνω σε αυτήν την πληροφορία γίνονται αναζητήσεις από τους υπόλοιπους χρήστες. Όταν κάποιος χρήστης εντοπίσει κάποιο αρχείο που επιθυμεί, ανοίγει μια απ' ευθείας σύνδεση με τον κάτοχο και το αντιγράφει στο σύστημα του. Υπάρχουν ποικίλα συστήματα αυτού του είδους, από απλές εφαρμογές με την ελάχιστη λειτουργικότητα μέχρι πιο εξειδικευμένα που παρέχουν επιπλέον υπηρεσίες, όπως κατανεμημένη αποθήκευση και άλλα.

Σε γενικές γραμμές προσφέρουν τα παρακάτω χαρακτηριστικά:

• Ανταλλαγή αρχείων.

• Ασφαλή αποθήκευση.

• Υψηλή διαθεσιμότητα.

• Ανωνυμία.

• Ευχρηστία.

Τα πιο σημαντικά ζητήματα σε αυτό το πεδίο εφαρμογής είναι η κατανάλωση εύρους ζώνης ,η ασφάλεια και η αναζήτηση.

4.2 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΣΥΓΧΡΟΝΟΥ ΔΙΑΜΟΙΡΑΣΜΟΥ

Έστω ένα σύστημα κατανεμημένου σύγχρονου διαμοιρασμού, όπου μια πηγή(server),δημιουργεί συρμό από τεμάχια ενός αντικειμένου πολυμέσων ,τα οποία στέλνονται σε ένα μικρό υποσύνολο των συμμετεχόντων χρηστών. Άρα κάθε χρήστης, ανταλλάσει τα τεμάχια αυτά με τους γείτονές του στο γράφο διαμοιρασμού, με τη χρήση ενός δυναμικού κατανεμημένου χρονοπρογραμματιστή ανταλλαγής, με σκοπό τη λήψη κάθε ενός ,από κάθε χρήστη-κόμβο, μέσα σε προκαθορισμένο χρονικό διάστημα, από τη δημιουργία του τεμαχίου. Οπότε για να επιτευχθεί ο στόχος, κάθε κόμβος διατηρεί και ανταλλάσει με τους γείτονές του, έναν πίνακα περιεχομένων τεμαχίων.

Άρα, το ζητούμενο είναι, να παρέχεται αρκετό εύρος ζώνης στους slow peers ,ενός δικτύου ιεραρχικής δομής, έτσι ώστε ο συρμός να διαμοιραστεί σε αυτούς ,με τρόπο τέτοιο ώστε να αποκτήσουν κάθε τεμάχιο του συρμού, μέσα στο προκαθορισμένο χρονικό διάστημα.

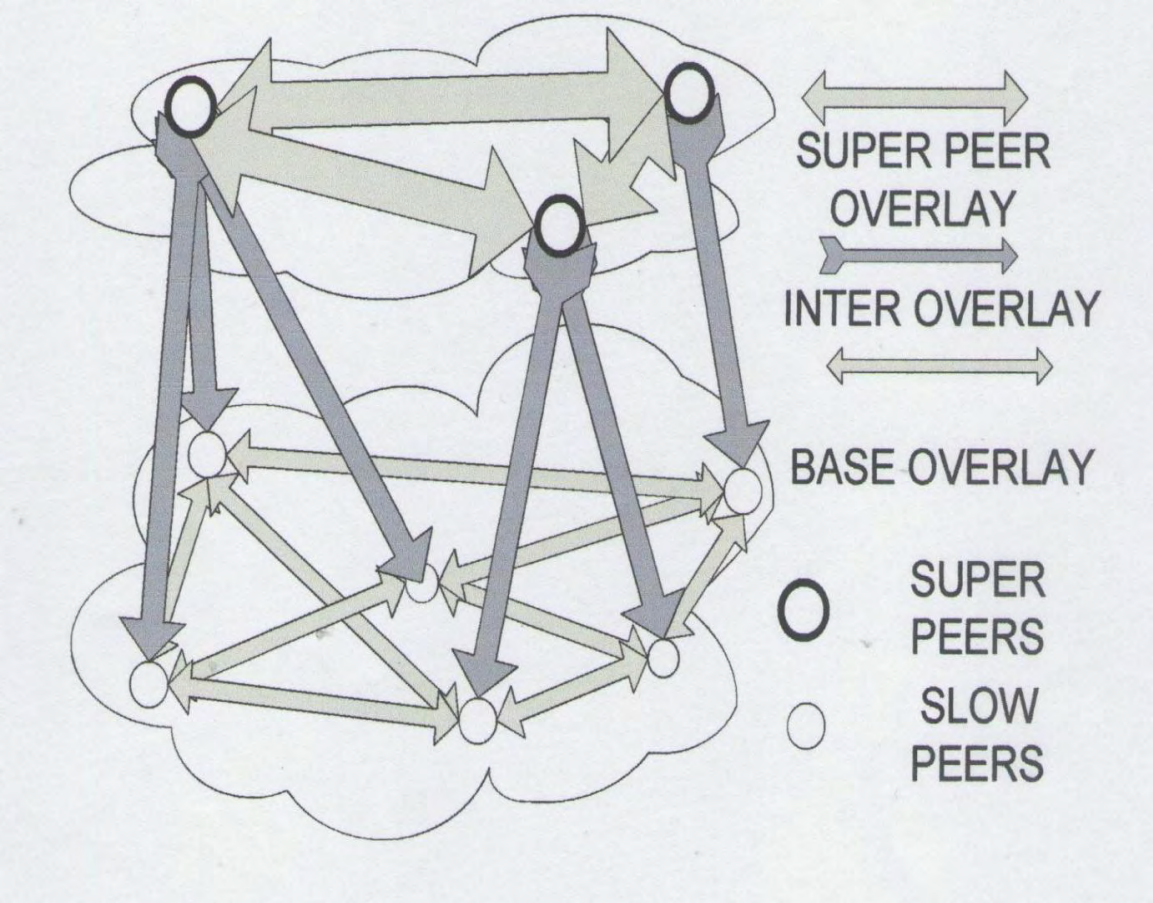
Αυτό πρέπει να γίνεται, ανεξάρτητα από τις όποιες καθυστερήσεις μετάδοσης υπάρχουν στις συνδέσεις μεταξύ :

super peer -- super peer (super peer overlay)

super peer -- slow peer (inter overlay)

slow peer –slow peer (base overlay)

και ανεξάρτητα από τη δυναμική εισαγωγή και αποχώρηση κόμβων ,στο σύστημα.



ΣΧΗΜΑ 4.1 Αναπαράσταση διασυνδέσεων μεταξύ κομβων σε ιεραρχικό δίκτυο

ΚΕΦΑΛΑΙΟ 5

ΓΕΝΙΚΕΣ ΑΡΧΕΣ ΔΙΑΦΟΡΩΝ ΣΥΣΤΗΜΑΤΩΝ(P2P)

5.1 Unstructured P2P Systems

5.1.1 GNUTELLA

Ουσιαστικά σχηματίζεται ένα είδος «εικονικού» δικτύου, που αποτελείται από τους συνδεόμενους μεταξύ τους peers, χωρίς να υπάρχει κεντρικός εξυπηρετητής, δημιουργώντας έτσι μια πιο «καθαρή» peer to-peer υποδομή. Δύο κόμβοι οι οποίοι διατηρούν μια ανοιχτή σύνδεση μεταξύ τους ονομάζονται γείτονες(neighbors). Ο αριθμός των γειτόνων που έχει ένας κόμβος ονομάζεται *outdegree*.

Τα μηνύματα δρομολογούνται μόνο διαμέσου των ανοιχτών συνδέσεων. Εάν ένα μήνυμα χρειάζεται να ταξιδέψει μεταξύ δύο κόμβων που δεν είναι γείτονες, τότε θα πρέπει να ταξιδέψει πάνω από πολλαπλές συνδέσεις .Το μήκος του μονοπατιού που διασχίζει ένα μήνυμα είναι γνωστό ως αριθμός από *hops*.

Ο κάθε peer είναι ελεύθερος να μπει ή να βγει από το δίκτυο των peers οποιαδήποτε στιγμή το επιθυμεί. Η είσοδος ενός peer στο δίκτυο προϋποθέτει τη γνώση ενός άλλου κόμβου που είναι ήδη στο δίκτυο. Για να διατηρείται όμως η ύπαρξη του δικτύου, το πρωτόκολλο που υποστηρίζει το Gnutella περιλαμβάνει και μηνύματα ping και pong. Τα πρώτα χρησιμοποιούνται για την ανεύρεση νέων peers και η εκτέλεση τους από έναν peer γίνεται σε τακτά χρονικά διαστήματα ή κατά την πρώτη εισαγωγή του στο δίκτυο. Τα pong μηνύματα αντίστοιχα είναι οι απαντήσεις στα μηνύματα ping που λαμβάνονται.Ενίοτε οι peers, που συμμετέχουν στο δίκτυο, ανοίγουν και νέες συνδέσεις με peers τους οποίους έχουν προηγουμένως εντοπίσει από την ανταλλαγή ping/pong μηνυμάτων.

Στο Gnutella είναι δυνατόν να υπάρχουν διαφορετικά, μη συνδεδεμένα υποδίκτυα από peers. Δεν υπάρχει δηλαδή κάποιος μηχανισμός που να εξασφαλίζει ότι τελικά όλα τα μέλη μέσω κάποιου μονοπατιού θα είναι συνδεδεμένα.

Αναλυτικότερα κάθε μέλος δημιουργεί μία από σημείο σε σημείο σύνδεση με ένα υποσύνολο από κάποια άλλα ήδη ενεργά μέλη του συστήματος (γείτονες peers).

Κάθε αίτηση αναζήτησης που λαμβάνει ένας peer την προωθεί με τη σειρά του στους γείτονες του κ.ο.κ. Έτσι δημιουργείται μια «πλημμύρα» από προωθούμενες αιτήσεις, η οποία ελέγχεται και τερματίζει, χάρη σε ένα πεδίο που περιέχεται σε κάθε αίτηση και είναι αντίστοιχο με το time-to-live (TTL) πεδίο που περιέχεται στα IP πακέτα. Αν στο μεταξύ υπάρξει και επιτυχία στην αναζήτηση αυτή επιστρέφει στον αρχικό αιτούντα μέσω του μονοπατιού, που έφτασε στον παραλήπτη, ενώ ταυτόχρονα συνεχίζεται και η προώθησή του. Επομένως η τεχνική αναζήτησης η οποία χρησιμοποιείται στο Gnutella είναι η διάσχιση κατά πλάτος πρώτα του δικτύου (breadth-first traversal-BFS) με όριο βάθους *D.* *(* μέγιστο time-to-live του μηνύματος και το οποίο μετριέται σε hops.)

Για να αποφευχθεί η επανάληψη αποστολής ενός προηγούμενου μηνύματος από ένα κόμβο του Gnutella, ανατίθενται στα μηνύματα μοναδικά αναγνωριστικά. Κάθε φορά που ένα μήνυμα διασχίζει το δίκτυο, το αναγνωριστικό του μηνύματος αποθηκεύεται από τους κόμβους από τους οποίους περνάει. Εάν υπάρχουν βρόγχοι στο δίκτυο τότε είναι πιθανόν ένας κόμβος να λάβει το ίδιο μήνυμα δύο φορές. Τότε το μήνυμα δε μεταδίδεται ξανά, καθώς το αναγνωριστικό του ήδη προϋπάρχει στον κόμβο.

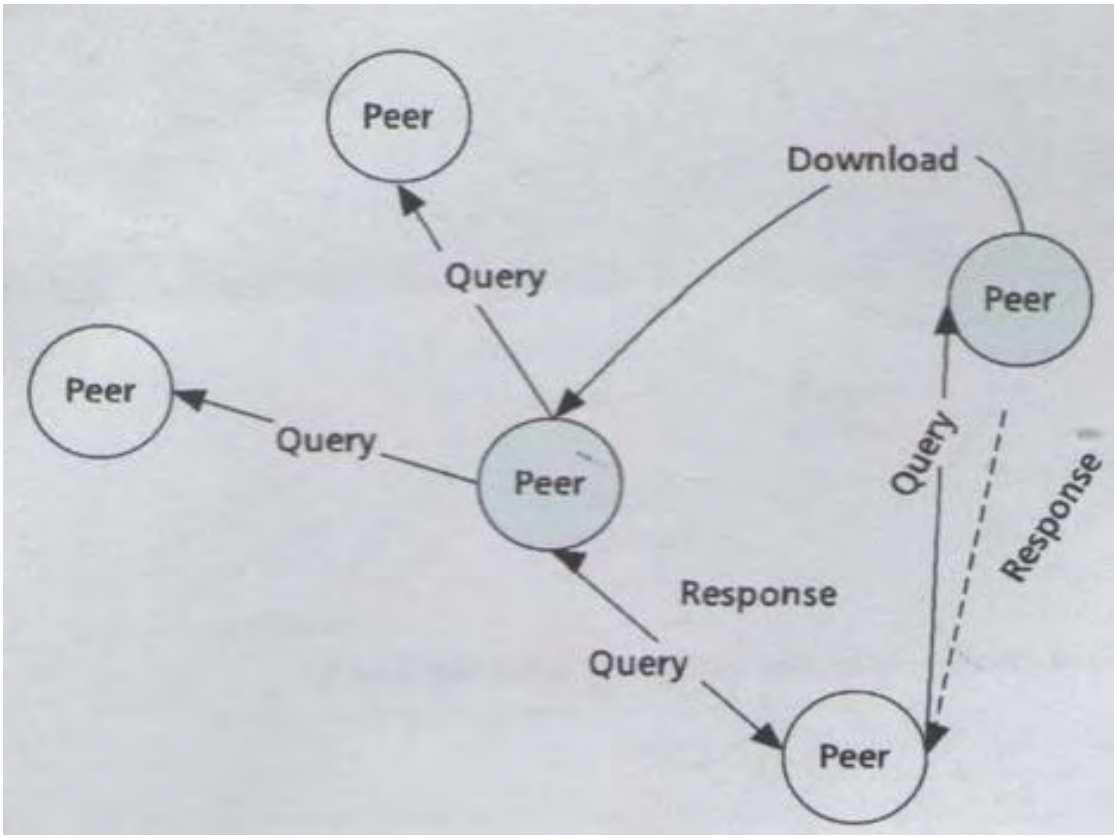
Στο Gnutella η τοποθεσία της πηγής στην οποία βρίσκονται τα δεδομένα, δεν είναι γνωστή στον αποκρινόμενο κόμβο. Στην περίπτωση αυτή, το μήνυμα απάντησης θα δρομολογηθεί πίσω κατά μήκος του αντίστροφου μονοπατιού που διέσχισε το μήνυμα ερώτησης. Εάν όμως η τοποθεσία της πηγής είναι γνωστή, τότε μπορεί να δημιουργηθεί μια προσωρινή σύνδεση και να μεταφερθούν τα δεδομένα απευθείας.

Ενώ η πρώτη μέθοδος χρησιμοποιεί περισσότερο bandwith από τη δεύτερη, αποφεύγει το «βομβαρδισμό» της πηγής με αιτήσεις σύνδεσης όπως κάνει η δεύτερη μέθοδος και παρέχει επίσης επιπρόσθετα οφέλη όπως είναι η ανωνυμία.

Το Gnutella εμφανίζει πολύ καλή προσαρμοστικότητα στην αλλαγή του πληθυσμού των κόμβων. Επιπλέον, υπάρχει χαμηλό κόστος στη δρομολόγηση της πληροφορίας. Ένα άλλο θετικό

στοιχείο που παρουσιάζεται είναι η μη ύπαρξη περιορισμού στην αποθήκευση, καθώς κάθε peer αποθηκεύει κλειδιά των αρχείων του.

Το πρόβλημα όμως που εμφανίζεται είναι απόδοση της αναζήτησης αφού ο αριθμός των μηνυμάτων που πρέπει να ανταλλαχθούν είναι πολύ μεγάλος. Επιπρόσθετα σημαντικό είναι και το πρόβλημα του *free riding*, καθώς πολύ peers είτε καταναλώνουν πόρους χωρίς να παρέχουν τίποτα, είτε προσφέρουν δεδομένα που δεν έχουν κανένα ενδιαφέρον. Μια άλλη πηγή αναποτελεσματικότητας οφείλεται σε peers που είναι περιορισμένων δυνατοτήτων, καθώς μπορεί να συνδέονται με συνδέσμους χαμηλής χωρητικότητας στο δίκτυο, και επιφέρουν την τμηματοποίηση του δικτύου κατά την αποχώρησή τους. Οι peers περιορισμένων δυνατοτήτων «πεθαίνουν» εξαιτίας των ίσων ρόλων και ευθυνών που τους ανατίθενται ανεξαρτήτως των δυνατοτήτων τους. Όμως υπάρχει σημαντική ετερογένεια ανάμεσα στις δυνατότητες των συμμετεχόντων peers.



ΣΧΗΜΑ 5.1 **Το δίκτυο Gnutella**

5.1.2 Freenet

Το Freenet όπως και το Gnutella είναι κατανεμημένο. Το Freenet μπορεί να περιγραφεί σαν ένα δίκτυο διαμοιρασμού bandwith και χώρου στους δίσκους, ενώ το Gnutella είναι ένα δίκτυο αναζήτησης και ανακάλυψης το οποίο προάγει την ελεύθερη μετάφραση και απάντηση στις ερωτήσεις.

Το Freenet είναι ένα προσαρμόσιμο peer-topeer σύστημα το οποίο υποστηρίζει την έκδοση, αντιγραφή, και ανάκτηση των δεδομένων. Επιπλέον διαφοροποιείται από το Gnutella στα παρακάτω θέματα:
1. Έχει έναν αποδοτικό μηχανισμό αναζήτησης, που ελέγχει τον αριθμό των μηνυμάτων που δημιουργούνται.
2. Έχει ένα μηχανισμό αντιγραφής για τη διασπορά των δεδομένων στο δίκτυο και με τον τρόπο αυτό αυξάνεται η απόδοση του συστήματος.
3. Παρέχει ανωνυμία. Οι peers δε γνωρίζουν τι αποθηκεύουν καθώς τα δεδομένα είναι κρυπτογραφημένα.

Η τεχνική αναζήτησης που χρησιμοποιείται είναι η αναζήτηση σε βάθος πρώτα (depth-first traversal-DFS) με όριο βάθους *D*. Κάθε κόμβος προωθεί την ερώτηση σε ένα μόνο γείτονα και περιμένει για μια οριστική απάντηση από αυτόν πριν προωθήσει την ερώτηση σε άλλο γείτονα, ή προωθεί τα αποτελέσματα πίσω στην πηγή της ερώτησης. Με την DFS τεχνική, επειδή κάθε κόμβος επεξεργάζεται την ερώτηση σειριακά, η αναζήτηση μπορεί να τερματιστεί όταν η ερώτηση θα έχει ικανοποιηθεί ελαχιστοποιώντας έτσι το κόστος, σε ότι αφορά το πλήθος των μηνυμάτων.

Οι peers στο Freenet διατηρούν επίσης ένα πίνακα δρομολόγησης για τους γειτονικούς peers. Αυτοί όμως αποθηκεύουν επιπρόσθετα το κλειδί των δεδομένων καθώς και τα αντίστοιχα δεδομένα. Έτσι όταν φθάνει μια αίτηση αναζήτησης σε ένα peer, είτε θα μπορεί να απαντηθεί απευθείας από τα δεδομένα που βρίσκονται στον πίνακα, ή θα προωθήσει την αίτηση σε κάποιον άλλο peer. Αυτό γίνεται επιλέγοντας εκείνον τον peer που έχει το πιο παρόμοιο κλειδί. Όταν φθάσει μια απάντηση, ο peer την αποθηκεύει στην δική του πηγή δεδομένων. Εάν η πηγή είναι ήδη γεμάτη τότε θα πρέπει να διαγράψει κάποια άλλα αποθηκευμένα δεδομένα.

Η στρατηγική που ακολουθείται στο σημείο αυτό είναι η διαγραφή των λιγότερα πρόσφατα χρησιμοποιούμενων (LRU) δεδομένων. Επομένως ύστερα από ένα σημείο ο κόμβος δεν αποθηκεύει τα δεδομένα που σχετίζονται με ένα κλειδί, αλλά μόνο τη διεύθυνση. Τα έγγραφα που έχουν μεγαλύτερη ζήτηση μετακινούνται προς τα πάνω αντικαθιστώντας τα λιγότερα ζητούμενα.

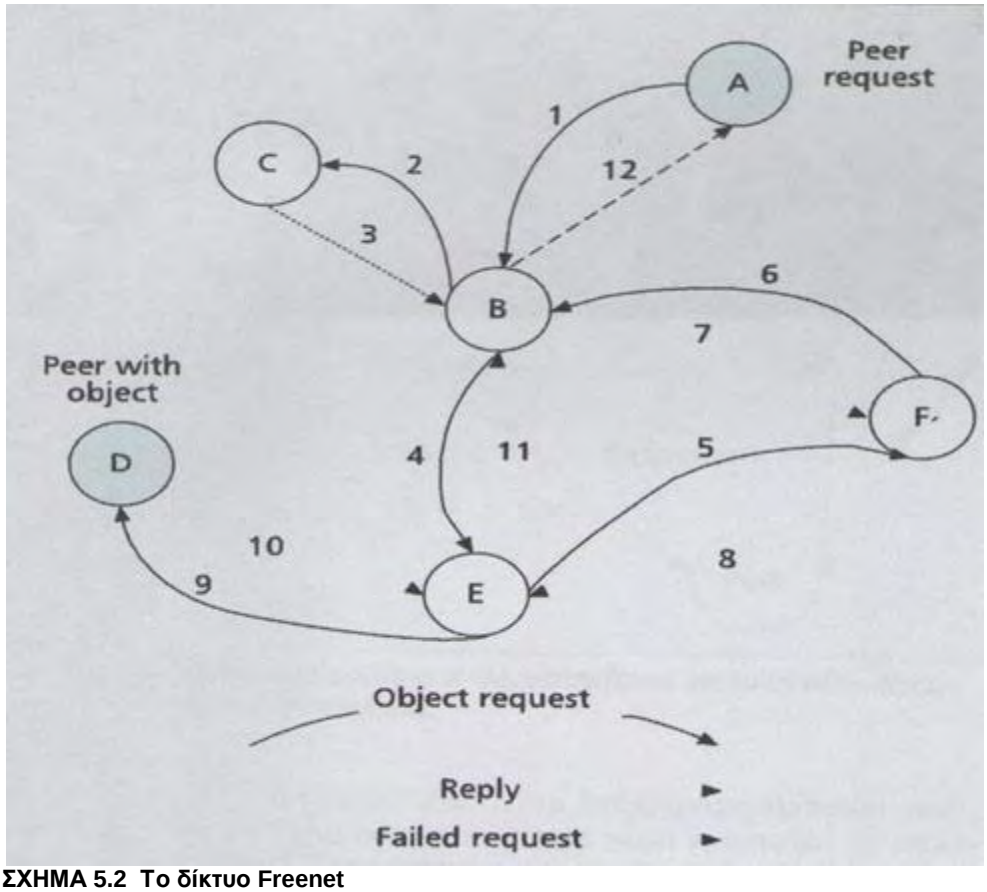
Η απόσταση προς τα πάνω ,που μετακινούνται τα έγγραφα συνδέεται με το μέγεθός τους, έτσι το μεγάλο μέγεθος ενός εγγράφου αποτελεί ένα μειονέκτημα. Συνεπώς, το Freenet προτρέπει τους χρήστες να μη σπαταλούν χώρο στο δίσκο και για το λόγο αυτό να συμπιέζουν τα έγγραφα πριν τα εισάγουν.

Η εισαγωγή των δεδομένων στο Freenet είναι παρόμοια με την αναζήτηση. Για να εισαχθεί ένα καινούργιο έγγραφο στο σύστημα θα πρέπει πρώτα να υπολογιστεί ένα κλειδί για το αρχείο που να μην υπάρχει ήδη. Στη συνέχεια αποστέλλεται ένα μήνυμα εισαγωγής που περιέχει το κλειδί με μία hops-to-live τιμή στους γειτονικούς κόμβους. Κάθε peer που λαμβάνει το μήνυμα ελέγχει εάν έχει ήδη το κλειδί στην τοπική cache που διατηρεί. Εάν το κλειδί βρεθεί, τότε επιστρέφεται το αποθηκευμένο αρχείο, διαφορετικά η δρομολόγηση συνεχίζεται μέχρι η τιμή hops-to-live να γίνει μηδέν. Εάν έχει γίνει η τιμή μηδέν και δεν έχει ανακαλυφθεί το ίδιο κλειδί, το έγγραφο αποστέλλεται κατά μήκος του ίδιου μονοπατιού και κάθε κόμβος που λαμβάνει το μήνυμα το αποθηκεύει.

Όπως συμβαίνει στο Gnutella έτσι και στο Freenet οι αιτήσεις αναζήτησης έχουν ένα χρονικό όριο ζωής (TTL). Τα μηνύματα επίσης χαρακτηρίζονται από μοναδικά αναγνωριστικά ώστε να αποφεύγονται οι επαναμεταδόσεις μηνυμάτων. Επιπλέον, ταύτιση των δύο συστημάτων υπάρχει και στο γεγονός ότι για να εισέλθει ένας κόμβος στο δίκτυο θα πρέπει να γνωρίζει κάποιον αρχικό κόμβο.

Το κύριο πλεονέκτημα της προσέγγισης του Freenet προέρχεται από την στρατηγική δρομολόγησης που είναι κατά βάθος πρώτα, και η ποιότητα της δρομολόγησης θα πρέπει συνεχώς να βελτιώνεται καθώς η στρατηγική caching σχεδιάστηκε με τρόπο που να τείνει να ομαδοποιήσει παρόμοια δεδομένα σε κόμβους του δικτύου και επομένως οι κόμβοι γίνονται συνεχώς περισσότερο ειδικευμένοι.

Το μειονέκτημα του Freenet έναντι του Gnutella, είναι ότι επιτρέπει μόνο την ακριβή ταύτιση των κλειδιών στην αναζήτηση ενός εγγράφου και συνεπώς δε γίνεται καθόλου αναζήτηση βάση περιεχομένου. Αυτό πρακτικά σημαίνει ότι για να βρει ένας χρήστης το έγγραφο που ζητάει θα πρέπει να ζητήσει είτε αρχεία τα οποία έχει ο ίδιος αποθηκευμένα, είτε άλλα αρχεία για τα οποία έχει μάθει το όνομά τους.



ΣΧΗΜΑ 5.2 Το δίκτυο Freenet

5.1.3 Bit Torrent

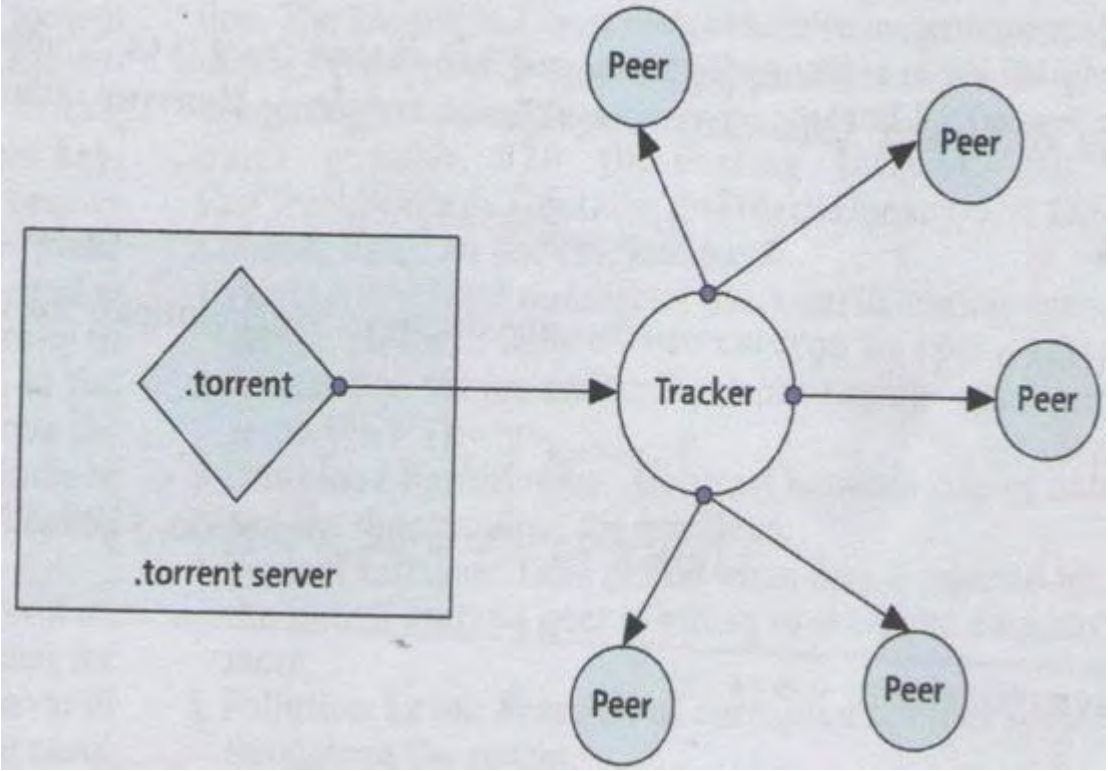
Το BitTorrent είναι ένα πρωτόκολλο σχεδιασμένο για μοίρασμα μεγάλων αρχείων σε κομμάτια χρησιμοποιώντας ένα αμοιβαίο μοίρασμα των κομματιών μεταξύ των peers (swarm). Η χρήση του είναι δωρεάν και είναι δομημένο δίκτυο. Δεν χρησιμοποιεί μηχανή αναζήτησης, την δουλεία αυτή την κάνουν κάποιες ιστοσελίδες όπως supernova.org ή btjunkie.org.

Στην μεταφορά των δεδομένων χρησιμοποιεί την τεχνική pipeline προκειμένου να έχει καλύτερα αποτελέσματα
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,δηλαδή :

- 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)
- Όμως, είναι κεντροποιημένο δίκτυο άρα υπάρχει περισσότερος έλεγχος και έτσι γίνεται δυσκολότερο να υπάρξουν ψεύτικες IP διευθύνσεις ή αριθμοί διόδου.



ΣΧΗΜΑ 5.3 Το δίκτυο Bit Torrent

5.1.4 Overnet/eDonkey

Είναι ένα υβριδικό, δύο επιπέδων P2P δίκτυο αποθήκευσης πληροφοριών ,που αποτελείται από client και server, και χρησιμοποιείται για τη δημοσίευση και την ανάκτηση μικρών κομματιών δεδομένων, δημιουργώντας ένα file-sharing δίκτυο.

Αυτή η αρχιτεκτονική παρέχει δυνατότητες όπως παράλληλη λήψη ενός αρχείου από πολλούς peers, ανίχνευση της φθοράς αρχείου χρησιμοποιώντας hashing, μερική ανταλλαγή αρχείων κατά τη διάρκεια της λήψης, και expressive querying methods για αναζήτηση αρχείων.

Για να ενταχθεί στο δίκτυο, ο peer (client) πρέπει να γνωρίζει τη διεύθυνση IP και τη θύρα ενός άλλου από peer(server) στο δίκτυο. Έπειτα εκκινείται μέσω του αλλου peer. Οι πελάτες συνδέονται σε ένα διακομιστή και να καταχωρήσουν object files που μοιράζονται παρέχοντας μετα-δεδομένα που περιγράφουν τα object files.

Μετά την εγγραφή, οι clients μπορούν είτε να κάνουν αναζήτηση (by querying the meta-data)ή να ζητήσουν ένα συγκεκριμένο αρχείο ,μέσω του μοναδικού τους αναγνωριστικού δικτύου.

Οι servers παρέχουν τις θέσεις των object files όταν ζητείται από τους πελάτες, έτσι ώστε οι πελάτες μπορούν να κατεβάσουν τα αρχεία απευθείας από τις αναφερόμενες περιοχές.

5.1.5 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. Μετά την σύνδεση όλες οι επικοινωνίες γίνονται χωρίς να επικοινωνούν με τον κεντρικό server.

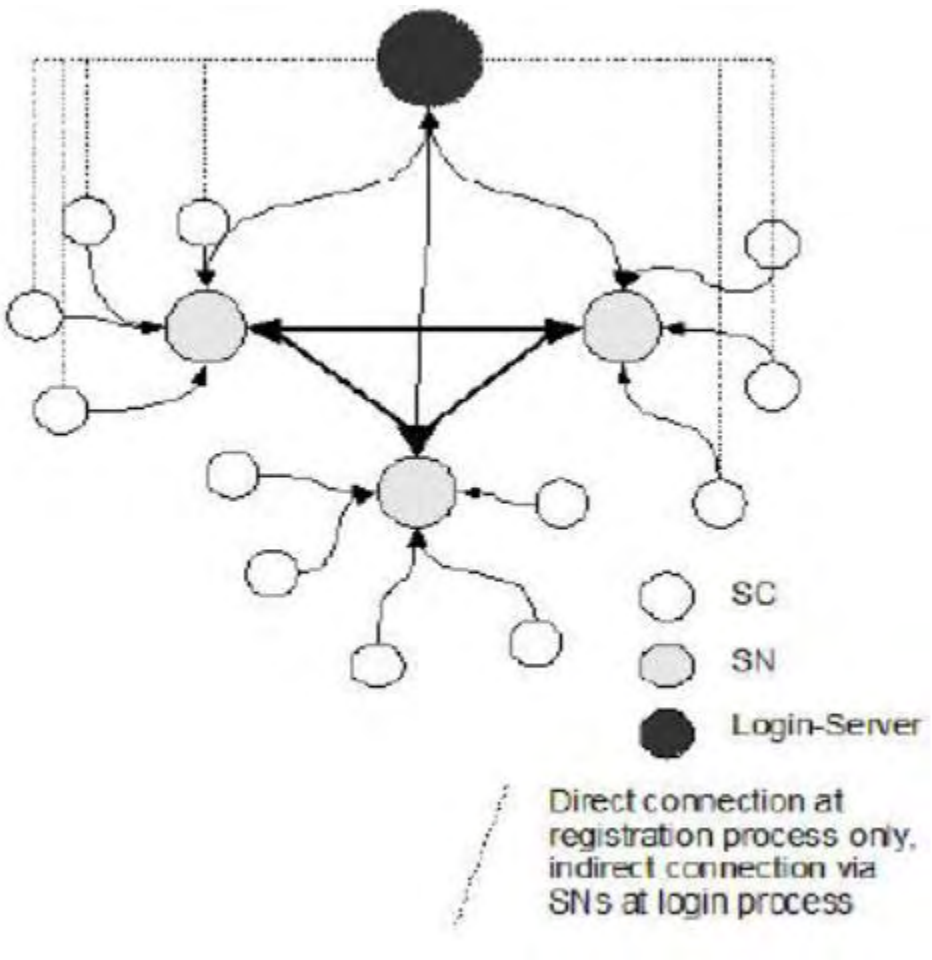
Αυτό που γίνεται είναι κάποιοι κόμβοι αρκετά δυνατοί να προωθούνται σε μεγάλους κόμβους που δρουν σαν δρομολογητές μεταξύ των απλών χρηστών. Για να είναι λειτουργικό αυτό το σύστημα, πρέπει οι μεγάλοι κόμβοι να γνωρίζουν την πλειοψηφία των υπόλοιπων.

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 . 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.

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 .



ΣΧΗΜΑ 5.4 Παράδειγμα δομής του δικτύου Skype

5.1.6 Joost

Οι δημιουργοί του KaZaA και του Skype δημιούργησαν και το Joost. Χρησιμοποιεί τεχνολογίες προβολής video και προσπαθεί να ενώσει την τηλεόραση με το διαδίκτυο με νόμιμα μέσα. Είναι μία υπηρεσία Internet protocol-based television (IPTV).
Τα προτερήματα του Joost:

- Αποτρέπει την εικόνα να παγώνει
- Κατηγοριοποιεί τα περιεχόμενα
- Ελεύθερα προσβάσιμο
- Επαγγελματική εμφάνιση
- Μηχανή αναζήτησης
- Συμβατό με iPod και iPhone
- Έχει υπογράψει συμβάσεις με αρκετές εταιρίες προκειμένου να διανέμονται τα media νόμιμα.
- Χρησιμοποιείται για να είναι ορατό το περιεχόμενο και να αλληλεπιδρά με άλλους χρήστες ταυτόχρονα
- Βλέπεις τι βλέπουν οι φίλοι σου
- Μπορούν να δημιουργηθούν ομάδες βάση των ενδιαφερόντων.

Χρησιμοποιεί κρυπτογράφηση σε επίπεδο byte. Δεν εγγυάται ασφάλεια για τα προσωπικά δεδομένα του χρήστη, αφενός γιατί μοιράζεται τα δεδομένα του για στατιστικούς λόγους ή διαφήμιση και αφετέρου γιατί υπάρχει φόβος πως θα μπορούσε κάποιος κακόβουλος να υποκλέψει τα δεδομένα του χρήστη, αλλά και τις επιθέσεις ιών και trojans. Όμως η επικοινωνία μεταξύ Joost servers και peers γίνεται μέσω κρυπτογραφημένων μηνυμάτων, οπότε και τα δεδομένα αυτά είναι δύσκολο να προσβληθούν από κάποιον τρίτο.

Εκτός από το θέμα της ασφάλειας, το joost έχει το μειονέκτημα ότι αν και φορτώνει πολύ αποδοτικά ισορροπημένες υπολογιστικές πηγές, το πετυχαίνει έχοντας μεγάλες επιπτώσεις στην αποδοτικότητα του δικτύου. Ακόμα έχει μικρή προσαρμογή στο ρυθμό πρόσβασης.

Το Joost, αρχικά, ήταν ένα κεντροποιημένο δίκτυο. Όμως όσο αυξάνονταν οι χρήστες του γινόταν περισσότερο αποκεντρωμένο, με πιο διανεμημένη αρχιτεκτονική. Χρησιμοποιεί τόσο απλούς κόμβους όσο και μεγάλους κόμβους. Τώρα είναι σε ένα υβριδικό στάδιο. Όσον αφορά το πρωτόκολλο ,χρησιμοποιεί UDP ή TCP ανάλογα με την περίσταση (UDP για βίντεο, TCP για έλεγχο) .

5.1.7 SopCast

Το SopCast είναι μία ελεύθερη P2PTV εφαρμογή και έχει πάρα πολλές ομοιότητες όσον αφορά την δομή του με το Joost. Τρέχει προγράμματα που προβάλλονται στην τηλεόραση με ταχύτητα αναμετάδοσης συνήθως από 250 έως 400 Kbps που όμως κάποιες φορές φτάνει και στα 800Kbps. Ακόμα μέσω του SopCast κάποιος ,μπορεί να δημιουργήσει το δικό του κανάλι.

Χρησιμοποιεί κατά συντριπτική πλειοψηφία το πρωτόκολλο UDP, καθώς χρειάζεται ταχύτητα στις προβολές, και ελάχιστα το TCP. Από τις ζωντανές μεταδόσεις είναι κατά μισό λεπτό με σαράντα πέντε δευτερόλεπτα περίπου πιο αργό.

Είναι υβριδικό δίκτυο, δηλαδή χρησιμοποιεί τόσο μεγάλους όσο και κανονικούς κόμβους, και δεν λαμβάνει υπ' όψιν του τη γεωγραφική τοποθεσία των peers κατά την διάρκεια της αναμετάδοσης. Ο μέσος ρυθμός λήψης βίντεο είναι σχεδόν ίδιος σε κάθε peer. Η ταχύτητα ανεβάσματος είναι συνήθως σταθερή για έναν peer, όμως ποικίλει μεταξύ διαφορετικών peers.

Τα μειονεκτήματά του είναι ότι υποφέρει συχνά από καθυστερήσεις των peers, γιατί κάποιοι peers που παρακολουθούν το ίδιο κανάλι μπορεί να μην είναι συγχρονισμένοι ,και το βίντεο και ο ήχος ίσως να μην είναι συγχρονισμένα. Επίσης, στα μειονεκτήματα μπορεί να προστεθεί η καθυστέρηση της εναλλαγής των καναλιών που είναι συνήθως γύρω στα πενήντα δευτερόλεπτα. Τέλος, το συνολικό ποσοστό χασίματος πακέτων είναι υψηλό.

Όσον αφορά την ασφάλεια στο SopCast, για τον χρήστη που θέλει να δει κάτι δεν τίθεται θέμα ασφάλειας αφού δεν δίνει κάποιο στοιχείο του. Τα δεδομένα όμως αν και προστατεύονται από τους ελέγχους των μεγάλων κόμβων, κρυπτογραφώντας τα μηνύματα, είναι ορισμένες φορές τρωτά στους κακόβουλους χρήστες, όπως φυσικά και όλα τα άλλα P2P συστήματα. Ακόμα το SopCast χρησιμοποιεί ασφάλεια End-to-End, δηλαδή η σύνδεση τερματίζεται αν κριθεί ότι κάτι δεν είναι ασφαλές.

5.1.8 ΣΥΓΚΡΙΣΗ Unstructured P2P Συστημάτων

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.

ΣΧΗΜΑ 5.5

5.2 Structured P2P Systems

5.2.1 Το σύστημα Chord

Το Chord είναι ένα κατανεμημένο πρωτόκολλο για τον εντοπισμό δεδομένων σε ένα peer-to-peer σύστημα που αναπτύχθηκε από ομάδα του MIT και παρουσιάσθηκε το 2001(Sigcomm conference).

Βασίζεται στη λειτουργία: δεδομένου ενός κλειδιού, αυτό αντιστοιχίζεται σε έναν κόμβο, όπου αποθηκεύεται το ζεύγος κλειδί/τιμή. Πρόκειται για ένα σύστημα απλό στην κατασκευή του, ανεκτικό στις αλλαγές του peer-to-peer δικτύου και εγγυάται την εύρεση των δεδομένων σε χρόνο O(logN) όπου N το πλήθος των κόμβων στο δίκτυο.

Το Chord μπορεί να λειτουργήσει σε ένα δυναμικό περιβάλλον όπου οι κόμβοι εισέρχονται και αποχωρούν από το σύστημα αυθαίρετα με την απαίτηση όμως κάθε κόμβος να αποθηκεύει τμήμα της πληροφορία για επιτυχή δρομολόγηση.

Τόσο οι κόμβοι όσο και τα δεδομένα, που έχουν μοναδικό m-bit (160 bits) αναγνωριστικό (ID) οργανώνονται σε έναν εικονικό δακτύλιο. Το ID του κόμβου κατακερματίζεται από την IP του, και το ID του αντικειμένου (δεδομένο) κατακερματίζεται από το όνομά του και έτσι προκύπτει η θέση τους πάνω στο δακτύλιο (δακτύλιος των 0 έως 2^m-1 θέσεων).

Οι κόμβοι κατέχουν πληροφορία για τον προηγούμενο και τον επόμενο κόμβο στο δακτύλιο. Ο κάθε κόμβος είναι υπεύθυνος για τα αντικείμενα που είναι μεταξύ του προηγούμενου κόμβου και του ίδιου. Οι βασικές λειτουργίες που μπορούν να πραγματοποιηθούν σε ένα τέτοιο σύστημα είναι η εισαγωγή νέου κόμβου (join), η αποθήκευση και ανάκτηση δεδομένου (store & retrieve) και η αποχώρηση ενός κόμβου (leave).

Εισαγωγή κόμβου (Join**):** Όταν ένας κόμβος n επιθυμεί να εισέλθει στο σύστημα κατακερματίζει την IP διεύθυνσή του για να προκύψει το αναγνωριστικό του και με κάποια διαδικασία (εξωτερικό μηχανισμό) μαθαίνει το αναγνωριστικό ενός κόμβου n’ που ήδη ανήκει στο Chord. Ο νέος κόμβος χρησιμοποιεί τον n’ κόμβο για να

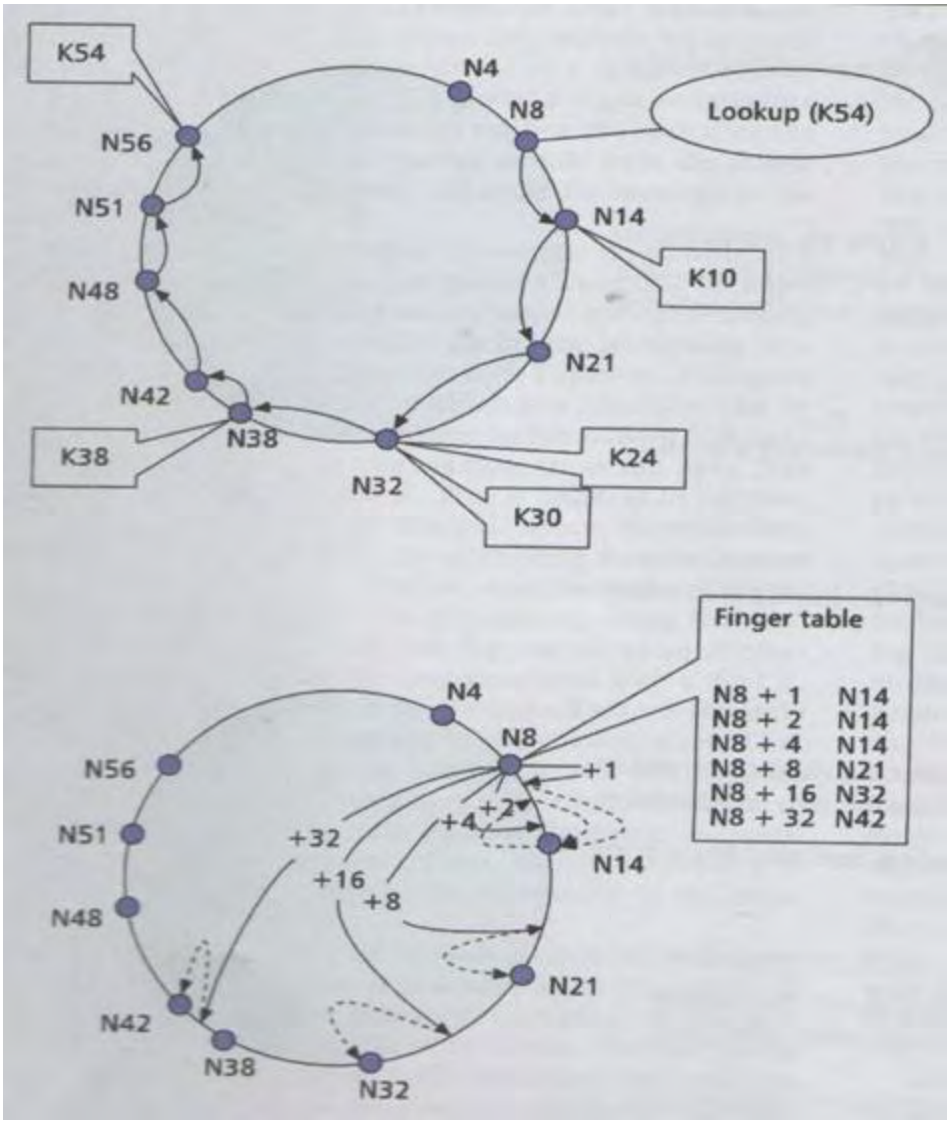
προσθέσει τον εαυτό του στο δίκτυο Chord και να αρχικοποιήσει την κατάσταση του που περιλαμβάνει και την μεταφορά κλειδιών από τον επόμενό του κόμβο, αν αυτά αντιστοιχίζονται τώρα σε αυτόν (ο κόμβος n είναι τώρα ο επόμενος τους). Η τελευταία διαδικασία απαιτεί το πολύ O(1/N) μετακινήσεις κλειδιών.

Αποχώρηση κόμβου (leave): Όταν ένας κόμβος αποχωρεί από το σύστημα, τότε τα κλειδιά που είχε στην ευθύνη του αντιστοιχίζονται στον επόμενό του κόμβο. Η διαδικασία της ενημέρωσης καθώς οι κόμβοι έρχονται και φεύγουν από το σύστημα απαιτεί $O(\log^2 N)$ μηνύματα.

Εισαγωγή Δεδομένων: Ο κόμβος που επιθυμεί να αποθηκεύσει ένα αντικείμενο στο σύστημα εφαρμόζει μια συνάρτηση κατακερματισμού στο όνομα του αντικείμένου και προκύπτει το αναγνωριστικό του. Το νέο αντικείμενο αντιστοιχίζεται (και αποθηκεύεται) στον κόμβο που έχει ID ίσο με το ID του, αν υπάρχει ή του αμέσως επόμενου αν δεν υπάρχει.

Για την **ανάκτηση** του αντικειμένου χρησιμοποιείται παρόμοια διαδικασία. Εφαρμόζεται η συνάρτηση κατακερματισμού και προκύπτει η θέση του αντικειμένου και δρομολογείται η ανάκτησή του.

Η μόνη πληροφορία που είναι απαραίτητη στο σύστημα Chord για επιτυχή δρομολόγηση είναι η γνώση του επομένου. Όμως ένα τέτοιο σχήμα δεν διατηρεί την ιδιότητα της κλιμάκωσης. Για το λόγο αυτό κάθε κόμβος διατηρεί ένα τμήμα πληροφορίας για τους άλλους κόμβους βελτιώνοντας έτσι την απόδοση του αλγορίθμου. Η πληροφορία που διατηρεί ο κάθε κόμβος είναι ένας πίνακας m εγγραφών, finger table, και περιέχει τους κόμβους που βρίσκονται σε απόσταση $2^0, 2^1, 2^2, , \ldots, 2^{m-1}$ από αυτόν. Οι ερωτήσεις τώρα δρομολογούνται μέσω του finger table και η αναζήτηση ολοκληρώνεται σε O(logN) το πολύ hops.



ΣΧΗΜΑ 5.6 Το δίκτυο Chord

5.2.2 Το σύστημα CAN

Το σύστημα CAN (Content Addressable Network) είναι ένα ακόμη δομημένο σύστημα που ο σχηματισμός του overlay δικτύου βασίζεται σε κατανεμημένους πίνακες κατακερματισμού. Το CAN αναπτύχθηκε από ομάδα του πανεπιστημίου του Berkeley και παρουσιάσθηκε το 2001.

Η σχεδίασή του είναι επίσης απλή και κατανοητή. Βασίζεται στον εικονικό καρτεσιανό χώρο d-διαστάσεων. Για κάθε διάσταση υπάρχει μια συνάρτηση κατακερματισμού. Ο χώρος διαμερίζεται και αντιστοιχίζεται στους κόμβους που μετέχουν στο σύστημα. Το τμήμα του χώρου που κατέχει ένας κόμβος καλείται zone. Τα αντικείμενα (δεδομένα) αντιστοιχίζονται σε σημεία στο χώρο από μια συνάρτηση κατακερματισμού και αποθηκεύονται στους κόμβους που κατέχουν το zone που ανήκουν τα σημεία, ως ζεύγη κλειδί/τιμή (K, V). Οι κόμβοι αποθηκεύουν τμήμα του κατανεμημένου πίνακα κατακερματισμού και πληροφορία για τους άμεσα γείτονές τους. Δύο κόμβοι θεωρούνται γείτονες αν οι d-1 διαστάσεις τους εκτεινόμενες συμπίπτουν και εφάπτονται στην άλλη διάσταση.

Δρομολόγηση: Ένα CAN μήνυμα περιλαμβάνει τις συντεταγμένες προορισμού. Έτσι δρομολογείται από έναν “greedy” αλγόριθμο προς τον κόμβο που βρίσκεται πιο κοντά στον προορισμό του. Το μέσο μήκος μονοπατιού για ένα χώρο d διαστάσεων με n ίσες zones είναι $(d/4)(n^{1/d})$ hops, η δε πληροφορία για τους γείτονες που διατηρεί κάθε κόμβος είναι 2d.

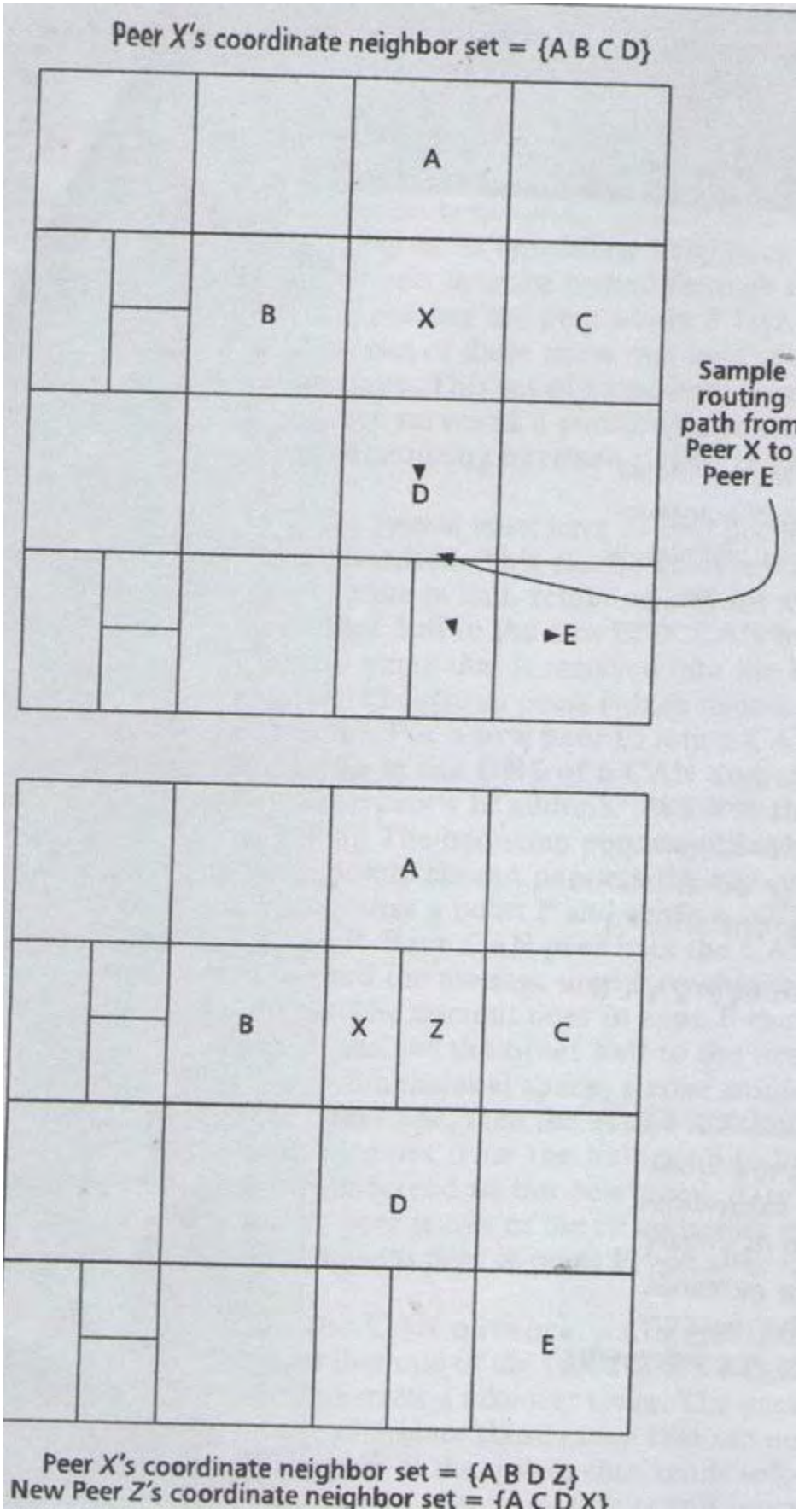
Εισαγωγή κόμβου (Join): Η σύνδεση ενός νέου κόμβου γίνεται μέσω της διαδικασίας bootstrap. Ο νεοεισερχόμενος κόμβος εντοπίζει έναν κόμβο που ανήκει ήδη στο σύστημα CAN. Με την χρήση του μηχανισμού δρομολόγησης, επιλέγει ένα τυχαίο σημείο P στο χώρο και στέλνει ένα μήνυμα συνένωσης στον κόμβο που καλύπτει το zone όπου ανήκει το P και ο οποίος μπορεί να διαμεριστεί. Ο νέος κόμβος αναλαμβάνει το μισό zone ενώ το άλλο μισό το αναλαμβάνει ο προκάτοχος του. Οι γείτονες κόμβοι ενημερώνονται έτσι ώστε να συμπεριλάβουν και τον νέο κόμβο.

Αποχώρηση κόμβου: Όταν ένας κόμβος αποχωρεί από το σύστημα τότε το zone που καταλαμβάνει συνενώνεται ή αναλαμβάνεται από κάποιον γείτονα – αν δεν είναι δυνατή η συνένωση - όπως επίσης και το τμήμα του πίνακα κατακερματισμού που διατηρούσε ο κόμβος που αποχώρησε.

Εισαγωγή αντικειμένου: Για την εισαγωγή ενός αντικειμένου (K, V) με κλειδί K και τιμή V στο σύστημα CAN έχει ως εξής: Ο κόμβος που επιθυμεί να εισάγει το αντικείμενο εφαρμόζει τις συναρτήσεις κατακερματισμού στο κλειδί K για την εύρεση των συντεταγμένων

στο χώρο, σημείο P. Στη συνέχεια δρομολογείται στο σημείο αυτό και αποθηκεύεται το ζεύγος (K, V) στον κόμβο που κατέχει το zone.

Το σύστημα CAN έχει μηχανισμούς για την αποχώρηση κόμβου, την ανάκαμψη και την συντήρηση. Σε εθελούσια αποχώρηση ο κόμβος ενημερώνει και παραδίδει την πληροφορία που κατέχει σε έναν από τους γείτονές του. Οι κόμβοι ανταλλάσσουν περιοδικά μηνύματα για ανίχνευση-ενημέρωση αλλαγών που έχουν προκύψει (συντεταγμένες zone, γείτονες, συντεταγμένες γειτόνων). Οι δημιουργοί του CAN προτείνουν μια σειρά από βελτιώσεις που περιλαμβάνουν αύξηση του αριθμού των διαστάσεων ή χρήση πολλών “realities” (καρτεσιανοί χώροι) για να μειωθεί το μήκος του μονοπατιού και καλύτερη ανοχή σε αποτυχίες, καλύτερες μετρικές δρομολόγησης που λαμβάνουν υπόψη την πραγματική τοπολογία του δικτύου και τον χρόνο RTT (Round – Trip - Time) για βελτίωση του latency και overloading coordinate zones (πολλοί κόμβοι μοιράζονται το ίδιο zone) για όλα τα παραπάνω πλεονεκτήματα (μείωση του μήκους του μονοπατιού, καλύτερη ανοχή σε αποτυχίες, βελτίωση latency).



ΣΧΗΜΑ 5.7 Παράδειγμα 2-d δίκτυο CAN (πρίν και μετά την είσοδο κόμβου Z)

5.2.3 Kademlia

Το Kademlia είναι ένα peer-to-peer σύστημα αποθήκευσης και αναζήτησης ζευγών (κλειδί, τιμή). Βασικό χαρακτηριστικό του είναι ότι ελαχιστοποιεί τον αριθμό των μηνυμάτων ρύθμισης που πρέπει να στείλουν οι κόμβοι, για να μάθουν όσα χρειάζεται να ξέρουν για τους άλλους κόμβους. Οι πληροφορίες ρύθμισης εξαπλώνονται αυτόματα, καθώς διεξάγονται οι αναζητήσεις κλειδιών. Επιπλέον, οι κόμβοι έχουν την απαραίτητη γνώση και ευελιξία, ώστε να δρομολογούν ερωτήσεις μέσω διαδρομών χαμηλής καθυστέρησης. Το Kademlia χρησιμοποιεί παράλληλα ασύγχρονα ερωτήματα, για να αποφύγει καθυστερήσεις λήξης χρόνου από νεκρούς κόμβους και ο αλγόριθμος με τον οποίο ενημερώνονται οι κόμβοι για την κατάσταση των άλλων κόμβων στο σύστημα είναι ανθεκτικός απέναντι στη βασική επίθεση DoS.

Τα κλειδιά στο Kademlia είναι λέξεις των 160 bits (π.χ. ο SHA-1 κατακερματισμός μεγαλύτερων δεδομένων). Κάθε ένας από τους συμμετέχοντες στο σύστημα υπολογιστές έχει ένα node ID σε έναν χώρο κλειδιών των 160 bits. Τα ζεύγη (κλειδί, τιμή) αποθηκεύονται σε κόμβους με IDs «κοντά» στο κλειδί. Τέλος, ο αλγόριθμος δρομολόγησης βασίζεται στα node IDs και επιτρέπει τον εντοπισμό των εξυπηρετητών κοντά σε ένα κλειδί-προορισμό από κάθε κόμβο.

Ένα από τα καινοτόμα στοιχεία του Kademlia είναι η XOR τεχνική μέτρησης της απόστασης ανάμεσα σε δύο σημεία στο χώρο των κλειδιών. Η XOR τεχνική είναι συμμετρική και επιτρέπει στους κόμβους του Kademlia να λαμβάνουν ερωτήσεις αναζήτησης από ακριβώς την ίδια κατανομή κόμβων που περιέχεται στους πίνακες

δρομολόγησης τους. Ένας κόμβος Kademlia μπορεί να στείλει ερώτηση σε οποιονδήποτε κόμβο εντός ενός διαστήματος και έτσι μπορεί να επιλέξει διαδρομές ανάλογα με την καθυστέρηση ή ακόμα και να στείλει παράλληλα ασύγχρονες ερωτήσεις. Για τον εντοπισμό κόμβων κοντά σε ένα συγκεκριμένο ID το Kademlia χρησιμοποιεί έναν απλό αλγόριθμο δρομολόγησης από την αρχή μέχρι το τέλος (άλλα συστήματα χρησιμοποιούν έναν αλγόριθμο για να πλησιάσουν το ID-στόχο και έναν άλλο για τα τελευταία λίγα βήματα).

Κάθε κόμβος Kademlia έχει ένα node ID των 160 bits. Κάθε μήνυμα που μεταδίδεται από έναν κόμβο περιλαμβάνει και το node ID του, επιτρέποντας κατ' αυτόν τον τρόπο στον παραλήπτη να καταγράψει την ύπαρξη του αποστολέα.

Τα κλειδιά είναι και αυτά αναγνωριστικά των 160 bits. Για τη δημοσίευση και την εύρεση ζευγών (κλειδί, τιμή) το Kademlia χρησιμοποιεί μια δική του έννοια για την απόσταση ανάμεσα σε δύο αναγνωριστικά.

Το πρωτόκολλο Kademlia περιλαμβάνει τέσσερις απομακρυσμένες κλήσεις διαδικασίας (Remote Procedure Calls – RPCs): PING, STORE, FIND_NODE και FIND_VALUE.

Η πιο σημαντική εργασία για έναν κόμβο Kademlia είναι ο εντοπισμός των k κοντινότερων κόμβων σε ένα δεδομένο ID. Αυτή η εργασία καλείται αναζήτηση κόμβου. Το Kademlia εφαρμόζει έναν αναδρομικό αλγόριθμο για τις αναζητήσεις κόμβων. Ο κόμβος – εκκινητής της αναζήτησης – (τον ονομάζουμε αρχικό) διαλέγει α κόμβους από τον κοντινότερο μη-άδειο k-κάδο του. Εάν ο κάδος έχει λιγότερα από α στοιχεία, παίρνει απλά του κοντινότερους α κόμβους που γνωρίζει. Ο αρχικός στέλνει στη συνέχεια παράλληλα και ασύγχρονα FIND_NODE RPCs στους α κόμβους που έχει επιλέξει. Το α είναι μια παράμετρος ταυτοχρονισμού του συστήματος.

5.2.4 PASTRY

Ένα από τα μεγαλύτερα προβλήματα στις peer-to-peer εφαρμογές μεγάλης κλίμακας είναι η εύρεση αποδοτικών αλγορίθμων για την αναζήτηση αντικειμένων στο δίκτυο, μια διαδικασία που ονομάζεται και δρομολόγηση. Το Pastry είναι ένα γενικό σχήμα peer-to-peer εντοπισμού αντικειμένων και δρομολόγησης βασισμένο σε ένα αυτο-διοργανούμενο υπερκείμενο δίκτυο κόμβων που συνδέονται μέσω του Διαδικτύου. Είναι πλήρως αποκεντρωμένο, ανθεκτικό σε λάθη, κλιμακούμενο σε μέγεθος και αξιόπιστο. Το Pastry προορίζεται για γενική υποδομή για τη δημιουργία πολλών και διαφορετικών peer-to-peer εφαρμογών, όπως η εκτεταμένη ανταλλαγή και αποθήκευση αρχείων, η επικοινωνία ομάδων και τα συστήματα ονοματοδοσίας.

Η λειτουργικότητα που παρέχει στις εφαρμογές που δομούνται πάνω σε αυτό είναι η ακόλουθη: κάθε κόμβος στο δίκτυο Pastry έχει ένα μοναδικό αναγνωριστικό (nodeId). Με ένα μήνυμα και ένα κλειδί ένας κόμβος που συμμετέχει σε ένα δίκτυο Pastry δρομολογεί αποδοτικά το μήνυμα στον κόμβο με nodeId που είναι αριθμητικά πλησιέστερο στο κλειδί σε σχέση με όλους τους άλλους “ζωντανούς” κόμβους Pastry. Ο αναμενόμενος αριθμός βημάτων δρομολόγησης είναι O(logN), όπου N το πλήθος των κόμβων Pastry στο δίκτυο. Από κάθε κόμβο Pastry που περνάει ένα μήνυμα κατά την προώθησή του προς τον προορισμό, ενημερώνεται η αντίστοιχη εφαρμογή και της δίνεται η δυνατότητα να τροποποιήσει το μήνυμα ανάλογα με τις ανάγκες της.

Το Pastry λαμβάνει υπόψη του την έννοια της τοπικότητας των κόμβων που συμμετέχουν στο δίκτυο :προσπαθεί να ελαχιστοποιήσει τις αποστάσεις που ταξιδεύουν τα μηνύματα

σύμφωνα με ένα μέτρο εγγύτητας των κόμβων, όπως ο αριθμός των IP βημάτων δρομολόγησης. Κάθε κόμβος Pastry είναι ενήμερος για τους άμεσους γείτονές του στο χώρο των nodeIds και πληροφορεί τις εφαρμογές για αφίξεις νέων κόμβων, αποτυχίες και ανακάμψεις κόμβων. Επειδή τα nodeIds ανατίθενται τυχαία, είναι πολύ πιθανό μια ομάδα κόμβων με γειτονικά αναγνωριστικά να βρίσκονται αρκετά μακριά γεωγραφικά. Μια ευρηματική τεχνική διασφαλίζει ότι ανάμεσα στο σύνολο των k κόμβων με τα κοντινότερα nodeIds στο κλειδί το μήνυμα μάλλον θα φτάσει πρώτο στον κόμβο που είναι «κοντινότερα» (όσον αφορά το μέτρο εγγύτητας) στον κόμβο από τον οποίο προέρχεται το μήνυμα.

5.2.5 TAPESTRY

Το Tapestry είναι μια επεκτάσιμη υποδομή, η οποία παρέχει αποκεντρωμένο εντοπισμό και δρομολόγηση ενός αντικειμένου (Decentralized Object Location and Routing – DOLR).

Το DOLR interface εστιάζει στη δρομολόγηση μηνυμάτων σε τελικά σημεία, όπως είναι οι κόμβοι και τα αντίγραφα αντικειμένων. Το DOLR παρέχει εικονικούς πόρους, αφού τα τελικά σημεία παίρνουν ονόματα αναγνωριστικών κωδικοποιώντας μηδενική πληροφορία για τη φυσική τους θέση. Με αυτήν την υλοποίηση επιτρέπεται η παράδοση μηνυμάτων σε κινητά τελικά σημεία ή αντίγραφα τελικών σημείων κατά την παρουσία αστάθειας στην υποκείμενη υποδομή.

Σαν αποτέλεσμα, ένα δίκτυο DOLR παρέχει μια απλή πλατφόρμα πάνω στην οποία μπορούν να υλοποιηθούν κατανεμημένες

εφαρμογές, ενώ απαλλάσσει τον σχεδιαστή των εφαρμογών από το να πρέπει να λάβει υπόψιν τη δυναμικότητα του δικτύου. Ήδη το Tapestry έχει κάνει δυνατή την ανάπτυξη εφαρμογών αποθήκευσης μεγάλης κλίμακας όπως το OceanStore και συστημάτων multicast διανομής όπως το Bayeux.

Το Tapestry χρησιμοποιεί προσαρμοστικούς αλγόριθμους, για να επιδείξει αντοχή σε λάθη. Η αρχιτεκτονική του είναι τμηματική και περιλαμβάνει μια εκτεταμένη λειτουργικότητα "upcall" γύρω από έναν απλό, υψηλής απόδοσης μηχανισμό δρομολόγησης. Αυτό το API επιτρέπει στους προγραμματιστές να επεκτείνουν την υπάρχουσα λειτουργικότητα, όταν αυτή κρίνεται ανεπαρκής για την εφαρμογή τους.

Το Tapestry επιτρέπει στις εφαρμογές να τοποθετούν αντικείμενα ανάλογα με τις ανάγκες τους και δε ρυθμίζει, όπως άλλα δομημένα συστήματα peer-to-peer, τον αριθμό και τη θέση των αντιγράφων των αντικειμένων μέσω ενός DHT interface.

Το Tapestry «δημοσιεύει» δείκτες θέσης μέσα στο δίκτυο, για να διευκολυνθεί η δρομολόγηση στα αντικείμενα που έχουν μικρή διασπορά στο δίκτυο. Αυτή η τεχνική δίνει στο Tapestry ιδιότητες τοπικότητας: οι ερωτήσεις για κοντινά αντικείμενα ικανοποιούνται γενικά σε χρόνο ανάλογο της απόστασης ανάμεσα στην πηγή της ερώτησης και στο πλησιέστερο αντίγραφο του αντικειμένου.

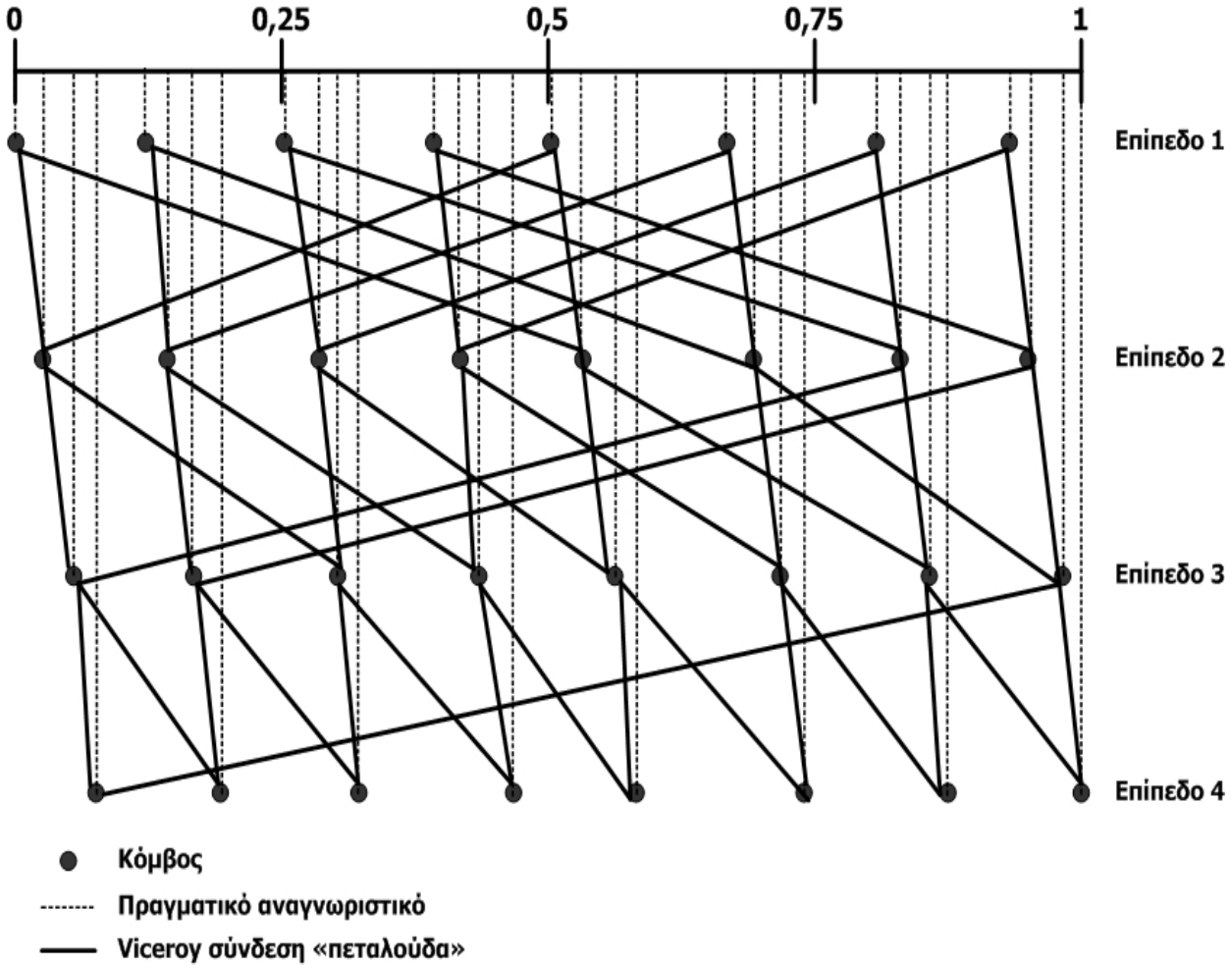
5.2.6 VICEROY

Το δίκτυο Viceroy βασίζεται στην σύνθεση ενός δικτύου που προσομοιώνει τη δομή της πεταλούδας, και ενός δακτυλίου με προγόνους και απογόνους. Επιπρόσθετα κάθε εξυπηρετητής διατηρεί και πέντε εξερχόμενες συνδέσεις ώστε να μπορεί να επιλέγει μακρινούς συνδέσμους.

Αρχικά, κάθε κόμβος επιλέγει ένα επίπεδο τυχαία έτσι ώστε όταν οι εξυπηρετητές είναι σε λειτουργία, ένα από τα logn επίπεδα να επιλέγεται ισοπίθανα. Για έναν κόμβο επιπέδου *l* δύο ακμές προστίθενται ώστε να συνδέεται με κόμβους στο επίπεδο *l+1*. Πιο συγκεκριμένα, προστίθεται μια «κάτω δεξιά» ακμή σε απόσταση περίπου 1/2*l* και μια «κάτω αριστερή» ακμή σε κοντινή απόσταση στον δακτύλιο του επιπέδου *l+1*. Επιπλέον, μια «άνω» ακμή προστίθεται σε κοντινή απόσταση από έναν κόμβο σε επίπεδο *l-1*, εφόσον *l*>1. Τέλος, προστίθενται συνδέσεις «επιπέδου-δακτυλίου» προς τον προηγούμενο και επόμενο κόμβο στο ίδιο επίπεδο *l*.

Κάθε εξυπηρετητής στο Viceroy συνδέεται με δύο τιμές που καθορίζουν τη σύνδεσή του με το υπόλοιπο σύστημα: την ταυτότητά του *s.id* στο [0,1) και το επίπεδό του *l*. Η ταυτότητά του παραμένει σταθερή αλλά το επίπεδό του μπορεί να αλλάξει κατά την εξέλιξη του δικτύου.

Οι συνδέσεις και η δομή δρομολόγησης ενός δικτύου Viceroy καθορίζεται αποκλειστικά από τις ταυτότητες και τις πληροφορίες επιπέδων των ενεργών εξυπηρετητών και δεν επηρεάζεται από προηγούμενες ρυθμίσεις ή άλλες τυχαίες πηγές.



ΣΧΗΜΑ 5.8 Το δίκτυο Viceroy

5.2.7 ΣΥΓΚΡΙΣΗ Structured P2P Συστημάτων

Algorithm taxonomy	Structured P2P Overlay Network Comparisons					
	CAN	Chord	Tapestry	Pastry	Kademlia	Viceroy
Decentralization	DHT functionality on Internet-like scale					
Architecture	Multi-dimensional ID coordinate space.	Uni-directional and circular NodeID space.	Plaxton-style global mesh network.	Plaxton-style global mesh network.	XOR metric for distance between points in the key space.	Butterfly network with connected ring of predecessor and successor links; data managed by servers.
Lookup protocol	{key, value} pairs to map a point P in the coordinate space using uniform hash function.	Matching key and NodeID.	Matching suffix in NodeID.	Matching key and prefix in NodeID.	Matching key and NodeID-based routing.	Routing through levels of tree until a peer is reached with no downlinks; vicinity search performed using ring and level-ring links.
System parameters	*N*-number of peers in network and *d*-number of dimensions.	*N*-number of peers in network.	*N*-number of peers in network and *B*-base of the chosen peer identifier.	*N*-number of peers in network and b-number of bits ($B = 2^b$) used for the base of the chosen identifier.	*N*-number of peers in network and *b*-number of bits ($B = 2^b$) of NodeID.	*N*-number of peers in network.
Routing performance	$O(d.N^{1/d})$	$O(logN)$	$O(log_B N)$	$O(log_B N)$	$O(log_B N)+c$ where c = small constant	$O(logN)$
Routing state	$2d$	$logN$	$log_B N$	$Blog_B N + Blog_B N$	$Blog_B N + B$	$logN$
Peers join/leave	$2d$	$(logN)^2$	$log_B N$	$log_B N$	$log_B N + c$ where c = small constant	$logN$
Security	Low level. Suffers from man-in-middle and Trojan attacks.					
Reliability/ fault resiliency	Failure of peers will not cause network-wide failure. Multiple peers responsible for each data item. On failures, application retries.	Failure of peers will not cause network-wide failure. Replicate data on multiple consecutive peers. On failures, application retries.	Failure of peers will not cause network-wide failure. Replicate data across multiple peers. Keep track of multiple paths to each peer.	Failure of peers will not cause network-wide failure. Replicate data across multiple peers. Keep track of multiple paths to each peer.	Failure of peers will not cause network-wide failure. Replicate data across multiple peers.	Failure of peers will not cause network-wide failure. Load incurred by lookups routing evenly distributed among participating lookup servers.

ΣΧΗΜΑ 5.9

ΚΕΦΑΛΑΙΟ 6

ΧΑΡΑΚΤΗΡΙΣΤΙΚΕΣ ΠΟΣΟΤΗΤΕΣ ΓΙΑ ΜΕΤΡΗΣΗ ΤΗΣ ΑΠΟΔΟΣΗΣ ΔΙΚΤΥΩΝ

Η μέτρηση της απόδοσης ενός σύγχρονου συστήματος (δικτύου) είναι απαραίτητη, για την κατανόηση και πρόβλεψη της συμπεριφοράς του συστήματος.

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)

6.1.1 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.

6.1.2 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).

6.1.3 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.

6.1.4 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.

6.1.5 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.

6.1.6 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.

6.1.7 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.

6.1.8 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.

ΚΕΦΑΛΑΙΟ 7

ΥΛΟΠΟΙΗΣΗ ΠΡΟΣΟΜΟΙΩΤΗ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΔΙΑΔΙΚΤΥΑΚΟΥ ΚΑΤΑΛΟΓΟΥ ΓΙΑ ΤΟ ΔΙΑΜΟΙΡΑΣΜΟ ΑΡΧΕΙΩΝ ΣΕ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ ΣΥΣΤΗΜΑΤΑ ΟΜΟΤΙΜΩΝ ΚΟΜΒΩΝ

Η υλοποίηση του προσομοιωτή ,που αποτελεί το προγραμματιστικό μέρος της εργασίας, έγινε με τρόπο τέτοιο ώστε να έχει ιδιότητες όμοιες με εκείνες του - ιεραρχικής δομής - συστήματος Kazaa.

Κατά συνέπεια, προσφέρει τις ίδιες περίπου λειτουργίες που συναντώνται και στο Kazaa,και οι οποίες αναλύθηκαν εκτενέστερα προηγουμένως.

7.1 ΠΕΡΙΓΡΑΦΗ ΥΛΟΠΟΙΗΣΗΣ ΚΑΙ ΑΝΑΛΥΣΗ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑΣ

Η εφαρμογή υλοποιείται από τις κλάσεις:

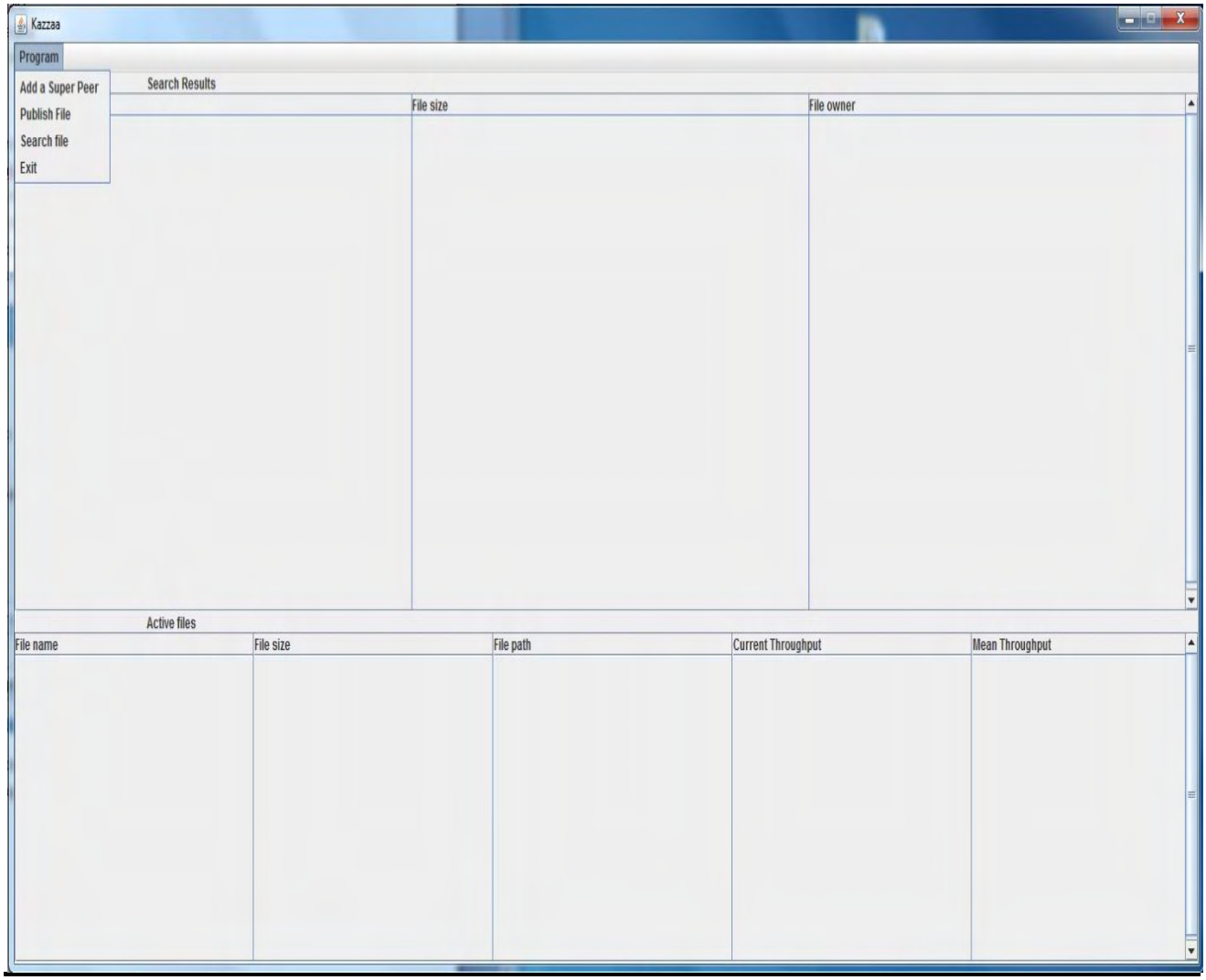
1. KazzaaClient.java
2. PeerAcceptThread.java
3. PeerConnectionServiceThread.java
4. SuperPeerConnectionServiceThread.java
5. KazzaaPeerInitDialog.java
6. Ip.java
7. KazzaaMessage.java
8. NoSuperNodeFoundException.java
9. PeerInfo.java
10. SharedFile.java
11. SuperPeerAdditionDialog.java

Κατά την εκκίνηση της εφαρμογής εμφανίζεται ένα παράθυρο (της κλάσης KazzaaPeerInitDialog.java), το οποίο ζητάει την είσοδο από τον χρήστη απαραίτητων πληροφοριών για την αρχικοποίηση της εφαρμογής.

Ο χρήστης πρέπει να εισάγει το alias(ψευδώνυμο), να επιλέξει την λειτουργία του client είτε ως super user είτε ως simple user και να δώσει το ip και το port κάποιου super user. Σε περίπτωση που ο client λειτουργεί ως super user μπορεί να αφήσει τα πεδία super-user IP και Super-user port κενά.

ΣΧΗΜΑ 7.1 Παράθυρο εκκίνησης της εφαρμογής

Αφού εισαχθούν τα απαραίτητα δεδομένα στο dialog και ο χρήστης επιλέξει Apply εμφανίζεται το παρακάτω παράθυρο:



ΣΧΗΜΑ 7.2 Μενού επιλογών του χρήστη

Στο παράθυρο διακρίνονται δύο τμήματα.

Στο πάνω μέρος βρίσκονται τα Search results, όπου υπάρχουνε λίστες, στις οποίες τοποθετούνται δεδομένα (αποτελέσματα) σύμφωνα με μια αναζήτηση που εκτελεί ο χρήστης.

Στο κάτω μέρος βρίσκονται τα active files, όπου υπάρχουν δεδομένα σχετικά με τα αρχεία τα οποία διαμοιράζονται ή έχουνε ληφθεί από τον χρήστη.

Στο menu εμφανίζονται οι επιλογές:

- **Add super peer:** Η λειτουργία είναι διαθέσιμη μόνο εάν ο client λειτουργεί σε super user mode. Εμφανίζει dialog της κλάσης SuperPeerAdditionDialog.java, στο οποίο ζητείται η εισαγωγή απαραίτητων πληροφοριών για την σύνδεση σε ένα super user (καθώς οι super users μπορούν να συνδέονται μεταξύ τους ως ομότιμοι κόμβοι)

- **Publish file:** δημοσίευση ενός αρχείου.

- **Search file:** αναζήτηση ενός αρχείου.

- **Exit:** έξοδος από την εφαρμογή.

Για την εκτέλεση μιας εντολής FETCH ο χρήστης θα πρέπει να πατήσει δεξί κλικ σε ένα αρχείο από την λίστα των search results και να του βγάλει την επιλογή Download.

Επιλέγοντάς την γίνεται η μεταφορά του αρχείου,από τον ιδιοκτήτη του αρχείου.

7.1.1 Η ΜΟΡΦΗ ΤΩΝ ΜΗΝΥΝΑΤΩΝ

Αναλυτικότερα:

Μορφή μηνυμάτων, όπως συναντάται στην κλάση KazzaaMessage.java

Type	Header	Ttl	More options
Join	JOIN	Ttl	Alias + accept Ip + accept Port
Accept	ACCEPT	Ttl	Alias
Publish	PUBLISH	Ttl	Filename + Owner's alias + size + owner's ip + owner's port
Search	SEARCH	Ttl	Search key
ResultSize	REASULTSIZE	Ttl	Result size
Results	RESULT	Ttl	Filename + Owner's alias + size + owner's ip + owner's port
Fetch	FETCH	Ttl	Filename
Invalid	INVALID	Ttl	

7.1.2 Η ΚΥΡΙΑ ΚΛΑΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

KazzaaClient.java: Η κύρια κλάση της εφαρμογής. Είναι υπεύθυνη για την δημιουργία και την παρουσίαση του GUI.

Όπως περιγράφεται παραπάνω οι επιλογές που δίνει στον χρήστη είναι:

- **Add super peer**: διαθέσιμη μόνο όταν η εφαρμογή λειτουργεί ως super peer. Η εφαρμογή κρατά λίστα με τους super peers με τους οποίους είναι συνδεδεμένη. Κάθε φορά που επιτυγχάνεται σύνδεση (αποστολή μηνύματος JOIN και λήψη απάντηση ACCEPT), μια νέα εγγραφή προστίθεται στην λίστα με τους super peers με τους οποίους είναι συνδεδεμένη η εφαρμογή. Αντίστοιχα ένα νέο SuperPeerConnectionServiceThread νήμα εκτελείται στον super peer για την εξυπηρέτηση της σύνδεσης. Οι simple peers συνδέονται με έναν μόλις super peer μόνο κατά την εκκίνηση της εφαρμογής. Η λίστα με τους super peers τους έχει πάντοτε το μέγεθος 1 (σε αντιδιαστολή με τους super peers,που κάθε ένας από αυτούς, μπορει να συνδέεται με περισσότερους από έναν, άλλους super peers).

- **Publish file**: Με την επιλογή εμφανίζεται ένα παράθυρο για την επιλογή ενός αρχείου προς δημοσίευση. Αφού επιλεγεί το αρχείο προστίθενται τα χαρακτηριστικά του(file name, file size, file path) στις λίστες active files. Στέλνεται μήνυμα PUBLISH στους super users με τους οποίους είναι συνδεδεμένη η εφαρμογή.

- **Search file**: Εμφάνιση παραθύρου, το οποίο ζητάει input από τον χρήστη σχετικά με το κλειδί προς αναζήτηση. Στέλνετε σε όλους τους super peers στους οποίους είναι συνδεδεμένος ο χρήστης μήνυμα SEARCH. Και κατόπιν αναμένονται μηνύματα RESULTSIZE από κάθε super peer. Για κάθε μήνυμα RESULTSIZE που λαμβάνει η εφαρμογή αναμένει τις αντίστοιχες πληροφορίες για τον δεδομένο αριθμό αποτελεσμάτων, που υποδεικνύεται από το result size. Τυχόν όμοια αποτελέσματα απορρίπτονται. Για κάθε αρχείο για το οποίο η εφαρμογή έλαβε αποτελέσματα γίνεται μια νέα εγγραφή στις λίστες search results. Όταν η εφαρμογή λειτουργεί σε κατάσταση super peer θα ελέγξει πρώτα τους συνδεδεμένους σε αυτήν peers για το αν διαμοιράζουνε αρχεία που ταιριάζουνε στο κλειδί.

- **Exit**: Κλείσιμο της εφαρμογής.

- **Download**: Δημιουργία ενός νήματος PeerConnectionServiceThread για να αναλάβει την λήψη του αρχείου από τον ιδιοκτήτη του.

7.1.3 ΟΙ ΥΠΟΛΟΙΠΕΣ ΚΛΑΣΕΙΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

PeerAcceptThread.java:

Το νήμα αυτό εκκινείται στον constructor του KazzaaClient. Δεδομένης μιας διεύθυνσης ip και ενός αριθμού πόρτας, που λαμβάνει ως ορίσματα στον constructor του, ανοίγει ένα ServerSocket. Στο socket αυτό το νήμα περιμένει για νέες συνδέσεις. Όταν το νήμα «ακούσει» μια νέα σύνδεση ελέγχει την κατάσταση στην οποία λειτουργεί η κύρια εφαρμογή(super peer ή simple peer). Σε περίπτωση που λειτουργεί ως super peer αναθέτει την εξυπηρέτηση της σύνδεσης σε ένα νήμα τύπου SuperPeerConnectionServiceThread, αλλιώς σε ένα νήμα PeerConnectionServiceThread. Το PeerConnectionServiceThread είναι υπεύθυνο μόνο για την μεταφορά αρχείων από peer σε peer. Αντίθετα το SuperPeerConnectionServiceThread αναλαμβάνει την γενικότερη εξυπηρέτηση ενός χρήστη, λαμβάνοντας και εξυπηρετώντας αιτήσεις όλων των τύπων. Λόγω αυτής της επικάλυψης τους, η κλάση SuperPeerConnectionServiceThread κάνει extend την κλάση PeerConnectionServiceThread.

PeerConnectionServiceThread.java:

Είναι το νήμα που αναλαμβάνει την αποστολή από τον ιδιοκτήτη και την λήψη από τον αιτούμενο ενός αρχείου. Στην εκκίνηση του το νήμα ελέγχει ποια θα είναι η λειτουργία που θα εκτελέσει.

- **Αποστολή αρχείου**: Αναμονή για μήνυμα τύπου FETCH. Αναζήτηση στο τοπικό σύστημα αρχείων για το προς αποστολή αρχείο. Αποστολή μηνύματος «OK» ή μηνύματος “INVALID” σε περίπτωση που το αρχείο δεν βρεθεί. Άνοιγμα του αρχείου και αποστολή αυτού μέσω του stream socket. Κλείσιμο της σύνδεσης.
- **Λήψη αρχείου**: Το μήνυμα “FETCH” έχει σταλεί από πριν. Το μήνυμα αναμένει απάντηση “OK” για να ξεκινήσει την λήψη. Μόλις ένα τέτοιο μήνυμα ληφθεί. Ανοίγεται αρχείο στο τοπικό σύστημα αρχείων για την αποθήκευση το αιτούμενου αρχείου. Προσθήκη εγγραφής στις λίστες active files. Διάβασμα του αρχείου από το stream socket. Υπολογισμός του Throughput της μεταφοράς μέσω ενός νήματος ThroughputCalculationThread. Κλείσιμο της σύνδεσης και σταμάτημα του νήματος για τον υπολογισμό του Throughput.

- **Υπολογισμός Throughput – ThroughputCalculationThread.java:**
  Πρόκειται για ένα νήμα εμφωλευμένο στην κλάση PeerConnectionServiceThread.

  Δεδομένης μιας περιόδου υπολογισμού του throughput, ελέγχει τα δεδομένα που έχουνε ληφθεί κατά τη διάρκεια αυτής της περιόδου. Υπολογίζει :
  - Current throughput = bytes received in period / period (bytes/sec).
  - Mean throughput = (current throughput + mean throughput) / 2 (bytes/sec).

  Αναλαμβάνει την ενημέρωση των πληροφοριών currentThroughput και mean throughput στις λίστες active file της εφαρμογής.

SuperPeerConnectionServiceThread.java:

Το νήμα αναμένει για αιτήσεις στο socket του. Οι αιτήσεις, που μπορεί να λάβει είναι οι παρακάτω:

- **Join – serviceJoinRequest**: διάβασμα του ονόματος χρήστη που αιτεί ο χρήστης. Αναζήτηση αν το όνομα αυτό χρησιμοποιείται. Ανανέωση πληροφοριών(user name, accept ip, accept port) για τον χρήστη στην λίστα με τους συνδεδεμένους users. Αποστολή μηνύματος “ACCEPT” προσθέτοντας το τελικό user name που δίνεται από τον super peer στον user.
- **Search** – **serviceSerachRequest**: αναζήτηση για το αρχείο που θέλει ο χρήστης ,στις δημοσιεύσεις κάθε χρήστη που είναι συνδεδεμένος στον super peer. Διατήρηση όλων των αποτελεσμάτων σε λίστα. Αποστολή μηνύματος “RESULTSIZE” με το συνολικό μέγεθος των αποτελεσμάτων. Για κάθε αποτέλεσμα αποστολή μηνύματος “RESULT” που περιέχει τα απαραίτητα δεδομένα για κάθε αρχείο.
- **Publish** – **servicePublishRequest**: Εισαγωγή μιας νέας εγγραφής για το προς δημοσιευση αρχείο στην λίστα των δημοσιεύσεων του peer, Που διατηρεί ο super peer.
- **Fetch**: Καλεί την συνάρτηση της πατρικής κλάσης PeerConnectionServiceThreadsendAFile για την αποστολή του αιτούμενου αρχείου.

Οι υπόλοιπες κλάσεις της εφαρμογής δεν παρουσιάζουν κάποια ειδική λειτουργικότητα, καθώς έχουν καθαρά βοηθητικό χαρακτήρα, και παραλείπεται η περιγραφή τους.

7.2 Η ΥΛΟΠΟΙΗΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

7.2.1 KazzaaClient.java (Η ΚΥΡΙΑ ΚΛΑΣΗ)

```
package kazzaclient;


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.StringTokenizer;
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.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.ScrollPaneConstants;



public class KazzaaClient extends JFrame {


   //To socket gia thn syndesh me enan oi perissoterous super nodes
   private LinkedList<Socket> peerSocket = null;
   private LinkedList<PrintWriter> outToConnection = null;
   private LinkedList<BufferedReader> inFromConnection = null;


   //plhrofories gia tous super users stous opoious einai syndedemeni i
efarmogi
   private LinkedList<PeerInfo> superUsersList;
   //lista me tous xtistes pou einai syndedemenoi me ton xrhsth(adeia se
periptwsh pou h efarmogi trexei ws aplos user)
   private LinkedList<PeerInfo> connectedPeersList;


   //flag gia na deixnei ean yparxei energh syndesh
```

```
private boolean connActive = false;
//to pseydonimo pou diatiroume sto dyktio
private String alias;

//ta arxeia pou einai dhmosieymena apo ton xrhsth
private LinkedList<SharedFile> peerShares;

private LinkedList<SharedFile> results;

//JList gia to GUI
private JList[] searchLists;
private DefaultListModel[] searchListModel;

private JList[] activeLists;
private DefaultListModel[] activeListModel;

//Thread gia thn apodoxh syndeshs apo xrhstes
private PeerAcceptThread acceptConnectionsThread;

private String acceptIp;
private int acceptPort;

private boolean search = false;
//flag gia na elegxoume ean h efarmogh trexei ws super peer h aplos peer
private boolean superPeer = false;
private boolean inited = false;
```

```
public KazzaaClient() {
    //kalei ton kataskeuastei tou Jframe
    super("Kazzaa");
    //8esh pou 8a emfanistei to para8uro
    int inset = 150;

      this.setBounds(inset, inset, 1500, 750);
    //mporei na to allaksoume giaauto vazoume false
    this.setResizable(false);
    //orizei th mpara tou menu
    this.setJMenuBar(createMenuBar());

    this.add(createView());

    Random portGen = new Random();
    acceptPort = portGen.nextInt(65535 - 1024) + 1024;

    //emfanish parathirou gia thn arxikopoihsh ths efarmoghs
    new KazzaaPeerInitDialog(KazzaaClient.this);

    if(!inited) {
      System.exit(0);
    }

    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    this.setVisible(true);
  }
```

```
   public void init(String alias, String connectIp, String connectPort, boolean
status) {

      //Se ti leeitourgia ekteleitai h efarmogh super user h simple user
      superPeer = status;
      this.alias = alias;

      peerShares = new LinkedList();

      if(superPeer) {

         connectedPeersList = new LinkedList();

      }

      try {

         acceptIp = InetAddress.getLocalHost().getHostAddress();
         acceptConnectionsThread = new PeerAcceptThread(acceptIp,
acceptPort, peerShares, activeListModel, alias, superPeer,
connectedPeersList);

      } catch (UnknownHostException ex) {
         ex.printStackTrace();
      }

      acceptConnectionsThread.start();

      peerSocket = new LinkedList();
```

```
        outToConnection = new LinkedList();
        inFromConnection = new LinkedList();

        superUsersList = new LinkedList();

        if(connectIp.length() > 0 && connectPort.length() > 0) {

            superUsersList.addFirst(new PeerInfo(connectIp,
Integer.parseInt(connectPort)));

            try {
                join(connectIp, connectPort);
            } catch (NoSuperNodeFoundException ex) {
                showErrorDialog("No Super User Found at specified Ip/Port");
            }

        }

        inited = true;

    }

    public void addSuperPeer(String connectIp, String connectPort) {
        superUsersList.addFirst(new PeerInfo(connectIp,
Integer.parseInt(connectPort)));
    }

    private void join(String connectIp, String connectPort) throws
NoSuperNodeFoundException {
```

```
        try {


            peerSocket.addFirst(new Socket(connectIp,
Integer.parseInt(connectPort)));

            outToConnection.addFirst(new
PrintWriter(peerSocket.getFirst().getOutputStream(), true));

            inFromConnection.addFirst(new BufferedReader(new
InputStreamReader(peerSocket.getFirst().getInputStream())));


            connActive = true;


            KazzaaMessage temp = new KazzaaMessage("JOIN", 1, alias + " " +
acceptIp + " " + acceptPort);

            outToConnection.getFirst().println(temp.msgToString());


            System.out.println(temp.msgToString());


            String rep = inFromConnection.getFirst().readLine();

            temp = KazzaaMessage.fromString(rep);


            alias = temp.getMoreOpt();


            System.out.println(alias);


        } catch (IOException e) {

            e.printStackTrace();

            throw new NoSuperNodeFoundException();

        }


    }
```

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

   private JMenuBar createMenuBar() {

      JMenuBar menuBar = new JMenuBar();

      JMenu menu = new JMenu("Program");

      JMenuItem item;

      item = new JMenuItem("Add a Super Peer");
      item.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {
            //Only superPeers are allowed to have more than one connections to
other superPeers
            if(superPeer) {
               new SuperPeerAdditionDialog(KazzaaClient.this);
            } else {
               showErrorDialog("No Simple Peer is allowed to connect to more
than one Super Peers");
            }
         }

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

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

   public void actionPerformed(ActionEvent e) {

      if(!connActive && !superPeer) {
         showErrorDialog("No connection to super user active");
         return;
      }

      JFileChooser chooser = new JFileChooser();

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

      //pernaei sto file pou dialexthke
      File publishFile = chooser.getSelectedFile();
      SharedFile selectedFile = new SharedFile(publishFile.getName(),
publishFile.getAbsolutePath(), publishFile.length(), acceptIp, acceptPort);
      peerShares.addLast(selectedFile);
      activeListModel[0].addElement(selectedFile.getFileName());
      activeListModel[1].addElement(selectedFile.getFileSize());
      activeListModel[2].addElement(selectedFile.getPath());
      activeListModel[3].addElement("0.0");
      activeListModel[4].addElement("0.0");
```

```
            if(!connActive)
                return;

            String additional = selectedFile.getFileName() + " " + alias + " " +
selectedFile.getFileSize() + " " + selectedFile.getOwnersIP() + " " +
selectedFile.getOwnersPort();
            KazzaaMessage temp = new KazzaaMessage("PUBLISH", 1,
additional);

            //Steile to mhnyma se olous tous pi8anous superUsers stous opoious
einai syndedemeno to peer
            for(int i=0; i<outToConnection.size(); i++){
                outToConnection.get(i).println(temp.msgToString());
            }

        }
    });

    menu.add(item);

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

        public void actionPerformed(ActionEvent e) {

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

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

            String inputValue = JOptionPane.showInputDialog("Please put a
value");
            //No input string
            if(inputValue.length() == 0) {
               showErrorDialog("Search key can not be empty");
               return;
            }

            results = new LinkedList<>();

            //Search the shared files of the peers connected to superPeer
            if(superPeer) {

                for(int i = 0; i < connectedPeersList.size(); i++) {
                  if(connectedPeersList.get(i).getUsername().equals(alias))
                      continue;

                   results.addAll(connectedPeersList.get(i).searchFile("(.*)" +
inputValue + "(.*)"));
               }

            }

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

```
            KazzaaMessage temp = new KazzaaMessage("SEARCH", 1,
inputValue);
            outToConnection.get(i).println(temp.msgToString());
         }

         try {

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

               if(peerShares.get(i).getFileName().matches(inputValue))
                  results.addLast(peerShares.get(i));

            }

            for(int i=0; i<inFromConnection.size(); i++){
               KazzaaMessage temp =
KazzaaMessage.fromString(inFromConnection.get(i).readLine());
               System.out.println(temp.msgToString());
               int resultsNumber = Integer.parseInt(temp.getMoreOpt());

               for(int j=0; j<resultsNumber; j++) {
                  temp =
KazzaaMessage.fromString(inFromConnection.get(i).readLine());
                  System.out.println(temp.msgToString());
                  String parts[];

                  StringTokenizer st = new StringTokenizer(temp.getMoreOpt());

                  parts = new String[st.countTokens()];
```

```
                for(int k=0; st.hasMoreElements(); k++) {
                    parts[k] = (String)st.nextElement();
                }

                SharedFile cur = new SharedFile(parts[0], parts[1],
Long.parseLong(parts[2]), parts[3], Integer.parseInt(parts[4]));

                //Elegxos gia pi8anwn omoies eggrafes
                int k;
                for(k = 0; k < results.size(); k++) {
                    if(results.get(k).isEqual(cur)) {
                        break;
                    }
                }
                if(k < results.size())
                    continue;

                results.add(cur);

            }
        }

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

            searchListModel[0].addElement(results.get(i).getFileName());
            searchListModel[1].addElement(results.get(i).getFileSize());
            searchListModel[2].addElement(results.get(i).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(){

            public void actionPerformed(ActionEvent e) {

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

                    PeerConnectionServiceThread temp = new
PeerConnectionServiceThread(downloadFile.getOwnersIP(),
downloadFile.getOwnersPort(), downloadFile.getFileName(), peerShares,
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<5; 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() && search) {
                    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() {
        JPanel panel = new JPanel();

        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(2500, 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.HORIZ
ONTAL_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);


        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;
    }

    private static void createAndShowGUI() {

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

    }

    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();
                }
            });
    }
}
```

7.2.2 PeerAcceptThread.java

```
package kazzaclient;

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 PeerAcceptThread extends Thread{

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

   public PeerAcceptThread(String threadIP, int threadPort,
LinkedList<SharedFile> clientShares, DefaultListModel[]
activeListModel, String alias, boolean superUs, LinkedList<PeerInfo>
connectedPeersList) {

      super();

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

   }

   public void run() {

      try {

         listeningSocket = new ServerSocket(threadPort, 0,
InetAddress.getByName(threadIP));
         System.out.println("User accepts connections at IP : " + threadIP
+ " and at port : "+ threadPort);
```

```
            while(true) {
                Socket connectionSocket = listeningSocket.accept();
                PeerInfo newPeer = new PeerInfo();
                if(superUs) {
                    connectedPeersList.addLast(newPeer);
                    SuperPeerConnectionServiceThread temp = new
SuperPeerConnectionServiceThread(connectionSocket, newPeer,
connectedPeersList, clientShares, activeListModel, alias);
                    temp.start();
                } else {
                    PeerConnectionServiceThread temp = new
PeerConnectionServiceThread(connectionSocket, clientShares,
activeListModel, alias);
                    temp.start();
                }

            }

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

    }
}
```

7.2.3 PeerConnectionServiceThread.java

```
package kazzaclient;

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 javax.swing.DefaultListModel;

public class PeerConnectionServiceThread extends Thread{

   protected Socket transferSocket;

   protected BufferedReader inFromPeer;
   protected PrintWriter outToPeer;

   protected String alias;
   private int position;

   protected boolean receiveThread;

   private String requestedFileName;
   protected LinkedList<SharedFile> peerShares;
   private DefaultListModel[] activeListModel;

   private final int BUFFER_SIZE = 5000000;


   public PeerConnectionServiceThread() {

   }

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

       this.transferSocket = transferSocket;
```

```
        try {
            outToPeer = new PrintWriter(transferSocket.getOutputStream(),
true);
            inFromPeer = new BufferedReader(new
InputStreamReader(transferSocket.getInputStream()));

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

        this.peerShares = peerShares;
        this.activeListModel = activeListModel;
        this.alias = alias;

        receiveThread = false;

    }

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

        try {

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

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

        this.requestedFileName = requestedFileName;
        this.peerShares = peerShares;
        this.activeListModel = activeListModel;
        this.alias = alias;

        receiveThread = true;
        this.position = position;

    }

    protected void receiveAFile() {

        try {
```

```
            String temp = inFromPeer.readLine();

            if(temp.equals("OK")) {
               //anoikse arxeio sto topiko systhma arxeiwn
               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
               peerShares.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);
                  try (DataInputStream fileToReceive = new
DataInputStream(transferSocket.getInputStream())) {


                     boolean moreToRead = true;

                     int tempBytes;
                     byte[] fileInBytes = new byte[BUFFER_SIZE];

                     //ekinisi thread gia ton ypologismo tou throughput
                     ThroughputCalculationThread throughput = new
ThroughputCalculationThread(activeListModel[3], activeListModel[4],
downloadFile, moreToRead);
                     throughput.start();
                     //diavazoume to arxeio apo to socket
                     while((tempBytes = fileToReceive.read(fileInBytes)) > 0) {
                        //grafoume ta bytes pou diavastikan sto arxeio sto
topiko file system
                        outFile.write(fileInBytes, 0, tempBytes);
                        //ananeonoume thn timh gia ta bytes pou exoune
diavastei sto GUI
                        activeListModel[1].set(position, downloadFile.length());
                     }
```

```
                    throughput.terminateRun();

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

                 }

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

              try {
                 Thread.sleep(2000);
              } catch(InterruptedException ie) {
              }

              inFromPeer.close();
              outToPeer.flush();
              outToPeer.close();
              transferSocket.close();

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

        }

        protected void sendAFile(KazzaaMessage msg) {

           if(!msg.getRequest().equals("FETCH")) {
              outToPeer.println(new KazzaaMessage().msgToString());
              return;
           }

           requestedFileName = msg.getMoreOpt();
           int i;

           //Search for the requested file in the active peers files
           for(i = 0; i < peerShares.size() &&
     !peerShares.get(i).getFileName().equals(requestedFileName); i++);

           if(i == peerShares.size()) {
              outToPeer.println(new KazzaaMessage().msgToString());
           } else {
              outToPeer.println("OK");
           }
```

```
        try {
           //anoigw to arxeio kai ta stream se ayto
           FileInputStream inFile;
           inFile = new FileInputStream(peerShares.get(i).getPath());
           try (DataOutputStream fileToTransfer = new
DataOutputStream(transferSocket.getOutputStream())) {

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

              //oso yparxoune dedomena sto arxeio
              while((bytesRead = inFile.read(fileInBytes)) > 0) {
                 fileToTransfer.write(fileInBytes, 0 , bytesRead);     //
stelnoume to arxeio

              }

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

           try {
                 Thread.sleep(2000);
           } catch(InterruptedException ie) {
           }

           inFromPeer.close();
           outToPeer.flush();
           outToPeer.close();
           transferSocket.close();

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

     }

     public void run() {

        if(receiveThread) {
           outToPeer.println(new KazzaaMessage("FETCH", 1,
requestedFileName).msgToString());
           receiveAFile();
        } else {
           try {
```

```
sendAFile(KazzaaMessage.fromString(inFromPeer.readLine()));
            } catch(IOException ioe) {
               ioe.printStackTrace();
            }
         }

      }

      private class ThroughputCalculationThread extends Thread {

         private DefaultListModel currentThroughputList,
meanThroughputList;
         private File downloadFile;
         private volatile boolean moreToRead;
         private int currentThroughput, meanThroughput;
         private final int PERIOD = 200;

         public ThroughputCalculationThread() {

         }

         public ThroughputCalculationThread(DefaultListModel
currentThroughputList, DefaultListModel meanThroughputList, File
downloadFile, boolean moreToRead) {

            this.currentThroughputList = currentThroughputList;
            this.meanThroughputList = meanThroughputList;
            this.downloadFile = downloadFile;
            this.moreToRead = moreToRead;

         }

         public void terminateRun() {
            moreToRead = false;
         }

         public void run() {

            long previousSize = 0, currentSize;

            try {

               sleep(PERIOD);
               currentSize = downloadFile.length();
```

```
                currentThroughput = new Double((currentSize -
previousSize)/(PERIOD/1000.0)).intValue();
                meanThroughput = currentThroughput;

                //Update Throughput Entries to GUI lists
                currentThroughputList.set(position, currentThroughput);
                meanThroughputList.set(position, meanThroughput);

                previousSize = currentSize;

            } catch(InterruptedException ie) {
                ie.printStackTrace();
            }

            while(moreToRead) {
                try {
                    sleep(PERIOD);
                    currentSize = downloadFile.length();

                    currentThroughput = new Double((currentSize -
previousSize)/(PERIOD/1000.0)).intValue();
                    meanThroughput = (meanThroughput + currentThroughput)
/ 2;

                    //Update Throughput Entries to GUI lists
                    currentThroughputList.set(position, currentThroughput);
                    meanThroughputList.set(position, meanThroughput);

                    previousSize = currentSize;

                } catch(InterruptedException ie) {
                    ie.printStackTrace();
                }
            }

            currentThroughputList.set(position, "0.0");

        }

    }
}
```

7.2.4 SuperPeerConnectionServiceThread.java

```
package kazzaclient;


import java.io.IOException;

import java.net.Socket;

import java.util.LinkedList;

import java.util.StringTokenizer;

import javax.swing.DefaultListModel;


public class SuperPeerConnectionServiceThread extends
PeerConnectionServiceThread{


   private LinkedList<PeerInfo> connectedPeers;

   private PeerInfo servicedPeer;


   public SuperPeerConnectionServiceThread(Socket communicationSocket,
PeerInfo servicedPeer, LinkedList<PeerInfo> connectedPeers,
LinkedList<SharedFile> peerShares, DefaultListModel[] activeListModel, String
alias) {


      super(communicationSocket, peerShares, activeListModel, alias);


      this.connectedPeers = connectedPeers;

      this.servicedPeer = servicedPeer;


   }


   public void run() {
```

```
boolean runFlag = true;

while(runFlag) {

   KazzaaMessage received = new KazzaaMessage();
   KazzaaMessage reply = new KazzaaMessage();

   try {
      received = KazzaaMessage.fromString(inFromPeer.readLine());
      System.out.println(received.msgToString());
   } catch(IOException ex) {
      ex.printStackTrace();
      connectedPeers.remove(servicedPeer);
      runFlag = false;
   }

   switch(received.getRequest()) {
      case "JOIN":
         serviceJoinRequest(received, reply);
         break;
      case "SEARCH":
         serviceSearchRequest(received, reply);
         break;
      case "PUBLISH":
         servicePublishRequest(received);
         break;
      case "FETCH":
         runFlag = false;
```

```
                sendAFile(received);
                break;
            case "INVALID":
                break;
            default:
                break;
        }
    }
}

public void serviceJoinRequest(KazzaaMessage received, KazzaaMessage
reply) {

    String parts[];
    StringTokenizer st = new StringTokenizer(received.getMoreOpt());
    parts = new String[st.countTokens()];

    for(int i=0; st.hasMoreElements(); i++) {
        parts[i] = (String)st.nextElement();
    }

    String user = new String(parts[0]);
    int cur = 1;

    for(int i = 0; i < connectedPeers.size(); i++) {
        if(connectedPeers.get(i).getUsername().equals(user)) {
            user = parts[0] + "" + cur;
            cur++;
            i = 0;
```

```
            }
        }

        servicedPeer.setUsername(user);

        servicedPeer.setClientIp(parts[1]);
        servicedPeer.setClientAcceptPort(Integer.parseInt(parts[2]));

        reply = new KazzaaMessage("ACCEPT", 1, servicedPeer.getUsername());
        outToPeer.println(reply.msgToString());

    }

    private void servicePublishRequest(KazzaaMessage received) {

        String parts[];
        StringTokenizer st = new StringTokenizer(received.getMoreOpt());
        parts = new String[st.countTokens()];

        for(int i=0; st.hasMoreElements(); i++) {
            parts[i] = (String)st.nextElement();
        }

        servicedPeer.addPublishedFile(new SharedFile(parts[0], parts[1],
Long.parseLong(parts[2]), servicedPeer.getClientIp(),
servicedPeer.getClientAcceptPort()));

    }
```

```
   private void serviceSearchRequest(KazzaaMessage received,
KazzaaMessage reply) {


      LinkedList<SharedFile> results = new LinkedList();


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

         if(connectedPeers.get(i).getUsername().equals(alias))

            continue;


         results.addAll(connectedPeers.get(i).searchFile("(.*)" +
received.getMoreOpt() + "(.*)"));

      }


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

         if(peerShares.get(i).getFileName().matches("(.*)" +
received.getMoreOpt() + "(.*)"))

            results.addLast(peerShares.get(i));

      }


      reply = new KazzaaMessage("RESULTSIZE", 1, results.size() + "");


      outToPeer.println(reply.msgToString());


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


         SharedFile temp = results.get(i);

         String fileInfo = temp.getFileName() + " " + temp.getPath() + " " +
temp.getFileSize() + " " + temp.getOwnersIP() + " " + temp.getOwnersPort();


         reply = new KazzaaMessage("RESULT", 1, fileInfo);
```

```
            System.out.println("REPLY : " + reply.msgToString());

            outToPeer.println(reply.msgToString());

        }

    }

}
```

7.2.5 KazzaaPeerInitDialog.java

```
package kazzaclient;


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 KazzaaPeerInitDialog extends JDialog{


   private String inputLabels[] = {"Alias", "Super-user IP", "Super-user port",
"Status"};


   private JTextField input[] = new JTextField[4];


   private boolean superUs = false;


   public KazzaaPeerInitDialog(JFrame owner) {
```

```
        //kalei ton constructor ths JDialog
        super(owner, "Connection info", true);

        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) {

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

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

}

input[3].setText("Simple user");
input[3].setEditable(false);

//pros8etoume ta 2 koumpia connect kai cancel
c.insets = new Insets(15, 40, 10, 50);
c.gridwidth = GridBagConstraints.RELATIVE;
c.weightx = 3.0;

JButton superUser = new JButton("Super User");
superUser.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {
        superUs = true;
        input[3].setText("Super user");
    }

});
```

```
addComponent(inputs, superUser, gridbag, c);

//End of Row
c.gridwidth = GridBagConstraints.REMAINDER;
c.weightx = 0.0;

JButton simpleUser = new JButton("Simple User");
simpleUser.addActionListener(new ActionListener() {

   public void actionPerformed(ActionEvent e) {
      superUs = false;
      input[3].setText("Simple user");
   }

});

addComponent(inputs, simpleUser, 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 apply = new JButton("Apply");
apply.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;
            }

            ((KazzaaClient) getParent()).init(input[0].getText(), input[1].getText(),
input[2].getText(), superUs);
            KazzaaPeerInitDialog.this.dispose();

         }
      });

      //pro8etoume to koumpi apply sto Dialog
      addComponent(inputs, apply, gridbag, c);

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

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

```
        //epistrefei to Jpanel oloklhrwmeno
        return inputs;

    }
}
```

7.2.6 Ip.java

```
package kazzaclient;

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;

   }

}
```

7.2.7 KazzaaMessage.java

```
package kazzaclient;

import java.util.StringTokenizer;

public class KazzaaMessage {

   private String request;
   private int ttl;

   private String moreOpt;

   public String getRequest() {
      return request;
   }

   public void setRequest(String request) {
      this.request = request;
   }

   public int getTtl() {
      return ttl;
   }

   public void setTtl(int ttl) {
      this.ttl = ttl;
   }
```

```
    public String getMoreOpt() {
        return moreOpt;
    }

    public void setMoreOpt(String moreOpt) {
        this.moreOpt = moreOpt;
    }

    public KazzaaMessage() {
        this.request = "INVALID";
    }

    public KazzaaMessage(String request, int ttl) {
        this.request = request;
        this.ttl = ttl;
    }

    public KazzaaMessage(String request, int ttl, String moreOpt) {

        this.request = request;
        this.ttl = ttl;
        this.moreOpt = moreOpt;

    }

    public String msgToString() {
        return request + " " + ttl + " " + moreOpt;
    }
```

```
   public static KazzaaMessage fromString(String mes) {

      KazzaaMessage temp = new KazzaaMessage();

      String parts[];

      StringTokenizer st = new StringTokenizer(mes);

      parts = new String[st.countTokens()];
      System.out.println("Tokens count " + parts.length);

      for(int i=0; st.hasMoreElements(); i++) {
         parts[i] = (String)st.nextElement();
      }
      switch (parts[0]) {
         case "JOIN":
            temp = new KazzaaMessage(parts[0], Integer.parseInt(parts[1]),
parts[2] + " " + parts[3] + " " +parts[4]);
            break;
         case "ACCEPT":
            temp = new KazzaaMessage(parts[0], Integer.parseInt(parts[1]),
parts[2]);
            break;
         case "PUBLISH":
            temp = new KazzaaMessage(parts[0], Integer.parseInt(parts[1]),
parts[2] + " " + parts[3] + " " + parts[4] + " " + parts[5] + " " + parts[6]);
            break;
         case "SEARCH":
```

```
            temp = new KazzaaMessage(parts[0], Integer.parseInt(parts[1]),
parts[2]);
            break;
         case "RESULTSIZE":
            temp = new KazzaaMessage(parts[0], Integer.parseInt(parts[1]),
parts[2]);
            break;
         case "RESULT":
            temp = new KazzaaMessage(parts[0], Integer.parseInt(parts[1]),
parts[2] + " " + parts[3] + " " + parts[4] + " " + parts[5] + " " + parts[6]);
            break;
         case "FETCH":
            temp = new KazzaaMessage(parts[0], Integer.parseInt(parts[1]),
parts[2]);
            break;
      }

      return temp;

   }

/*    public KazzaaMessage createJoinMsg(String ip, int port, String alias) {

      KazzaaMessage temp = new KazzaaMessage("JOIN", 1, alias, ip, port);
      return temp;

   }*/

}
```

7.2.8 NoSuperNodeFoundException.java

```
package kazzaclient;


public class NoSuperNodeFoundException extends Exception{


}
```

7.2.9 PeerInfo.java

```
package kazzaclient;

import java.util.LinkedList;

public class PeerInfo {

   private String alias;
   private LinkedList<SharedFile> clientShares;

   private String clientIp;
   private int clientAcceptPort;

   PeerInfo() {

      alias = "";
      clientShares = new LinkedList();
```

```
    }

    public PeerInfo(String username) {

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

    }

    public PeerInfo(String username, String clientIp, int clientAcceptPort) {

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

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

    }

    public PeerInfo(String peerIp, int peerAcceptPort) {

        this.clientIp = peerIp;
        this.clientAcceptPort = peerAcceptPort;

    }

    public String getUsername() {
        return alias;
```

```
    }

    public void setUsername(String username) {
        this.alias = 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) {

        for(int i=0; i<clientShares.size(); i++) {
            if(clientShares.get(i).isEqual(aFile))
                return;
        }

        clientShares.addLast(aFile);

    }

}
```

7.2.10 SharedFile.java

```
package kazzaclient;

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;
    }

    public boolean isEqual(SharedFile aFile) {

        if(this.fileName.equals(aFile.getFileName()) && this.fileSize ==
aFile.getFileSize()
```

```
            && this.ownersIP.equals(aFile.getOwnersIP()) && this.ownersPort ==
aFile.getOwnersPort())

          return true;


       return false;


   }


}
```

7.2.11 SuperPeerAdditionDialog.java

```
package kazzaclient;


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 SuperPeerAdditionDialog extends JDialog{


   private String inputLabels[] = {"Super-user IP", "Super-user port"};
   private JTextField input[] = new JTextField[2];


   public SuperPeerAdditionDialog(JFrame owner) {


      super(owner, "Add super user", true);


      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(SuperPeerAdditionDialog.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 apply = new JButton("Apply");
        apply.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;
                }

                ((KazzaaClient) getParent()).addSuperPeer(input[0].getText(),
input[1].getText());
                SuperPeerAdditionDialog.this.dispose();

            }
        });

        //pro8etoume to koumpi apply sto Dialog
        addComponent(inputs, apply, gridbag, c);

        //dhmiourgia koumpiou cancel
```

```
        c.weightx = 2.0;
        JButton cancel = new JButton("Cancel");
        cancel.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                SuperPeerAdditionDialog.this.dispose();
            }
        });

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

}
```

ΕΠΙΛΟΓΟΣ

Όλα και περισσότεροι χρήστες στη σημερινή εποχή προτιμούν να αποθηκεύουν τα δεδομένα τους σε συστήματα βάσεων δεδομένων, και να τα διαμοιράζονται.

Τα σχεσιακά δεδομένα όμως έχουν πλούσια δομή, σημασιολογία, και οι επερωτήσεις που τίθενται είναι πολύπλοκες.

Παρά το γεγονός ότι τα peer-to-peer συστήματα έχουν μια μεγάλη ποικιλία από εφαρμογές, υπάρχουν ακόμη πεδία εφαρμογών τα οποία δεν έχουν αναπτυχθεί πολύ.

Η κλιμακοσιμότητα αυτών των συστημάτων και η δυναμική φύση των κόμβων καθώς είτε γίνονται μέλη του δικτύου, ή το εγκαταλείπουν δημιουργεί ποικίλες προκλήσεις.
Εξαιτίας της δυναμικής φύσης κάθε κόμβου η θεώρηση ενός συνολικού σχήματος δεν ανταποκρίνεται στην πραγματική πληροφορία που είναι διαθέσιμη μια συγκεκριμένη χρονική στιγμή.

Σημαντική πρόκληση ακόμα αποτελεί ο τρόπος με τον οποίο μπορεί να περιοριστεί ο πλεονασμός της πληροφορίας και των υπολογισμών που διεξάγονται.
Ο πλεονασμός της πληροφορίας δεν μπορεί να αποφευχθεί, εκτός και αν υπάρξει κάποιος έλεγχος στην τοποθέτηση των δεδομένων όπως γίνεται στα συστήματα κατανεμημένου πίνακα κατακερματισμού (hash table).

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Networks for P2P Applications- Γεωργούλας Άγγελος

[2] Information Retrieval -Yannis Tzitzikas, U. of Crete

[3] Malicious Threats of Peer-to-Peer Networking *by Eric Chien,* Symantec Security Response

[4] Κατανεμημένη βελτιστοποίηση διότιμων συστημάτων- Ευθυμιόπουλος Νικόλαος

[5] GnuSim: A General Purpose Simulator for Gnutella and Unstructured P2P Networks-Murat Karakaya, _Ibrahim Ko¨rpeog_lu and O¨ zgu¨r Ulusoy

[6] IMPROVING THE PERFORMANCE OF THE GNUTELLA NETWORK by Andr´e Dufour

[7] STRUCTURED PEER-TO-PEER NETWORKS: HIERARCHICAL ARCHITECTURE ANDPERFORMANCE EVALUATION- *Zhonghong Ou*

[8] A SURVEY AND COMPARISON OF PEER-TO-PEER OVERLAY NETWORK SCHEMES- ENG KEONG LUA, JON CROWCROFT, AND MARCELO PIAS, UNIVERSITY OF CAMBRIDGE RAVI SHARMA, NANYANG TECHNOLOGICAL UNIVERSITY STEVEN LIM, MICROSOFT ASIA

[9] An Evaluation Framework for Structured Peer-to-Peer (Overlay) Networks-Juan José Molinero Horno

[10] A Framework for Evaluating the Performance of Cluster Algorithms for Hierarchical Networks- Jie Lian, *Member, IEEE*, Kshirasagar Naik, *Member, IEEE*, and Gordon B. Agnew, *Member, IEEE*

[11] A Framework for Evaluating the Performance of Cluster Algorithms for Hierarchical Networks -Jie Lian, *Member, IEEE*, Kshirasagar Naik, *Member, IEEE*, and Gordon B. Agnew, *Member, IEEE*

[12] A Two-Level Caching Protocol for Hierarchical Peer-to-Peer File Sharing Systems Qiying-WEI Tingting QIN* Satoshi FUJITA

[13] Performance Evaluation and Benchmarking of the JXTA Peer-To-Peer Platform- By Emir Halepovic

[14]Wikipedia