ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σύστημα Προσομοίωσης Δικτύου για Αλγόριθμους Ομότιμων (peer-to-peer) Κόμβων.

Μαρία Η. Κατσούφη

Επιβλέπων: Ευστάθιος Χατζηευθυμιάδης, Επ. Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΔΕΚΕΜΒΡΙΟΣ 2005

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σύστημα Προσομοίωσης Δικτύου για αλγόριθμους Ομότιμων (peer-to-peer) κόμβων

Μαρία Η. Κατσούφη

Α.Μ.:1206

ΕΠΙΒΛΕΠΩΝ:
Χατζηευθυμιάδης Ευστάθιος, Επ. Καθηγητής ΕΚΠΑ

Περίληψη

Τα P2P συστήματα έχουν αναδειχθεί τα τελευταία χρόνια σε ένα σημαντικό κοινωνικό και τεχνολογικό φαινόμενο. Κάποια από αυτά είναι το Gnutella, το Gnutella2, το Chord και το CAN, που εξαιτίας της ανοικτής αρχιτεκτονικής τους και της επεκτασιμότητάς τους, αποτελούν ένα πολύ ενδιαφέρον αντικείμενο μελέτης. Τα συστήματα αυτά, όπως και πολλές άλλες P2P εφαρμογές, σχηματίζουν στο επίπεδο εφαρμογής, ένα εικονικό δίκτυο (overlay network) με δικούς τους μηχανισμούς δρομολόγησης και αναζήτησης. Το σύστημα που υλοποιήθηκε είναι μία πλατφόρμα για τη μελέτη τέτοιων συστημάτων ως προς τους μηχανισμούς αυτούς.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Peer-to-peer συστήματα.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: peer-to-peer συστήματα, προσομοίωση, αλγόριθμος CAN, αλγόριθμος Chord, αλγόριθμοςGnutella.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους ανθρώπους που με διάφορους τρόπους με βοήθησαν στην εκπόνηση αυτής της εργασίας: Τον επιβλέποντα καθηγητή κ. Ευστάθιο Χατζηευθυμιάδη και κυρίως τον Τάσο Ιωαννίδη που μου έδωσε την δυνατότητα να ασχοληθώ με το θέμα των Peer-to-Peer δικτύων και με καθοδήγησε- με μεγάλη υπομονή- σε όλη την διάρκεια της εργασίας μου.

Τη συμφοιτητρια και φίλη μου Ειρήνη για τη συμπαράσταση της και τέλος, τον πατέρα μου, που με τα πολύ εύστοχα σχόλιά του με βοήθησε αποφασιστικά σε αρκετές φάσεις της εργασίας μου.

Περιεχόμενα

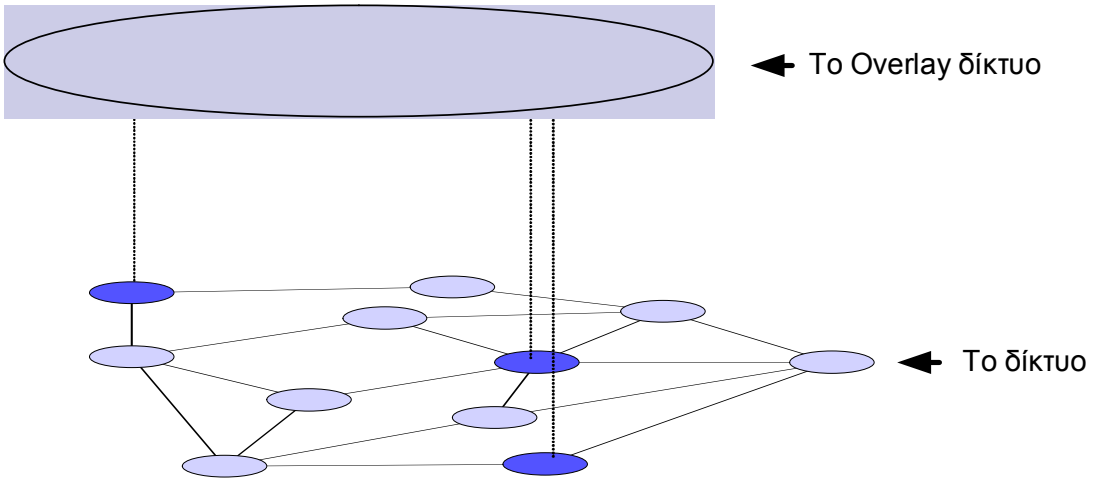
1 Εισαγωγή

1.1 Το P2P Μοντέλο Επικοινωνίας

Στην client-server αρχιτεκτονική, ένας κόμβος του δικτύου μπορεί να έχει ρόλο server ή client. Ο server είναι αυτός που παρέχει κάποια υπηρεσία μέσω του δικτύου ενώ ο client είναι αυτός που ζητά και λαμβάνει την υπηρεσία αυτή. Το μοντέλο αυτό παρουσιάζει κάποια μειονεκτήματα. Για παράδειγμα, αν κάποιος από τους servers για κάποιο λόγο δεν μπορεί να παρέχει τις υπηρεσίες του (π.χ. αν δεχτεί επίθεση, παρουσιάσει κάποια βλάβη ή δεχτεί περισσότερες αιτήσεις από αυτές που μπορεί να εξυπηρετήσει) τότε η υπηρεσία που προσέφερε δε θα είναι διαθέσιμη στο δίκτυο μέχρι να αποκατασταθεί το πρόβλημα. Το P2P μοντέλο επικοινωνίας είναι μια εναλλακτική προσέγγιση του κλασικού μοντέλου επικοινωνίας Client-Server. Εφαρμόζεται πάνω στην υπάρχουσα υποδομή, δηλαδή, θα μπορούσαμε να πούμε ότι τα δύο μοντέλα συνυπάρχουν και αλληλοσυμπληρώνονται. Στις επόμενες παραγράφους ακολουθεί η περιγραφή των βασικών αρχών και χαρακτηριστικών του P2P μοντέλου επικοινωνίας.

1.1.1 Βασικές Αρχές Μοντέλου

- ***Overlay Network:*** Βασική ιδέα στο Peer–to–Peer (P2P) μοντέλο είναι η δημιουργία ενός εικονικού δικτύου πάνω από το υπάρχον. Το δίκτυο αυτό το ονομάζουμε Overlay Network (Εικόνα 1). Δηλαδή, οι κόμβοι που συμμετέχουν ήδη στο δίκτυο, μπορούν μέσω κάποιας εφαρμογής να συμμετέχουν και σε ένα P2P δίκτυο, με σκοπό την απευθείας ανταλλαγή δεδομένων, το διαμοιρασμό πόρων και, γενικά, την ασφαλή μεταξύ τους επικοινωνία.



Εικόνα 1: Overlay Network

Στην Εικόνα 1, στο κάτω επίπεδο απεικονίζεται το δίκτυο (π.χ. το internet). Κάποιοι από τους κόμβους που είναι συνδεδεμένοι στο δίκτυο επιλέγουν να συνδεθούν και να σχηματίσουν και ένα άλλο δίκτυο (Overlay Network), ενώ εξακολουθούν να είναι συνδεδεμένοι με το αρχικό.

- ***Client και Server***: Στην Client – Server αρχιτεκτονική ο ρόλος του server, δηλαδή ενός συστήματος που πρέπει να είναι πάντα διαθέσιμο για να εξυπηρετεί τις αιτήσεις που δέχεται, έχει ανατεθεί σε συγκεκριμένους κόμβους. Ένας client απλά συνδέεται με αυτούς και ζητά τις υπηρεσίες που προσφέρουν. Στο P2P μοντέλο όμως οι ρόλοι του client και του server δεν είναι τόσο διακριτοί. Στην πραγματικότητα, ένας κόμβος (peer), που συνδέεται σε ένα P2P δίκτυο, μπορεί να είναι συγχρόνως client, αλλά και server για άλλους κόμβους του δικτύου. Δηλαδή, μπορεί συγχρόνως και να ζητά και να παρέχει υπηρεσίες με τρόπο αυτόνομο, χωρίς εξάρτηση από κάποιον κεντρικό server. Η βασική ιδέα είναι οποιαδήποτε επικοινωνία ανάμεσα σε δύο peers να γίνεται με άμεση μεταξύ τους σύνδεση. Ωστόσο, δεν αποκλείεται να προβλέπεται από ένα P2P πρωτόκολλο η μεσολάβηση κάποιου ενδιάμεσου, με σκοπό την βελτίωση της απόδοσης.

1.1.2 Άλλα χαρακτηριστικά:

- ***Αποκέντρωση – Κατανομή ευθύνης***: Με την κατάργηση του κεντρικού server που κρατάει και διαχειρίζεται όλη την πληροφορία, επιτυγχάνεται ένα είδος αποκέντρωσης και κατανομής της ευθύνης σε όλους τους κόμβους που συμμετέχουν στο P2P δίκτυο. Κάθε κόμβος είναι υπεύθυνος για την αποθήκευση μιας «ποσότητας» δεδομένων και την προσφορά κάποιας υπηρεσίας σε σχέση με τα δεδομένα αυτά. Ο τρόπος με τον οποίο γίνεται η κατανομή αυτή καθορίζεται από το εκάστοτε P2P σύστημα.
- ***Διαθεσιμότητα πόρων:*** Σε αντίθεση με το client-server μοντέλο στο οποίο υπάρχει σαφής δαχωρισμός μεταξύ αυτών που διαθέτουν τους πόρους (servers) και αυτών που τους καταναλώνουν (clients), στο P2P μοντέλο επικοινωνίας οι πόροι των περιφερειακών κόμβων γίνονται διαθέσιμοι προς χρήση από κάθε άλλο κόμβο που είναι συνδεδεμένος με το P2P δίκτυο. Με αυτόν τον τρόπο γίνεται δυνατή η αξιοποίηση των πόρων όλων των συνδεδεμένων κόμβων. Παράλληλα, κάθε κόμβος έχει δυνατότητα πρόσβασης σε πόρους που σε άλλη περίπτωση πολύ δυσκολα θα μπορούσε να έχεί.
- ***Ετερογένεια***: Οι κόμβοι που συμμετέχουν σε ένα P2P δίκτυο έχουν διαφορετικά χαρακτηριστικά μεταξύ τους. Διαφέρουν σε υπολογιστική ισχή, σε διάθεση αποθηκευτικού χώρου, ταχύτητα σύνδεσης, καθώς και στο «ποσό» της πληροφορίας που αποφασίζουν να μοιραστούν με τους υπόλοιπους κόμβους. Παρά την ετερογένεια αυτή, οι κόμβοι που συνδέονται σε ένα P2P δίκτυο θεωρούνται γενικά ισότιμοι, εκτός και αν ορίζεται κάτι διαφορετικό από το σύστημα για την βελτίωση της απόδοσης του συστήματος.
- ***Μερική διαθεσιμότητα κόμβων:*** Οι κόμβοι δεν είναι συνεχώς συνδεδεμένοι στο P2P δίκτυο. Ανά πάσα στιγμή, κόμβοι συνδέονται και αποσυνδέονται. Το σύστημα θα πρέπει να εγγυάται τη συνδεσμικότητα σε επίπεδο εφαρμογής. Δηλαδή, ένα P2P σύστημα δε θα πρέπει να καταρρέει σε περίπτωση που κάποιος/κάποιοι κόμβοι αποσυνδέονται, αλλά να εξακολουθεί να παρέχει υπηρεσίες στους υπόλοιπους συνδεδεμένους κόμβους.
- ***Πλεονασμός παρεχόμενων υπηρεσιών και πληροφοριών:*** Κάθε κόμβος που συνδέεται στο P2P σύστημα προσφέρει κάποια πληροφορία. Συνήθως περισσότεροι από έναν κόμβοι προσφέρουν την ίδια πληροφορία. Δηλαδή

περισσότεροι από έναν κόμβοι προσφέρουν την ίδια υπηρεσία. Για παράδειγμα, κάποιοι από τους χρήστες μπορεί να μοιράζονται ένα συγκεκριμένο μουσικό αρχείο. Στην περίπτωση αυτή, ένας κόμβος-χρήστης που ψάχνει το αρχείο αυτό μπορεί να το βρει σε περισσότερους από έναν κόμβους και να επιλέξει από ποιον θα το κατεβάσει, δηλαδή ποιανού την υπηρεσία θα χρησιμοποιήσει.

Τα παραπάνω χαρακτηριστικά δεν εμφανίζονται συγχρόνως σε όλα τα P2P συστήματα.

1.2. Συστήματα P2P

Γενικά, τα συστήματα διαμοιρασμού περιεχομένου [11] αποτελούνται από ένα μεγάλο πλήθος υπολογιστικών κόμβων, που μπορούν να προσφέρουν πόρους (resources) (υπολογιστικούς και αποθηκευτικούς) και περιεχόμενο (π.χ. κείμενα) στο σύστημα. Έτσι, οι κόμβοι-χρήστες μπορούν να προσφέρουν περιεχόμενο στην υπόλοιπη κοινότητα των κόμβων, ενώ επιπλέον επιτρέπουν τη χρήση των δικών τους πόρων για αποθήκευση περιεχομένου άλλων χρηστών, καθώς και την πρόσβαση σε αυτό από άλλα μέλη της κοινότητας. Δεδομένων αυτών των χαρακτηριστικών, ο κεντρικός έλεγχος της πληροφορίας σε εξειδικευμένους κόμβους δεν είναι επιθυμητός, για την αποφυγή κεντρικών σημείων αποτυχίας (central points of failure) και συμφόρησης (performance bottlenecks), καθώς και για την προστασία της ανωνυμίας των χρηστών.

Ως αποτέλεσμα όλων αυτών, το P2P (peer-to-peer) μοντέλο για την κατασκευή κατανεμημένων συστημάτων γίνεται τελευταία ιδιαίτερα δημοφιλές. Τα P2P συστήματα αποτελούνται από ένα σύνολο αυτόνομων κόμβων (peers/nodes), που μπορούν να συνεργάζονται μεταξύ τους συγκεντρώνοντας όλους τους διαθέσιμους πόρους τους με σκοπό να παράσχουν υπηρεσίες ή να διαχειριστούν συνεργατικά μεγάλες υπολογιστικές εργασίες.

Πλεονεκτήματα P2P συστημάτων

- Οι απαιτήσεις για να ξεκινήσει και να αναπτυχθεί ένα τέτοιο σύστημα είναι χαμηλές καθώς δεν απαιτείται καμία διοικητική ή οικονομική συμφωνία μεταξύ παροχέα και χρήστη (server και client), σε αντίθεση με τα κεντρικοποιημένα στυστήματα.
- Προσφέρουν έναν τρόπο αξιοποίησης της υπολογιστικής και αποθηκευτικής δυνατότητας των υπολογιστών που είναι συνδεδεμένοι στο internet.
- Η αποκεντρωμένη και κατανεμημένη φύση των P2P συστημάτων τα καθιστά περισσότερο ανθεκτικά σε σφάλματα ή επιθέσεις. Έτσι, μπορούν να θεωρηθούν ιδανικά για μακροπρόθεσμη αποθήκευση δεδομένων ή μακροσκελείς υπολογισμούς.

Μειονεκτήματα P2P συστημάτων

- Η αποκέντρωση και η κατανομή της ευθύνης, όπως αναφέρθηκε στην παραπάνω παράγραφο, είναι ένα από τα πλεονεκτήματα των P2P συστημάτων. Ωστόσο, η έλλειψη ενός κεντρικού σημείου ελέγχου εκτός από θετικό στοιχείο μπορεί να θεωρηθεί και αρνητικό. Σε ένα P2P δίκτυο, μπορεί για παράδειγμα να διακινείται ανεξέλεγκτα πληροφορία, το περιεχόμενο της οποίας συχνά είναι παράνομο.
- Σε ένα P2P δίκτυο δεν είναι δυνατό να δοθούν εγγυήσεις για την ποιότητα υπηρεσιών (Quality of Service) που παρέχονται**.**

Βασικές λειτουργίες των P2P συστημάτων

Σε ένα P2P σύστημα κόμβοι συνδέονται και σχηματίζουν σε επίπεδο εφαρμογής ένα ιδεατό δίκτυο, το P2P δίκτυο. Ενας κόμβος συνδέεται (*Join*) στο σύστημα μέσω κάποιας P2P εφαρμογής οπότε και κάνει διαθέσιμους κάποιους από τους πόρους του. Από τη στιγμή αυτή μπορεί να υποβάλλει αιτήσεις αναζήτησης (*Lookup*) για κάποια πληροφορία αλλά και να προσφέρει τις υπηρεσίες του σε άλλους κόμβους, όπως π.χ να απαντήσει σε κάποια δική τους αίτηση αναζήτησης. Παράλληλα, κάθε κόμβος εφαρμόζει εναν μηχανισμό σταθεροποίησης (*Stabilize*). Με αυτόν τον τρόπο ενημερώνει τους υπόλοιπους συνδεδεμένους κόμβους για την ύπαρξή του και συγχρόνως ενημερώνεται και αυτός για το ποιοι κόμβοι είναι συνδεδεμένοι στο δίκτυο. Τέλος, κάθε κόμβος όταν ολοκληρώσει όσα ήθελε να κάνει, αποσυνδέεται από το σύστημα. Σημειώνουμε ότι υπάρχει και η περίπτωση της «βίαιης» αποσύνδεσης του κόμβου, η οποία δε γίνεται κατόπιν δικής του επιλογής, αλλά λόγω κάποιου άλλου εξωτερικού γεγονότος.

Από την παραπάνω περιγραφή, την οποία θα μπορούσαμε να ονομάσουμε και «κύκλο ζωής» ενός P2P κόμβου, φαίνονται και οι βασικές λειτουγίες ενός P2P αλγορίθμου. Συνοπτικά, οι λειτουργίες αυτές είναι οι εξής:

- ***Join***: Έτσι ονομάζεται η λειτουργία αρχικοποίησης με την οποία ένας κόμβος συνδέεται με ένα P2P δίκτυο.
- ***Lookup***: Έτσι ονομάζεται η λειτουργία αναζήτησης δεδομένων.
- ***Stabilize***: Έτσι ονομάζεται η λειτουργία με την οποία ένας κόμβος ενημερώνεται για τους γειτονικούς του κόμβους. Καθώς οι κόμβοι συνεχώς συνδέονται και αποσυνδέονται, συνεχώς πρέπει να ανανεώνονται και οι πληροφορίες που χρειάζεται κάθε κόμβος προκειμένου να διασφαλίζεται η ορθότητα του αποτελέσματος της αναζήτησης. Η Stabilize έχει ιδιαίτερη σημασία καθώς είναι η λειτουργία που εξασφαλίζει το αναλλοίωτο ενός P2P συστήματος. Δηλαδή, φροντίζει ώστε το κάθε σύστημα να διατηρεί κάποια χαρακτηριστικά χωρίς τα οποία δεν είναι δυνατό να λειτουργήσει σωστά. Για παράδειγμα, στο Chord η Stabilize φροντίζει κάθε κόμβος να γνωρίζει πάντα τον επόμενο και τον προηγούμενό του, καθώς μόνο έτσι είναι δυνατό να γίνει αναζήτηση.
- ***Disconnect/ Depart***: Έτσι ονομάζεται η αποσύνδεση ενός κόμβου από το P2P δίκτυο. Κάποια συστήματα ορίζουν ειδικές διαδικασίες που λαμβάνουν χώρα με

την αποσύνδεση ενός κόμβου, ενώ κάποια άλλα θεωρούν, κατά κάποιο τρόπο, τη λειτουργία αυτή σαν μέρος της Stabilize.

Κάθε P2P σύστημα ορίζει τους δικούς του κανόνες για τον τρόπο με τον οποίο πραγματοποιούνται οι παραπάνω λειτουργίες.

1.3 Το Πρόβλημα Της Αναζήτησης Στα P2P Συστήματα

Το πρόβλημα της αναζήτησης (*Lookup*) [7] ορίζεται ώς εξής: Με δεδομένο ότι ένα πακέτο δεδομένων X είναι αποθηκευμένο σε κάποιους κόμβους του συστήματος, βρες αυτό το X.

Το πρόβλημα αυτό είναι κοινό σε όλα τα P2P συστήματα. Ο τρόπος με τον οποίο επιλέγεται να αντιμετωπιστεί ορίζει και κάποιες κατηγορίες στις οποίες θα μπορούσαμε να εντάξουμε τα P2P συστήματα.

1) **Αναζήτηση με τη βοήθεια κεντρικού server:** Στην προσέγγιση αυτή η αναζήτηση γίνεται με τη βοήθεια ενός κεντρικού server. Αυτός λειτουργεί σαν μία βάση δεικτών που αντιστοιχεί τα διαθέσιμα δεδομένα στους κόμβους στους οποίους υπάρχουν. Ωστόσο, η εξάρτησή του συστήματος από κεντρικούς servers το καθιστούν ευάλωτο καθώς, σε περίπτωση επίθεσης σε αυτούς, καταρρέει και όλο το υπόλοιπο P2P σύστημα. Ένα τέτοιο παράδειγμα είναι το Napster που ήταν και το πρώτο P2P σύστημα, που παρουσιάστηκε και χρησιμοποιήθηκε ευρέως για την ανταλλαγή μουσικών αρχείων.
2) **Προσέγγιση με broadcast**: Σ'αυτήν την περίπτωση, δεν υπάρχει κανένα είδος ιεραρχίας. Κατά την αναζήτηση, όλοι οι κόμβοι θεωρούνται ισότιμοι και αυτοοργανώνονται. Η αναζήτηση γίνεται με broadcasting. Έστω για παράδειγμα ότι ένας κόμβος A ψάχνει για το X. Τότε, στέλνει μία αίτηση αναζήτησης σε όλους τους γείτονικούς του κόμβους και αυτοί με τη σειρά τους σε όλους τους γειτονικούς τους κόμβους κ.ο.κ. Η προσέγγιση αυτή, λόγω του μεγάλου πλήθους μηνυμάτων που εισάγει στο δίκτυο δημιουργεί μεγάλο φόρτο και καταναλώνει πολύτιμο εύρος ζώνης (bandwidth) περιορίζοντας τις δυνατότητες κλιμάκωσης (scaling) του συστήματος. Παράδειγμα τέτοιου συστήματος είναι το Gnutella.
3) **Περιορισμένο broadcasting – Μερική Ιεραρχία:** Αυτό το σχήμα, για να περιορίσει την κίνηση που προκαλεί το broacasting, θεωρεί δύο κατηγορίες κόμβων: τους κοινούς κόμβους και τους supernodes. Με κάθε supernode συνδέονται απλοί κόμβοι. Σε γενικές γραμμές, η βασική ιδέα στην οποία βασίζεται η αναζήτηση είναι ότι μία αίτηση αναζήτησης που υποβάλλει ένας απλός κόμβος στέλνεται μόνο στους κόμβους που είναι συνδεδεμένοι στο ίδιο supernode. Παράδειγμα τέτοιων συστημάτων είναι το FastTrack, το Gnutella2 και άλλα.
4) **Distributed Hash Table (DHT) - Δομημένοι Αλγόριθμοι:** Αυτή η κατηγορία αποτελεί τη «νέα γενιά» αλγορίθμων. Ένας πίνακας κατακερματισμού (Hash Table) αποθηκεύει ζευγάρια της μορφής (κλειδί, τιμή), και η ανάκτηση μιας αποθηκευμένης τιμής γίνεται με βάση το κλειδί. Ένας κατανεμημένος πίνακας κατακερματισμού (DHT) κάνει ότι κάνει και ένας πίνακας κατακερματισμού, με τη διαφορά ότι ο πίνακας δε βρίσκεται σε έναν κόμβο, αλλά είναι κατανεμημένος σε όλους τους κόμβους του δικτύου. Δηλαδή, σε κάθε κόμβο αποθηκεύονται

ζευγάρια (κλειδί, τιμή). Η αναζήτηση μιας τιμής γίνεται με βάση ένα κλειδί. Παραδείγματα συστημάτων που βασίζονται σε DHT είναι το Chord, CAN, το Kademlia και άλλα.

1.4 P2P Αλγόριθμοι

Ακολουθεί η παρουσίαση κάποιων χαρακτηριστικών P2P αλγόριθμων:

Gnutella

Το Gnutella [4, 5, 12], είναι ένα κατανεμημένο σύστημα για αναζήτηση και διαμοιρασμό πληροφορίας. Η αρχιτεκτονική του βασίζεται στην σύνδεση πολλών ανεξάρτητων κόμβων. Οι κόμβοι αυτοί συνδέονται μεταξύ τους με TCP συνδέσεις και είναι όλοι ισότιμοι. Κάθε κόμβος προσφέρει κάθε είδος και όσα αρχεία επιθυμεί ο χρήστης. Αυτά γίνονται διαθέσιμα σε οποιονδήποτε συνδεθεί στο P2P δίκτυο. Κάθε κόμβος είναι συγχρόνως και *serv*er και cli*ent,* και γι' αυτό ονομάζεται *servent.* Κάθε κόμβος συνδέεται απευθείας με ένα σύνολο κόμβων (γειτονικοί κόμβοι), οι οποίοι θα λέμε ότι απέχουν από αυτόν ένα *hop*.

Μηνύματα
Το πρωτόκολλο Gnutella ορίζει τον τρόπο με τον οποίο επικοινωνούν οι servents μεσω του δικτύου. Συγκεκριμένα, ορίζονται πέντε είδη μηνυμάτων τα οποία ανταλλάσσονται προκειμένου να γίνουν οι βασικές λειτουργίες που γίνονται σε ένα P2P σύστημα.

Κατηγορίες Μηνυμάτων
Τα μηνύματα που ανταλλάσσονται μπορούν να υπαχθούν στις παρακάτω κατηγορίες:

- ***Μηνύματα Σύνδεσης Ping - Pong***
  - ***Ping***: Ένα μήνυμα ping αποτελείται μόνο από επικεφαλίδα.
  - ***Pong***: Περιέχει επικεφαλίδα, ενώ στο κυρίως μήνυμα περιλαμβάνονται πληροφορίες όπως η IP διεύθυνση και το port του κόμβου που το στέλνει καθώς και τον αριθμό και το συνολικό μέγεθος των αρχείων που μοιράζεται.
- ***Μηνύματα Αναζήτησης***
  - ***Query:*** Το μήνυμα αυτό αποτελείται από επικεφαλίδα και το κυρίως μήνυμα στο οποίο περιλαμβάνονται τα κριτήρια της αναζήτησης, καθώς και η ελάχιστη ταχύτητα με την οποία ο παραλήπτης πρέπει να απαντήσει.
  - ***QueryHit***: Στέλνεται σαν απάντηση σε ένα Query. Στην επικεφαλίδα, το ID του μηνύματος πρέπει να είναι ίδιο με αυτό του Query στο οποίο απαντά. Το κυρίως μήνυμα περιλαμβάνει την IP διεύθυνση, το port και την ταχύτητα του κόμβου που στέλνει το μήνυμα. Επιπλέον, περιλαμβάνει πληροφορίες για το αρχείο που του ζητήθηκε, όπως το όνομα, το μέγεθος και ένα αναγνωριστικό του αρχείου. Τέλος, περιέχεται και ένα αναγνωριστικό που καθορίζει μοναδικά τον κόμβο-αποστολέα μέσα στο δίκτυο, που χρησιμοποιείται για το Push.(βλ.: *Μηνύματα για τη Μεταφορά Αρχείων: push*).

- ***Μηνύματα για τη Μεταφορά Αρχείων***
  - ***Push***: Τα μηνύματα Push διαχειρίζονται θέματα firewall. Συγκεκριμένα, αν ο κόμβος που έστειλε το QueryHit και έχει το ζητούμενο αρχείο βρίσκεται πίσω από firewall, τότε ο κόμβος που έστειλε το Query δεν μπορεί να δημιουργήσει σύνδεση με αυτόν και να κατεβάσει το αρχείο. Σ' αυτήν την περίπτωση στέλνει ένα μήνυμα Push, προκειμένου να δημιουργήσει ο άλλος μία TCP σύνδεση μαζί του και να μπορέσει έτσι να κατεβάσει το αρχείο. Περιλαμβάνει ένα αναγνωριστικό, που καθορίζει μοναδικά τον κόμβο-αποστολέα μέσα στο δίκτυο (που λαμβάνεται από το QueryHit), το αναγνωριστικό του αρχείου που θέλουμε να κατεβάσουμε, και την IP διεύθυνση και το port του κόμβου που στέλνει το Push. (Σημειώνουμε ότι ένας κόμβος απαντάει σε ένα μήνυμα Push μόνο αν το αναγνωριστικό του κόμβου που περιλαμβάνεται στο Push και αυτού που περιλαμβάνεται στο QueryHit ταυτίζονται).

Παρατηρήσεις:

- Τα μηνύματα Pong, QueryHit και Push ακολουθούν πάντα την ίδια διαδρομή με τα αντίστοιχα αρχικά Ping, Query και Get. Αυτό εγγυάται ότι μόνο οι servents από τους οποίους πέρασαν τα αρχικά μηνύματα θα δούν τις απαντήσεις. Έτσι, αν για παράδειγμα ένας servent δεχτεί ένα μήνυμα QueryHit με ID = n, χωρίς όμως να έχει προηγουμένως λάβει ένα Query με ID = n, θα πρέπει να απορρίψει το μήνυμα αυτό.
- Ένας servent θα προωθήσει ένα εισερχόμενο μήνυμα, ένα Ping για παράδειγμα, σε όλους τους γείτονές του εκτός από αυτόν που του έστειλε το μήνυμα.
- Αν ένας servent δεχτεί δύο φορές το ίδιο μήνυμα, δεν το προωθεί ξανά για λόγους οικονομίας bandwidth.

Τα παραπάνω επιτυγχάνονται ως εξής: Όπως αναφέρθηκε και προηγουμένως (βλ. *Κατηγορίες μηνυμάτων*), στα μηνύματα Query και Pong δεν περιλαμβάνεται καμία διεύθυνση κόμβου παρά μόνο ένα αναγνωριστικό μηνύματος (ID μηνύματος). Κάθε κόμβος αποθηκεύει σε έναν πίνακα δρομολόγησης, με κλειδί το αναγωριστικό αυτό, τη σύνδεση από την οποία έφτασε το αντίστοιχο μήνυμα. Με αυτόν τον τρόπο παρέχεται και μια σχετική ανωνυμία, καθώς σε αυτό το επίπεδο οι κόμβοι δε γνωριζουν καμία διεύθυνση. Επίσης, με τη χρήση αυτών των πινάκων αποφεύγεται η προώθηση μηνυμάτων που έχουν ήδη προωθηθεί. Συγκεκριμένα, αν το ID ενός μηνύματος που φτάνει σε έναν κόμβο υπάρχει ήδη στον πίνακα δεν ξαναστέλνεται. Τέλος, με τον τρόπο αυτό, οι απαντήσεις των μηνυμάτων μπορούν να ακολουθήσουν την αντίστροφη διαδρομή: κάθε φορά ελέγχεται από ποια σύνδεση είχε προέλθει το αρχικό μήνυμα και προωθείται εκεί.

Δομή μηνυμάτων

Επικεφαλίδα μηνυμάτων

Byte	0 15	16	17	18	**19** **23**
Πεδίο	ID Μηνύματος	PayLoad Μηνύματος	TTL	Hops	Payload Length

Εικόνα 2: Επικεφαλίδα Μηνύματος στο Gnutella

ID Μηνύματος: Αποτελείται από 16 bytes και ορίζει μοναδικά το μήνυμα στο δίκτυο. Δηλαδή, δύο servents δεν μπορούν να παράγουν μηνύματα με το ίδιο ID (για κάποιο λογικό χρονικό διάστημα). Αντίστοιχα, ένας servent δεν πρέπει να παράγει ένα μήνυμα με το ίδιο ID.
PayLoad Μηνύματος: Ορίζεται το είδος του μηνύματος, δηλαδή αν θα είναι για παράδειγμα ένα Query ή ένα Ping.
TTL (Time To Live): Υποδεικνύει τον αριθμό των δρομολογήσεων του μηνύματος πριν απομακρυνθεί από το δίκτυο. Κάθε servent από τον οποίο περνάει το μήνυμα, μειώνει την τιμή του πεδίου κατά ένα και το προωθεί στον επόμενο κόμβο. Όταν η τιμή του μηδενιστεί, το μήνυμα «απορροφάται» από το δίκτυο, δηλαδή, σταματάει να προωθείται. Αυτό το πεδίο αποτελεί ουσιαστικά το μηχανισμό που εγγυάται την απόδοση του δικτύου περιορίζοντας κατά το δυνατό το φόρτο του δικτύου. Η αρχική του τιμή πρέπει να είναι τέτοια, ώστε να εγγυάται την απόδοση και συγχρόνως να αυξάνει την πιθανότητα εύρεσης του αρχείου. Μια ενδεικτική τιμή είναι το 7.
Hops: Το πεδίο αυτό δηλώνει από πόσους servents έχει περάσει το μήνυμα. Το πεδίο αυτό συνδέεται με το TTL πεδίο με τη σχέση TTL(0) = TTL(i) + Hops(i), όπου TTL(i) και Hops(i) είναι οι τιμές των πεδίων όταν το μήνυμα έχει φτάσει στο Hop i. Κάθε servent από τον οποίο περνάει το μήνυμα, αυξάνει την τιμή του πεδίου κατά ένα και το προωθεί στον επόμενο κόμβο.
Payload Length: Το μέγεθος του μηνύματος. Περιλαμβάνεται στην επικεφαλίδα, έτσι ώστε ο servent που λαμβάνει το μήνυμα να ξέρει πότε θα λάβει το τελευταίο πακέτο.

Σώμα μηνύματος
Το σώμα του μηνύματος περιλαμβάνει το port και την IP διεύθυνση του κόμβου καθώς και άλλες πληροφορίες, ανάλογα με το είδος του μηνύματος. Για παράδειγμα, ένα Pong μήνυμα περιέχει το συνολικό αριθμό των αρχείων που μοιράζεται ο κόμβος, καθώς και το συνολικό μέγεθος των αρχείων αυτών.

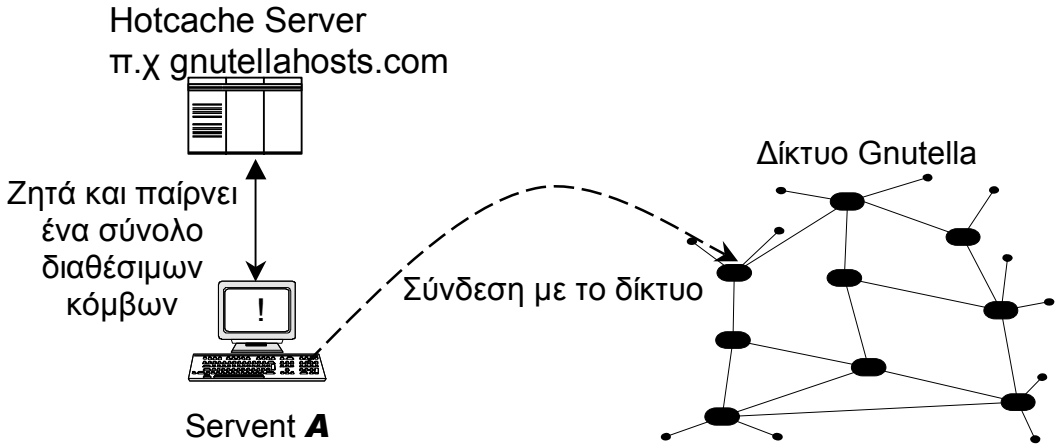
Βασικές λειτουργίες στο Gnutella
Συνοπτικά, οι βασικές λειτουργίες του Gnutella είναι οι εξής:

Λειτουργία	**Περιγραφή**
Join	Σύνδεση κόμβου με το δίκτυο
Lookup	Αναζήτηση
Stabilize	Ανακάλυψη νέων κόμβων
Έξοδος Κόμβου	Αποσύνδεση από το δίκτυο

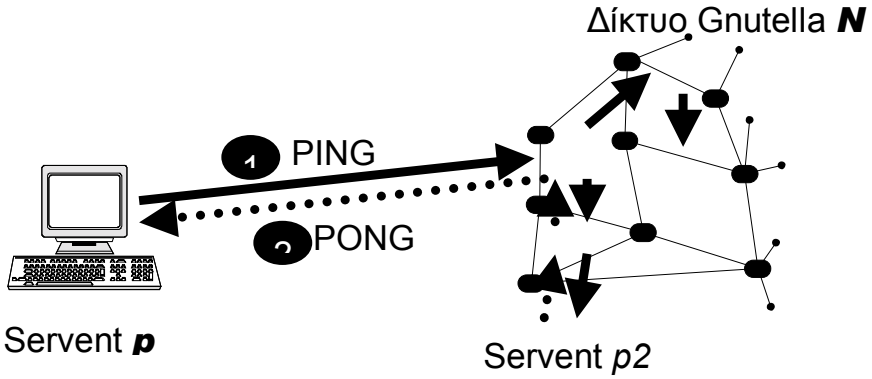
Ακολουθεί η περιγραφή των βασικών λειτουργιών του Gnutella:

Join: Μπορούμε να θεωρήσουμε ότι η Join [13] χωρίζεται σε δύο φάσεις: Στη Bootstrap και στην Initial Stabilization. Για να συνδεθεί ένας κόμβος Α με το Gnutella θα πρέπει σε πρώτη φάση να γνωρίζει έναν ή περισσότερους κόμβους, που είναι ήδη συνδεδεμένοι στο δίκτυο, ώστε να συνδεθεί με αυτούς, και μέσω αυτών και με το υπόλοιπο δίκτυο. Οι κόμβοι αυτοί είναι σχεδόν πάντα διαθέσιμοι και του στέλνουν μία λίστα με διευθύνσεις

κόμβων που είναι συνδεδεμένοι. Σε δεύτερη φάση, αφού ο κόμβος έχει συνδεθεί με το δίκτυο, στέλνει μήνυματα ping για να ενημερώσει τους υπόλοιπους κόμβους για την ύπαρξη του. Όταν ένας κόμβος λάβει ένα μήνυμα ping, το μεταβιβάζει στους γείτονές του και στέλνει σαν απάντηση ένα μήνυμα pong. Έτσι ο Α αυξάνει την πληροφορία του για τους κόμβους που είναι συνδεδεμένοι στο δίκτυο και μπορεί να συνδεθεί απευθείας με αυτούς.

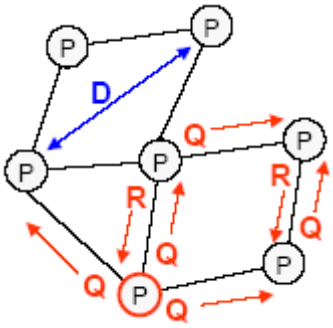


Εικόνα 3: Join στο Gnutella [22]



Εικόνα 4: Initial Stabilization – Stabilize [13]

Lookup: Η αναζήτηση γίνεται με την αποστολή ενός Query σε όλους του γείτονες του κόμβου (broadcast). Κάθε ένας από αυτούς ελέγχει αν διαθέτει τη ζητούμενη πληροφορία. Αν την έχει, τότε απαντάει στέλνοντας ένα QueryHit μήνυμα στον αποστολέα του Query. Παράλληλα, προωθεί το Query στους υπόλοιπους γείτονες του και αυτοί στους δικούς τους. Κάθε κόμβος από τον οποίο περνάει το μήνυμα μειώνει το TTL του μηνύματος. Το μήνυμα προωθείται μέχρι να μηδενιστεί το TTL του.



Εικόνα 5: Lookup P: Peer Q: Query R: QueryHit D: Download [2]

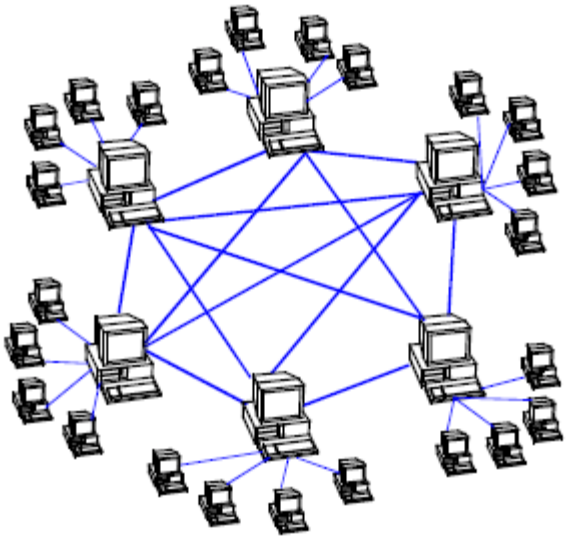
Stabilize: Κάθε κόμβος, άπαξ και συνδεθεί με το δίκτυο, στέλνει περιοδικά μηνύματα Ping για να ανακαλύψει νέους κόμβους που συνδέθηκαν στο δίκτυο και να «δηλώσει» ότι είναι ακόμα συνδεδεμένος. Αυτό γίνεται και πάλι με broadcast. Αν δε λάβει απάντηση Pong από κάποιον κόμβο, τότε συμπεραίνει ότι ο κόμβος αυτός αποσυνδέθηκε.
Disconnect/ Depart: Ο χειρισμός της εξόδου των κόμβων από το δίκτυο δεν αποτελεί ιδιαίτερη περίπτωση. Απλά, όταν ένας κόμβος αποσυνδέεται από το δίκτυο κρατάει μία λίστα με τους κόμβους με τους οποίους ήταν συνδεδεμένος για να τη χρησιμοποιήσει την επόμενη φορά που θα συνδεθεί.

Συνολική αποτίμηση Gnutella
Το Gnutella είναι ένα πολύ απλό και ανθεκτικό σύστημα που μπορεί να προσαρμόζεται δυναμικά στην συνεχή είσοδο και έξοδο κόμβων από το δίκτυο. Οι δυνατότητες κλιμάκωσής του είναι της τάξης των δεκάδων χιλιάδων κόμβων, αλλά με μικρές τροποποιήσεις μπορεί να φτάσει τους 500,000 κόμβους. Ωστόσο, παρουσιάζει κάποια βασικά μειονεκτήματα. Ένα από αυτά είναι ότι με το να στέλνονται οι αιτήσεις αναζήτησης σε όλους τους γειτονικούς κόμβους δημιουργείται πολύ μεγάλος φόρτος. Μάλιστα, η κίνηση που παράγεται από ένα Query μεγαλώνει εκθετικά, όσο σπανιότερο είναι αυτό που ψάχνουμε. Επίσης, καθώς δεν επιβάλλεται η συνεισφορά δεδομένων από τους χρήστες του συστήματος, το μεγαλύτερο ποσοστό χρηστών επιλέγει να μη συνεισφέρει πληροφορία και να παρατηρείται το φαινόμενο των “free-riders”, δηλαδή κόμβων που χρησιμοποιούν το δίκτυο χωρίς να προσφέρουν καθόλου. Ακόμα, στο Gnutella δεν μπορούν να δοθούν εγγυήσεις για το χρόνο απόκρισης σε μία αίτηση αναζήτησης, αλλά ούτε και για την εύρεση του αρχείου, αν αυτό υπάρχει. Δηλαδή, ένα αρχείο μπορεί, παρόλο που υπάρχει στο δίκτυο, να μη βρεθεί. Τέλος, η τοπολογία του δικτύου που σχηματίζεται είναι άγνωστη. Ωστόσο, πολλοί υποστηρίζουν ότι έχει power law χαρακτηριστικά.

Gnutella2

Το Gnutella2 [6] είναι ένα κατανεμημένο σύστημα για αναζήτηση και διαμοιρασμό πληροφορίας. Η αρχιτεκτονική του βασίζεται στην σύνδεση πολλών ανεξάρτητων κόμβων. Οι κόμβοι αυτοί συνδέονται μεταξύ τους με TCP ή UDP συνδέσεις, ανάλογα με το είδος των πακέτων που πρόκειται να μεταφερθούν. Σε αντίθεση με το Gnutella, οι

κόμβοι δεν είναι όλοι ισότιμοι μεταξύ τους. Υπάρχουν δύο ειδών κόμβοι: οι *κόμβοι-hubs* και οι *κόμβοι-φύλλα* (Εικόνα 6). Και οι δύο κόμβοι προσφέρουν κάθε είδος και όση ποσότητα αρχείων επιθυμούν.



Εικόνα 6: Gnutella2

***Κόμβοι-φύλλα*:** Σε αυτήν την κατηγορία ανήκουν οι περισσότεροι κόμβοι που συνδέονται με το δίκτυο. Θεωρούνται ότι βρίσκονται «στην άκρη» του δικτύου. Χαρακτηρίζονται από περιορισμένο bandwidth, CPU και RAM, χαμηλό ή απρόβλεπτο χρόνο παραμονής στο δίκτυο και αδυναμία να υποστηρίξουν πολλές συνδέσεις TCP ή UDP. Οι κόμβοι αυτοί συνδέονται με 3 κόμβους-hubs. Θα μπορούσαμε να πούμε ότι είναι σαν τους κόμβους του Gnutella.

***Κόμβοι-hubs*:** Σε αυτήν την κατηγορία ανήκουν κόμβοι, που πληρούν κάποιες προϋποθέσεις και αναλαμβάνουν επιπλέον ευθύνες. Τα κριτήρια επιλογής τους είναι τα εξής: Κατάλληλο λειτουργικό σύστημα που να μπορεί να υποστηρίξει μεγάλο αριθμό sockets, να είναι διαθέσιμοι για τουλάχιστον 2 ώρες, να έχουν επαρκές bandwidth, CPU και RAM και να μπορούν να υποστηρίξουν πολλές TCP ή UDP συνδέσεις. Οι κόμβοι αυτοί συνδέονται και μεταξύ τους σχηματίζοντας ένα δίκτυο από hubs το οποίο ονομάζουμε *hub cluster*. Κάθε hub συνδέεται με 5 έως 30 άλλους hubs, ανάλογα με το μέγεθος του δικτύου. Οι κόμβοι-hubs που ανήκουν στο ίδιο hub cluster, κρατάνε διάφορα στοιχεία ο ένας για τον άλλον και ανταλλάσσουν συνεχώς πληροφορίες σχετικά με το φόρτο του δικτύου καθώς και άλλα χρήσιμα στατιστικά. Σε κάθε hub συνδέονται 300 με 500 κόμβοι-φύλλα (ανάλογα με τις δυνατότητες του). Ένα hub cluster είναι η μικρότερη περιοχή στην οποία μπορεί να γίνει αναζήτηση. Οι κόμβοι–hubs επιφορτίζονται με κάποιες επιπλέον ευθύνες:

- Κρατάνε πληροφορίες για τα γειτονικά τους hubs μέσα στο hub-cluster και για τα γειτονικά hub-clusters και ενημερώνουν τους γείτονές τους για αλλαγές.
- Διατηρούν έναν πίνακα δρομολόγησης που αντιστοιχεί τα GUID των κόμβων σε TCP συνδέσεις.
- Κρατάνε ένα *Query Hash Table* για κάθε φύλλο αλλά και κάθε hub με το οποίο συνδέεται, έτσι ώστε να διαχειρίζονται έξυπνα τις αιτήσεις αναζήτησης (queries). Ένα Query Hash Table είναι ένας πίνακας που περιέχει πληροφορίες

για το τι *δεν* έχει ένας κόμβος (ή οι απόγονοι του). Με αυτόν τον τρόπο μια αίτηση αναζήτησης δεν προωθείται σε κόμβους στους οποίους ξέρουμε ότι δεν υπάρχει αυτό που ψάχνουμε, και έτσι μειώνεται το κόστος αναζήτης.

- Κρατάνε ένα *Συγκεντρωτικό (Aggregate) Query Hash Table* στο οποίο αποθηκεύονται τα περιεχόμενα του ίδιου αλλά και όλων των κόμβων-φύλλων (όχι των hubs) που είναι συνδεδεμένοι με αυτόν (μέσω των Query Hash Tables των κόμβων φύλλων), ώστε να να είναι σε θέση να παρέχουν πληροφορίες για το περιεχόμενο των φύλλων τους στους γειτονικούς hubs. Με τους πίνακες αυτούς, γίνεται ένα αποτελεσματικό φιλτράρισμα των αιτήσεων, ώστε να μην προωθούνται σε κόμβους στους οποίους ξέρουμε ότι δεν θα πάρουμε αποτέλεσμα. Κάθε φορά που γίνεται κάποια αλλαγή στο περιεχόμενο του ίδιου του κόμβου ή των φύλλων του, ο πίνακας αυτός πρέπει να δημιουργείται ξανά.
- Παρακολουθούν και αποφασίζουν αν κάποιος κόμβος-hub πρέπει να υποβαθμιστεί σε κόμβο-φύλλο.

Τέλος, κάθε κόμβος Gnutella2 έχει ένα *Known Hub Cache* στο οποίο κρατάει πληροφορίες (διεύθυνση κόμβου, χρόνο τελευταίας εμφάνισης και άλλα) για όλους τους κόμβους-hub που ξέρει. Χρησιμοποιείται σε διάφορες περιπτώσεις, όπως για παράδειγμα στην αναζήτηση. Το *Hub Cluster Cache* κρατάει πληροφορίες για τους γειτονικούς hubs και τους γείτονες των γειτονικών hubs, δηλαδή για κάθε hub που βρίσκεται από ένα έως δύο hops μακρυά. Είναι χρήσιμο κυρίως στους κόμβους–hubs ενώ στους κόμβους-φύλλα έχει ρόλο καθαρά πληροφοριακό.

Κατηγορίες μηνυμάτων - πακέτων

Το πρωτόκολλο Gnutella2 ορίζει τον τρόπο με τον οποίο επικοινωνούν οι κόμβοι μεταξύ τους.

Τα πακέτα αυτά μπορούν να υπαχθούν στις παρακάτω κατηγορίες:

- ***Μηνύματα Σύνδεσης***
  - ***Ping /PI***: Ένα πακέτο ping στέλνεται προκειμένου να διαπιστωθεί αν ο παραλήπτης του είναι ακόμα συνδεδεμένος και μπορεί να λάβει δεδομένα, αλλά και αν ο ίδιος ο αποστολέας μπορεί να λάβει δεδομένα μέσω UDP από κάποιον με τον οποίο δεν είναι συνδεδεμένος. Στο σώμα του μηνύματος, ανάλογα με το σκοπό για τον οποίο αποστέλλονται, μπορεί να περιέχονται κάποια πακέτα, ή τίποτα.
  - ***Pong /PO***: Ένα Pong στέλνεται ώς απάντηση σε ένα Ping. Στο σώμα του μηνύματος μπορεί να περιέχονται κάποια πακέτα, ανάλογα με το σκοπό για τον οποίο αποστέλλονται.
- ***Μηνύματα Αναζήτησης***
  - ***Gnutella2 Query /Q2*:** Με την αποστολή ενός τέτοιου πακέτου ξεκινά μία αναζήτηση. Περιέχει πληροφορίες, όπως τα κριτήρια της αναζήτησης, το GUID του αποστολέα, στοιχεία για αυθεντικοποίηση και άλλα. Όταν λαμβάνεται ένα τέτοιο πακέτο, γίνεται καταρχήν ο έλεγχος για αυθεντικοποίηση, και στη συνέχεια, αν ο παραλήπτης είναι φύλλο ή το έλαβε μέσω UDP, το προωθεί σε όλους τους κόμβους που είναι συνδεδεμένοι με αυτόν. Αν ο παραλήπτης είναι hub, το προωθεί μόνο στα φύλλα που είναι συνδεδεμένα με αυτόν.

- ***Query Acknowledgement /QA***: Όταν ένας κόμβος λάβει ένα /Q2, στέλνει ένα /QA για να ενημερώσει τον αποστολέα ότι η αίτησή του έφτασε σε ένα hub και προωθείται. Επιπλέον, περιλαμβάνει μία λίστα με τους άμεσα γειτονικούς κόμβους–hubs στους οποίους προωθήθηκε η αρχική αίτηση (ώστε να μην ξαναερωτηθούν), αλλά και τους κόμβους-hubs δεύτερου βαθμού. Με αυτόν τον τρόπο αυξάνεται και η γνώση του αποστολέα για τους hubs που μπορεί να ρωτήσει στο μέλλον, αν θέλει να συνεχίσει την αναζήτηση.
- ***QueryHit /QH2:*** Στο πακέτο αυτο περιλαμβάνονται αποτελέσματα, καθώς και όλες οι απαραίτητες πληροφορίες για να κατεβάσει το αρχείο. Περιλαμβάνει πάντα το πακέτο /GU το οποίο περιέχει το GUID του κόμβου στον οποίο βρίσκεται το αποτέλεσμα, το πακέτο /NA (Node Address) που περιέχει τη διεύθυνση του κόμβου, και άλλα προαιρετικά πακέτα.

- ***Μηνύματα για τη Μεταφορά Αρχείων Push:*** Τα μηνύματα push διαχειρίζονται θέματα firewall.
- ***Μηνύματα για Ασφάλεια κατά την αναζήτηση***
  - ***Query Key Request /QKR:*** Ένας κόμβος Α, πριν στείλει ένα /Q2 σε έναν κόμβο- hub, του στέλνει ένα /QKR. Ζητά δηλαδή ένα κλειδί (query key)το οποίο θα σχετίζεται μόνο με τον ίδιο (με τη διεύθυνσή του) και τον κόμβο – hub.
  - ***Query Key Answer /QKA:*** Όταν ένας hub λάβει ένα /QKR απαντάει με ένα /QKA. Στο πακέτο αυτό περιλαμβάνεται το query key. Ο Α θα το συμπεριλάβει στο /Q2. Όταν ο hub λαμβάνει ένα /Q2 ελέγχει το query key. Αν αυτό ταιριάζει με αυτό που είχε στείλει, απαντάει. Διαφορετικά απορρίπτει το μήνυμα. Με αυτόν τον τρόπο μειώνεται η κίνηση που προκαλείται από μία αίτηση αναζήτησης, καθώς απαντάει μόνο ο hub στον οποίο προοριζόταν η αίτηση.
- ***Μηνύματα Ενημέρωσης***
  - ***Local Node Informarion /LNI***: Κόμβοι οι οποίοι συνδέονται με TCP στέλνουν ένα τέτοιο πακέτο για να ενημερώσουν ο ένας τον άλλο για αλλαγές. Τα πακέτα-παιδιά που μπορεί να περιλαμβάνει είναι /GU για να ενημερώσει για αλλαγή στο GUID του, /NA (Node Address) για αλλαγή στη διεύθυνση, /HS (Hub Status) που αποστέλεται μόνο μεταξύ hubs και ενημερώνει για την κατάσταση του αποστολέα-hub, και άλλα.
  - ***Known Hub List /KHL***: Στέλνεται σε τακτά χρονικά διαστήματα μέσω TCP μεταξύ hubs ή μεταξύ hubs και φύλλων (και αντίστροφα), για να έχουν όλοι όσοι συμμετέχουν στο cluster μία ενημερωμένη εικόνα του.
  - ***Query Hash Tables /QHT*****:** Στέλνεται όταν αλλάζει ο Συγκεντρωτικός Πίνακας ή ένα Query Hash Table. Όταν ένα hub λάβει ένα τέτοιο πακέτο πρέπει να ενημερώσει τους πίνακες του και να το προωθήσει στους γειτονικούς hubs του cluster του.

Δομή Πακέτων-Μηνυμάτων

Τα πακέτα στο Gnutella2 έχουν δενδρική μορφή. Κάθε πακέτο μπορεί να περιλαμβάνει δεδομένα ή/και άλλα πακέτα-παιδιά.

Επικεφαλίδα (12 Bytes)			
Byte ελέγχου	Μήκος	Όνομα πακέτου	Δεδομένα ή/ και πακέτα-παιδιά

Εικόνα 7: Επικεφαλίδα Μηνυμάτων στο Gnutella2

Κάθε πακέτο περιλαμβάνει:

- **Επικεφαλίδα**
  - **Byte Ελέγχου**: (1 Byte) Έχει πάντα μή μηδενική τιμή. Όταν είναι 0 υποδεικνύει το τέλος μιας ροής πακέτων.
  - **Μήκος**: (0-3 Bytes) Πρόκειται για το συνολικό μέγεθος των δεδομένων του πακέτου. Δηλαδή, στην τιμή του περιλαμβάνεται και το μέγεθος των πακέτων-παιδιών (αν υπάρχουν), αλλά όχι της επικεφαλίδας.
  - **Όνομα πακέτου**: (1-8 Bytes) Για παράδειγμα /Q2 , όταν πρόκειται για μια αίτηση αναζήτησης.
- **Σώμα**
  - **Δεδομένα**: Τα δεδομένα που περιέχονται στο πακέτο. Υπάρχουν μόνο όταν το Byte Ελέγχου είναι διαφορετικό του 0.
  - **Πακέτα-παιδιά:** Έχουν την ίδια ακριβώς δομή με τα γονικά πακέτα.

Βασικές λειτουργίες στο Gnutella2
Οι βασικές λειτουργίες στο Gnutella2 γίνονται ως εξής:

Join: Έστω ότι ο κόμβος Α θέλει να συνδεθεί στο δίκτυο. Δημιουργεί μία TCP σύνδεση με κάποιον γνωστό hub και στη συνέχεια ακολουθεί μια διαδικασία handshake 3 βημάτων. Σημειώνουμε ότι η διαδικασία είναι ίδια με αυτή του πρωτοκόλλου Gnutella 0.6. Αρχικά, ο Α στέλνει ένα GNUTELLA CONNECT/0.6 πακέτο το οποίο περιλαμβάνει διάφορες πληροφορίες, όπως τη διεύθυνσή του και αν μπορεί να λειτουργήσει σαν hub ή απλό φύλλο. Ο hub απαντάει στον Α με ένα πακέτο στο οποίο του λέει αν μπορεί να τον δεχτεί, καθώς και το αν θα τον συνδέσει σαν φύλλο ή hub. Στο τρίτο βήμα ο Α απαντάει αν αποδέχεται τη σύνδεση αυτή.
Lookup: Η αναζήτηση γίνεται και εδώ με την αποστολή ενός Query (/Q2), με τη διαφορά ότι αυτό δεν προωθείται συνεχώς μέσα στο δίκτυο σε βάθος που καθορίζει το TTL, όπως στο Gnutella, καθώς στο Gnutella2 δεν υπάρχει TTL. Θα μπορούσαμε να πούμε ότι είναι σαν να κάνουμε αναζήτηση στο Gnutella με TTL= 2. Δηλαδή το Query προωθείται μόνο στον κόμβο-hub και στους κόμβους-φύλλα που ανήκουν στον hub αυτό.

Έστω για παράδειγμα ότι ο κόμβος-φύλλο Α θέλει να υποβάλλει ένα Query. Επιλέγει έναν κόμβο-hub με τον οποίο είναι συνδεδεμένος (από τη Known hub Cache), ο οποίος δεν έχει ερωτηθεί ακόμα στα πλαίσια αυτής της αναζήτησης και με τον οποίο δεν έχει επικοινωνήσει πρόσφατα (έστω Η) και του στέλνει το (keyed) Query (καθώς έχει προηγηθεί η ανταλλαγή μηνυμάτων /QKR- /QKA). Ο Η, μόλις το λάβει, το εξετάζει ώς προς την εγκυρότητά του και απαντά στον Α με ένα QureyAcknowledgment (/QA). Το μήνυμα αυτό περιλαμβάνει μία λίστα με τους γειτονικούς κόμβους-hubs του Η, οι οποίοι «ψάχτηκαν», και μία με τους κόμβους-hubs που βρίσκονται 2 hops μακριά από τον Η και οι οποίοι δεν «ψάχτηκαν» σε αυτήν τη φάση. Ο Α προσθέτει την πρώτη λίστα στους

κόμβους που δεν πρέπει να ξαναψάξει και τη δεύτερη στη Known Hub Cache (για μελλοντική χρήση). Εν τω μεταξύ, ο Η εξετάζει το Query για να ελέγξει αν το έχει λάβει ξανά στο παρελθόν. Αν το έχει ξαναλάβει, το απορρίπτει. Διαφορετικά, κοιτά το Query Hash Table για να βρει ποιοι κόμβοι θα μπορούσαν να ικανοποιήσουν το Query και τους το προωθεί. Παράλληλα, εξετάζει και ο ίδιος τοπικά το Query για να δει αν μπορεί να απαντήσει. Αν το έχει, επιστρέφει το αποτέλεσμα. Το ίδιο κάνουν και οι κόμβοι-φύλλα. Αυτοί μπορούν να επιλέξουν αν θα στείλουν οι ίδιοι απευθείας την απάντηση ή θα επιφορτίσουν τον Η να το κάνει. Οι υπόλοιποι κόμβοι-hubs του cluster εξετάζουν αν έχουν ξαναλάβει το Query. Αν όχι, κοιτούν το Query Hash Table τους για να βρούν ποιοι κόμβοι-φύλλα θα μπορούσαν να ικανοποιήσουν το Query και το προωθούν σε αυτούς. Δεν στέλνουν QureyAcknowledgment, ούτε προωθούν το Query στους γειτονικούς τους κόμβους-hubs. Οι κόμβοι-φύλλα και ο κόμβος-hub στέλνουν τα αποτελέσματά τους. Ο Α παίρνει τις απαντήσεις και αποφασίζει αν θα συνεχίσει να ψάχνει (και πέρα από το hub cluster) αξιοποιώντας τις πληροφορίες για νέους hubs που συνέλλεξε. Αποδεικνύεται ότι η πολυπλοκότητα της αναζήτησης στο Gnutella2 είναι O($\sqrt{n}$).

Stabilize: Όταν ένας κόμβος δεν έχει λάβει για κάποιο διάστημα κάποιο έγκυρο πακέτο από έναν κόμβο με τον οποίο είναι συνδεδεμένος, ενεργοποιεί έναν keep- alive μηχανισμό στέλνοντάς του ένα Ping. Αν δε λάβει ένα απαντητικό μήνυμα Pong μέσα σε κάποιο χρονικό διάστημα, κλείνει τη σύνδεση του. Παράλληλα, κάθε κόμβος στέλνει περιοδικά πακέτα /LNI (Local Node Information, /QHT (Query Hash Tables) και /KHL(Known Hub List). Οπότε, σε περίπτωση που κάποιος κόμβος αποσυνδεθεί, οι υπόλοιποι κόμβοι δε θα λάβουν από αυτόν πακέτα, και συνεπώς, δεν θα περιλαμβάνονται πληροφορίες για αυτόν στις ενημερωμένες λίστες τους.

Disconnect/ Depart: Όπως αναφέρθηκε και στη Stabilize, όταν ένας κόμβος αποσυνδεθεί, θα σταματήσει να στέλνει πληροφορίες στους γειτονικούς του κόμβους, οπότε και θα σταματήσει να περιλαμβάνεται στις λίστες με τους διαθέσιμους κόμβους.

Chord

Το σύστημα Chord [8] παρέχει μια κατανεμημένη υπηρεσία εντοπισμού, η οποία επιτρέπει στις εφαρμογές την εισαγωγή, τον εντοπισμό και τη διαγραφή τιμών, χρησιμοποιώντας κλειδιά. Ανήκει στη «νέα γενιά» αλγορίθμων, με πραγματικά κατανεμημένη αναζήτηση, καθώς υλοποιεί έναν Κατανεμημένο Πίνακα Κατακερματισμού (Distributed Hash Table). Κάθε κόμβος που συμμετέχει στο Chord χαρακτηρίζεται από ένα κλειδί που προκύπτει με κατακερματισμό (hashing) του ζευγαριού (IP address, port) με τον αλγόριθμο SHA-1.

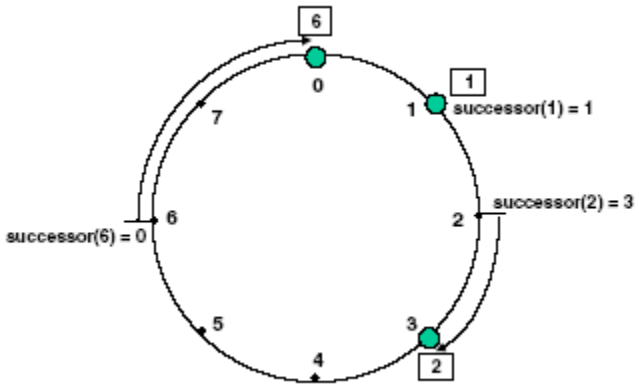
π. χ. (150.140.143.1, 6362) $\xrightarrow{\text{SHA-1}}$ nodeID = 62

Ένα αντίστοιχο κλειδί δημιουργείται και για κάθε πακέτο δεδομένων.

π. χ. key = “A” $\longrightarrow$ keyID = 62

Κάθε κόμβος διατηρεί έναν πίνακα δρομολόγησης (*finger table*) με τις εξής πληροφορίες: τη διεύθυνση του προηγούμενου και του επόμενου κόμβου, που είναι και η πιο σημαντική, καθώς και μία λίστα δεικτών προς επόμενους κόμβους. Κάθε καταχώρηση i του πίνακα περιέχει το αναγνωριστικό του πρώτου κόμβου, s, που διαδέχεται τον n σε απόσταση τουλάχιστον 2^{i-1} πάνω στον αναγνωριστικό κύκλο (mod 2^m), για $1 \leq i \leq m$ και έναν δείκτη στον προηγούμενο κόμβο. Επιπλέον, κρατάει και μία λίστα με τους successors του, μήκους r = logN, όπου N το πλήθος των συνδεδεμένων κόμβων στο σύστημα.

Ο καταμερισμός των πακέτων στους κόμβους που συμμετέχουν στο δίκτυο γίνεται ώς εξής: Οι κόμβοι ανάλογα με το κλειδί τους διατάσσονται σε ένα δακτύλιο. Ενα πακέτο δεδομένων ανήκει σε ένα κόμβο, αν και μόνο αν το κλειδί του είναι μεγαλύτερο ή ίσο του κλειδιού του κόμβου και δεν υπάρχει άλλος κόμβος με ενδιάμεσό τους κλειδί. Όπως αναφέρθηκε και παραπάνω, η i-οστή καταχώρηση του πίνακα ενός κόμβου n περιέχει το αναγνωριστικό του πρώτου κόμβου, s, που διαδέχεται τον n σε απόσταση τουλάχιστον 2^{i-1} στον δακτύλιο. Ο κόμβος s (successor) αναφέρεται ως ο i-οστός δείκτης (finger) του κόμβου n. Δηλαδή ισχύει $s=successor(n+2^{i-1})$ ενώ συμβολίζεται και με n.finger[i].node.Οπότε ο successor(k) είναι υπεύθυνος για το k. Τα παραπάνω φαίνονται στο παρακάτω σχήμα (Εικόνα 8).



Εικόνα 8: Δακτύλιος του Chord για τους κόμβους 0,1,3

Στην Εικόνα 8 παρουσιάζεται ένας δακτύλιος με 3 κόμβους (m=3): 0, 1, 3. Σύμφωνα με τα παραπάνω, θα ισχύει ότι successor(1)= 1,οπότε το κλειδί (keyId) με τιμή 1 θα εντοπιζόταν στον κόμβο 1. Αντίστοιχα, το κλειδί με τιμή 2 θα εντοπιζόταν στον 3 και το 6 στον 0.

Με αυτή την προσέγγιση κάθε κόμβος αποθηκεύει πληροφορία για ένα μικρό αριθμό άλλων κόμβων, ενώ η πληροφορία που διατηρείται για άλλους κόμβους μειώνεται εκθετικά σε σχέση με την απόσταση του από τους κόμβους αυτούς (δηλαδή κρατάει περισσότερες πληροφορίες για τους κοντινούς κόμβους). Αποδεικνύεται ότι αν έχουμε ένα σύνολο N κόμβων και K κλειδιών, ο κάθε κόμβος είναι υπεύθυνος για το πολύ (1+e)K/N κλειδιά, όπου e= O(logN).

Το μέγεθος του κλειδιού πρέπει να είναι τέτοιο ώστε η πιθανότητα δύο κλειδιά να αντιστοιχούν στον ίδιο κόμβο να είναι πολύ μικρή.

Στη συνέχεια παρουσιάζονται βασικές λειτουργίες του Chord. Συνοπτικά είναι οι εξής:

Πίνακας 1: Βασικές Λειτουργίες Chord

Λειτουργία	***Περιγραφή***
insert(key, value)	Εισάγει το (key, value) σε r κόμβους
lookup(key)	Επιστρέφει την τιμή που σχετίζεται με το κλειδί
update(key, newval)	Εισάγει το (key, newval) σε r κόμβους
join(n)	Συνδέει έναν νέο κόμβο με τον κόμβο n (Join)
leave()	Αποσυνδέεται από το Chord
Stabilize()	Περιοδική ενημέρωση προηγούμενου κόμβου

Αναλυτικότερα:

Insert: Όταν καλείται η insert(key, value), το ζευγάρι (κλειδί, τιμή) αποθηκεύεται σε r επιλεγμένους κόμβους. Η τιμή του r είναι μία παράμετρος του συστήματος και καθορίζει τον επιθυμητό βαθμό πλεονασμού σε αυτό.

Lookup: Με την κλήση της lookup(key) εντοπίζεται το ζευγάρι (κλειδί, τιμή) σε έναν κόμβο και επιστρέφεται η αντίστοιχη τιμή. Η αναζήτηση γίνεται με χρήση της λίστας δεικτών. Αν ο τρέχων κόμβος έχει το κλειδί, το επιστρέφει. Διαφορετικά, βρίσκει τον κόμβο που είναι πλησιέστερα (αλλά πριν) από το δεδομένο κλειδί από τη λίστα δεικτών του και του προωθεί την ερώτηση (αναδρομικά). Το πλήθος των μηνυμάτων που ανταλλάσσονται είναι O(log n).

Ακολουθεί ο ψευδοκώδικας για τη lookup.

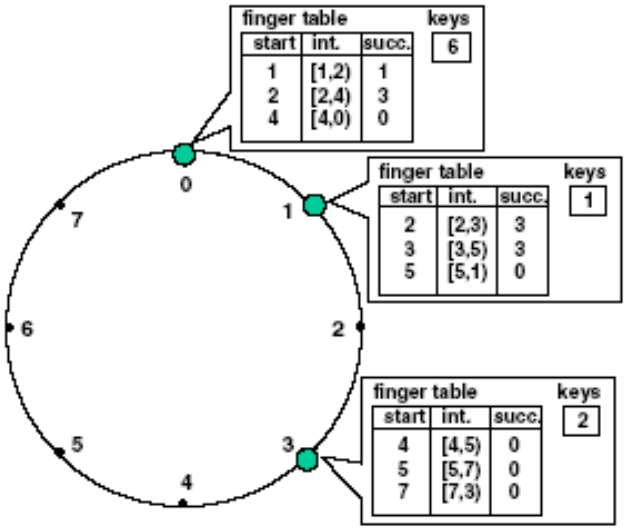
```
// ask node n to find id's successor
n. find_ successor (id)
        n' = find_ predecessor (id);
        return n'. successor;

// ask node n to find id's predecessor
n.find_ predecessor(id)
        n'=_n;
        while (id ∈ (n', n'. successor)
                n'= n'. closest_ preceding_ finger (id);
        return n0;

// return closest finger preceding id
n'.closest_ preceding _finger(id)
        for i = m downto 1
                if (finger[i].node ∈ (n, id))
                        return finger[i].node;
        return n;
```

find_ successor: Βρίσκει τον αμέσως προηγούμενο κόμβο p από αυτόν που περιέχει το ζητούμενο id. Ο successor του p είναι αυτός που περιέχει το id.
find_ predecessor: όταν ο n καλεί την find_ predecessor, ρωτάει μια σειρά κόμβων κατά μήκος του κύκλου. Έστω ότι ρωτάει τον n'. Αν το ζητούμενο id πέφτει ανάμεσα στο id του n' και του successor n, τότε επιστρέφεται το n'. Διαφορετικά, ο n ρωτά τον n' για τον κοντινότερο στο ζητούμενο id κόμβο που ξέρει.

- ✓ **Παράδειγμα lookup**:
  Έστω ότι ο κόμβος με nodeId= 3 ψάχνει για το keyId=1. Όπως φαίνεται και στην Εικόνα 9, το 1 βρίσκεται στο κυκλικό διάστημα [7, 3). Άρα ο 3 ελέγχει την τρίτη καταχώρηση του πίνακα δρομολόγησής του που του υποδεικνύει τον κόμβο 0. Επειδή το 0 προηγείται του 1, ο 3 θα ζητήσει από τον 0 να βρει τον successor του 1. Ο 0 βρίσκει από τον πίνακα δρομολόγησής του ότι ο successor του 1 είναι ο ίδιος ο 1, οπότε και το επιστρέφει.



Εικόνα 9: Πίνακες Δρομολόγησης - Finger Tables

Update: Η ενημέρωση του ζευγαριού (κλειδί, τιμή) γίνεται με την update(key, value), αλλά μόνο από το δημιουργό του συγκεκριμένου κλειδιού. Παρατηρούμε ότι η λειτουργία της διαγραφής ενός ζευγαριού πραγματοποιείται με χρήση της update.
Join: Για την εισαγωγή ενός κόμβου στο σύστημα. Ο νέος κόμβος συνδέεται με έναν κόμβο του δικτύου. Αυτός, τον ενημερώνει για το ποιος είναι ο επόμενός του. Στη συνέχεια ο νέος κόμβος καλεί τη Stabilize η οποία θα του χτίσει τη λίστα δεικτών του και θα ενημερώσει τους υπόλοιπους για την ύπαρξή του. Ο αριθμός των κόμβων που χρειάζεται να ενημερωθούν σε μια εισαγωγή είναι O(logn) ενώ για την εύρεση και την ενημέρωσή τους απαιτείται ($\log^2$ n). Σημειώνουμε ότι, με κάποιες τροποποιήσεις, ο χρόνος αυτός μπορεί να μειωθεί σε O(logn). Ακολουθεί ο ψευδοκώδικας για τη Join.

```
// node n joins the network
// node n' is an arbitrary node in the network
```

```
n.join(n')
predecessor = nil;
successor = n'.find_ successor(n);
```

Leave (Disconnect/ Depart)*:* Για την έξοδο ενός κόμβου από το δίκτυο. Διακρίνουμε δύο περιπτώσεις: Η πρώτη είναι η έξοδος ενός κόμβου λόγω αποτυχίας, και η άλλη, η ηθελημένη έξοδος από το δίκτυο. Την πρώτη περίπτωση θεωρούμε ότι τη διαχειρίζεται η stabilize, όπως περιγράφεται παρακάτω, με τη διατήρηση της λίστας με successors. Στη δεύτερη περίπτωση, ο κόμβος που φεύγει μπορεί να μεταφέρει τα κλειδιά του στον επόμενό του, ή να ενημερώσει τον προηγούμενό του για τον νέο επόμενο. Τα κλειδιά που μεταφέρονται είναι O(1/n).

Stabilize: Σκοπός της λειτουργίας αυτής είναι να κρατά ενημερωμένους τους δείκτες προς τους successors. Η ορθότητα των δεικτών αυτών εγγυάται την ορθότητα των πινάκων δρομολόγησης, και συνεπώς, και της lookup. Αν μια lookup γίνει αμέσως μετά από μία join η οποία επηρέασε κάποιους κόμβους του δακτυλίου, 3 πράγματα μπορούν να συμβούν. Πρώτο και συνηθέστερο είναι οι δείκτες των κόμβων που εμπλέκονται στη lookup να είναι πρόσφατα ενημερωμένοι και η lookup να βρει τον σωστό successor. Δεύτερη περίπτωση είναι οι δείκτες προς τους successors να είναι ανακριβείς. Σε αυτήν την περίπτωση η lookup δίνει σωστά αποτελέσματα, αλλά χρειάζεται περισσότερο χρόνο. Η τελευταία περίπτωση είναι όταν οι κόμβοι που επηρεάστηκαν από την join έχουν λανθασμένους δείκτες προς τους successors, ή τα κλειδιά δεν έχουν ακόμα μεταφερθεί στους νέους κόμβους. Σε αυτήν την περίπτωση, η lookup ενδέχεται να δώσει λάθος αποτέλεσμα, δηλαδή να μη βρίσκει το ζητούμενο αρχείο, ενώ αυτό υπάρχει στο δίκτυο. Επίσης, με τη stabilize το δίκτυο μαθαίνει για τους νέους κόμβους που συνδέονται. Κάθε κόμβος καλεί περιοδικά τη stabilize. Έστω ότι ο n καλεί την stabilize. Γίνονται τα εξής: ρωτά τον successor του n για τον successor του προηγούμενού του, έστω p, και αποφασίζει αν ο p πρέπει να είναι ο successor του n. Αυτό θα συνέβαινε αν ο p είχε μόλις συνδεθεί (join). Η stabilize ενημερώνει επίσης τον successor του n για την ύπαρξη του n, δίνοντάς του τη δυνατότητα να αλλάξει τον προηγούμενό του σε n. Ο successor το κάνει αυτό μόνο αν αυτός είναι ο κοντινότερος προηγούμενος που γνωρίζει. Σε περίπτωση που δύο κόμβοι πιστεύουν ότι έχουν τον ίδιο successor, s, κάθε ένας από αυτούς θα ενημερώσει σχετικά τον s, και αυτός θα κρίνει ποιος από τους δύο είναι πιο κοντα για να γίνει ο successor του. Αυτός που βρισκόταν πιο μακρυά θα μάθει για το ποιος είναι τελικά ο successor του s, όταν θα επικοινωνήσει με τον s. Επίσης, καλείται και η fix_ fingers(), η οποία για τους νέους κόμβους αρχικοποιεί τον πίνακα δρομολόγησης, και για τους υπάρχοντες, φροντίζει να συμπεριλάβουν και τους καινούργιους κόμβους στους πίνακες τους. Τέλος, η stabilize ενημερώνει τη λίστα με τους successors του κόμβου. Έτσι, αν ένας κόμβος n αντιληφθεί ότι ο successor του αποσυνδέθηκε μπορεί να τον αντικαταστήσει με τον πρώτο successor από αυτή τη λίστα με τους successors. Σταδιακά, η stabilize θα διορθώνει και τους πίνακες δρομολόγησης, αλλά και τις λίστες με τους successors, ώστε να δίνουν σωστές πληροφορίες.

- ✓ **Παράδειγμα Stabilize**

  Έστω ότι ο n συνδέεται με το δίκτυο και το ID του κυμαίνεται από n_p έως n_s. Ο n θα θεωρήσει τον n_s για successor του. Στη συνέχεια ο n_s θα ειδοποιηθεί από τον n, οπότε θα κάνει τον n προηγούμενό του. Όταν ο n_p τρέξει αργότερα τη

stabilize, θα ρωτήσει τον n_s για τον προηγούμενό του (που τώρα είναι ο n). Ο n_p θα θεωρήσει τον n σαν successor του. Τέλος, ο n_p θα ενημερώσει τον n ότι αυτός είναι τώρα ο προηγούμενός του.

Ακολουθεί ο ψευδοκώδικας της stabilize

```
// periodically verify n's immediate successor,
// and tell the successor about n.

n.stabilize()
        x = successor. predecessor;
        if (x ∈ (n, successor))
                successor = x;
        successor. notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
        if (predecessor is nil or n' ∈ (predecessor, n))
                predecessor = n';

// periodically refresh finger table entries.
n.fix_ fingers()
        i = random index > 1 into finger[];
        finger[i].node = find_ successor(finger[i].start);
```

Τέλος, ο χρόνος που απαιτείται για την ανακάλυψη του successor ενός κόμβου αλλά και για την ενημέρωση του πίνακα δρομολόγησής του είναι $O(\log^2 n)$.

Συνολική Αποτίμηση

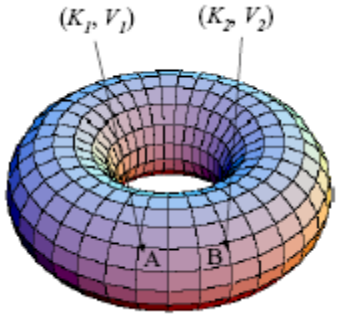
Το μοντέλο που παρέχεται από το Chord είναι ανθεκτικό στις αποτυχίες, καθώς αρκεί ένας μόνο κόμβος από τους r που αποθηκεύουν κάθε κλειδί να είναι διαθέσιμος και ενεργός. Επίσης, αν υποθέσουμε ότι έχουμε ένα δίκτυο με n κόμβους, εγγυάται ότι αν υπάρχει η τιμή που ψάχνουμε θα την εντοπίσει και μάλιστα σε χρόνο O(logn). Η εισαγωγή (Join) και η έξοδος (Leave) ενός κόμβου γίνεται με ανταλλαγή $O(\log^2 n)$ μηνυμάτων. Επιπλέον, επιτυγχάνει εξισορρόπηση φορτίου, καθώς για την ανάθεση των κλειδιών στους κόμβους χρησιμοποιείται ένας consistent hashing αλγόριθμος, ενώ όταν ένας κόμβος συνδέεται ή αποσυνδέεται από το δίκτυο, μόνο ένας μικρός αριθμός ζευγαριών (κλειδί, τιμή) πρέπει να μετακινηθεί. Ακόμα, το Chord έχει μεγάλο scalability και μπορεί να υποστηρίξει εκατομμύρια κόμβους. Τέλος, η τοπολογία του δικτύου που σχηματίζεται χαρακτηρίζεται ως τοπολογία πλέγματος (grid).

CAN

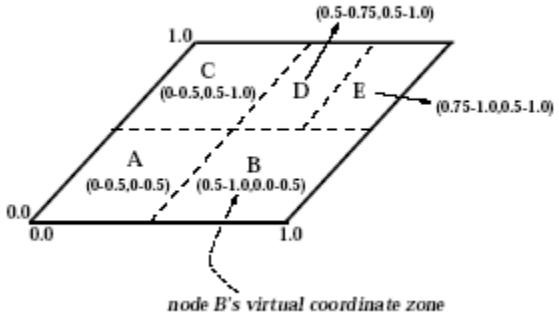
Στο Content - Addressable Network CAN [9] ορίζεται ένας d–διάστατος χώρος συντεταγμένων ο οποίος «διπλώνει» σχηματίζοντας ένα d- torus (Εικόνα 10). Ο χώρος

αυτός χωρίζεται σε ζώνες οι οποίες μοιράζονται δυναμικά σε όλους τους κόμβους που συνδέονται στο δίκτυο. Δηλαδή, σε κάθε κόμβο αντιστοιχεί και μία περιοχή του χώρου αυτού.

Για να αποθηκευτεί σε έναν κόμβο ένα ζευγάρι της μορφής (κλειδί, τιμή), υπολογίζεται το σημείο P= Hash(κλειδί). Το σημείο αυτό θα ανήκει σε μια ζώνη D του χώρου συντεταγμένων. Τότε, το ζευγάρι (κλειδί, τιμή) θα αποθηκευτεί στον κόμβο που αντιστοιχεί στην ζώνη αυτή. Στην Εικόνα 11 φαίνεται ένας δισδιάστατος χώρος [0 ,1] x [0, 1]. ο οποίος έχει χωριστεί σε 5 ζώνες καθώς στο δίκτυο είναι συνδεδεμένοι 5 κόμβοι. Επίσης φαίνονται και οι συντεταγμένες κάθε ζώνης. Για λόγους απλότητας ο χώρος φαίνεται επίπεδος και όχι όπως είναι πραγματικά.



Εικόνα 10: Torus - ο χώρος συντεταγμένων [9]



Εικόνα 11: Δισδιάστατος χώρος χωρισμένος σε 5 ζώνες [9]

Ο πίνακας δρομολόγησης, που δημιουργεί και διατηρεί κάθε κόμβος, κρατάει τις συντεταγμένες της ζώνης του, τις IP διευθύνσεις των κόμβων με τις ζώνες των οποίων συνορεύει η δική του ζώνη, καθώς και τις συντεταγμένες των γειτονοκών του ζωνών. Εφόσον ο πίνακας δρομολόγησης κάθε κόμβου περιλαμβάνει πληροφορίες μόνο για τους γειτονικούς κόμβους, η διατήρησή τους απαιτεί μόνο O(d).

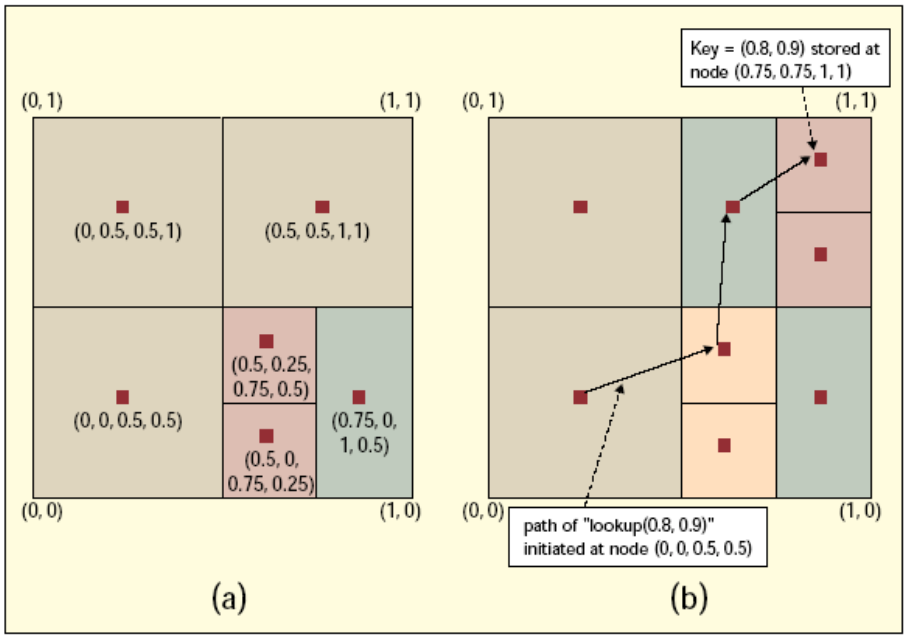
Ακολουθεί η περιγραφή των βασικών λειτουργιών για το CAN. Συνοπτικά έχουμε τις εξής λειτουργίες:

Πίνακας 2: Βασικές λειτουργίες CAN

Λειτουργία	***Περιγραφή***
Bootstrap	Σύνδεση κόμβου με το δίκτυο
Join	Ανάθεση ζώνης
Έξοδος κόμβου	Αποσύνδεση κόμβου – συγχώνευση ζώνης
Lookup(key, value)	Αναζήτηση του (key, value)
Stabilize	Περιοδική ενημέρωση γειτονικών ζωνών

Αναλυτικότερα:
Lookup: Η αναζήτηση κάποιας πληροφορίας που αντιστοιχεί στο σημειο P γίνεται ως εξής: αν το P ανήκει στην ζώνη του τρέχοντος κόμβου, τότε επιστρέφεται το (κλειδί, τιμή), ή λάθος, σε περίτωση που δεν υπάρχει. Διαφορετικά, η αίτηση προωθείται στη γειτονική ζώνη της οποίας οι συντεταγμένες είναι πιο κοντά στις συντεταγμένες του P. Το μέσο μήκος μονοπατιού αναζήτησης είναι $dn^{1/d}$ hops, δηλαδή είναι $O(n^{1/d})$, όπου n το πλήθος των κόμβων και d το πλήθος των γειτόνων. Στην Εικόνα 12b φαίνεται η αναζήτηση για το κλειδί που αντιστοιχεί στο σημείο (0,8, 0.9), δηλαδή lookup(0.8, 0.9), που ξεκινά από τον κόμβο με συντεταγμένες (0.0, 0.5, 0.5).

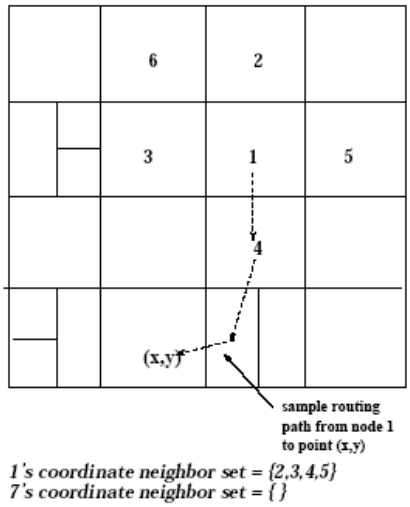
Σημειώνουμε ότι υπάρχουν πολλά εναλλακτικά μονοπάτια μεταξύ δύο σημείων, οπότε σε περίπτωση που αποσυνδεθεί ο κοντινότερος γειτονικός κόμβος, ακολουθείται το αμέσως κοντινότερο μονοπάτι.

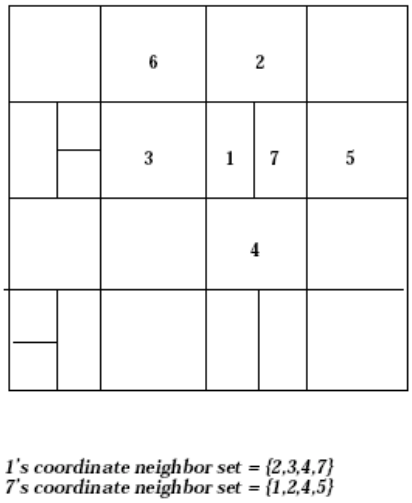


Εικόνα 12α: Δισδιάστατος χώρος αναζήτησης **Εικόνα12b: lookup(0.8, 0.9) [9]**

Bootstrap- Join: Στην περίπτωση του CAN, οι δύο αυτές λειτουργίες θεωρούμε ότι γίνονται μαζί. Συγκεκριμένα, για να συνδεθεί ένας κόμβος στο CAN πρέπει να γνωρίζει την IP διεύθυνση ενός κόμβου που είναι ήδη συνδεδεμένος στο δίκτυο. Οι διευθύνσεις αυτές μπορούν για παράδειγμα σε ένα site. Έστω N1 ο κόμβος που πρόκειται να εισαχθεί και έστω ότι γνωρίζει μια τέτοια διεύθυνση. Ονομάζουμε τον γνωστό κόμβο που ανήκει ήδη στο CAN N2. Διαλέγουμε τυχαία ένα σημείο P στο χώρο, που ανήκει στην ζώνη του

N3. Ο N2 στέλνει μία αίτηση JOIN για το P. Ο N3 διαιρεί την ζώνη του στα δύο και στέλνει τα ζευγάρια (κλειδί, τιμή) που αντιστοιχούν στο ένα από τα δύο τμήματα στον N1. Ο N1 ενημερώνεται για το ποιες είναι οι γειτονικές του ζώνες από τον N3. Επίσης, ο N3 ενημερώνει τους υπόλοιπους γείτονες για τις αλλαγές. Η εισαγωγή ενός κόμβου επηρεάζει μονο O(d) κόμβους. Στην Εικόνα 13α, φαίνεται ο χώρος συντεταγμένων πριν την εισαγωγή του κόμβου 7. Το σύνολο των γειτονικών κόμβων του 1 είναι το {2, 3, 4, 5} και του 7 το {}. Στην Εικόνα 13b, φείνεται ο χώρος μετά την είσοδο του 7. Το σύνολο των γειτονικών κόμβων του 1 είναι τώρα το {2, 3, 4, 7} και του 7 το {1, 2, 4, 5}.





Εικόνα 13α: Πριν την εισαγωγή του κόμβου 7 **Εικόνα 13b: Μετά την εισαγωγή του κόμβου 7 [9]**

Έξοδος Κόμβου (Explicit departure): Ο κόμβος που αποχωρεί παραδίδει την ζώνη του σε κάποιον άλλο με την προϋπόθεση ότι προκύπτει μία έγκυρη ζώνη, ή συγχωνεύεται με μία μικρότερη.

Stabilize: Περιοδικά, μεταξύ των γειτονικών κόμβων, ανταλλάσσονται ενημερωτικά (update) μηνύματα. Σε περίπτωση που ένας κόμβος σταματήσει να λαμβάνει τα μηνύματα αυτά από κάποιο γείτονά του, ένας μηχανισμός (TAKEOVER) αναλαμβάνει να συγχωνεύσει τη ζώνη με κάποια μικρότερη. Έστω, για παράδειγμα, ότι ο K αποσυνδέθηκε. Τότε, μόλις οι γειτονικοί του κόμβοι το καταλάβουν, ξεκινούν από έναν timer. Οι timers αυτοί είναι ανεξάρτητοι μεταξύ τους και αρχικοποιούνται σε μια τιμή ανάλογη προς το μέγεθος της ζώνης τους. Ο πρώτος timer που λήγει στέλνει ένα TAKEOVER μήνυμα. Όταν ένας κόμβος λάβει ένα τέτοιο μήνυμα ακυρώνει τον δικό του timer, ή αν η δική του ζώνη είναι μικρότερη, απαντάει με το δικό του TAKEOVER. Με αυτόν τον τρόπο επιλέγεται ο κόμβος με τη μικρότερη ζώνη.

Ο μηχανισμός που περιγράφηκε παραπάνω μπορεί να αναθέσει σε έναν κόμβο περισσότερες από μία ζώνες. Ιδανικά, επιδιώκεται μία 1-1 αντιστοιχία κόμβων και ζωνών έτσι ώστε να περιορίζεται κατα το δυνατόν ο κατακερματισμός του χώρου συντεταγμένων. Αυτό προσπαθεί να επιτύχει ένας μηχανισμός επανανάθεσης (reassignment) που δρα παράλληλα. Ο μηχανισμός αυτός λειτουργεί σε γενικές γραμμές ως εξής: Έστω ότι έχουμε τον κόμβο I. Τότε γίνονται τα εξής:

1) Έστω d_k η διάσταση/συντεταγμένη κατά την οποία έγινε η τελευταία διχοτόμηση της ζώνης του I.

2) Ο Ι βρίσκει από τον πίνακα δρομολόγησής του έναν γειτονικό κόμβο ως προς τη συντεταγμένη/ διάσταση d_k, έστω J, η ζώνη του οποίου είναι το άλλο μισό της ζώνης του Ι πριν από αυτή την τελευταία διχοτόμηση.
3) Αν το μέγεθος της ζώνης του Ι ισούται με αυτό του J, τότε οι Ι και J είναι ένα ζευγάρι «δίδυμων κόμβων» (sibling nodes) και οι ζώνες τους μπορούν να συνδυαστούν.
4) Αν το μέγεθος της ζώνης του J είναι μικρότερο από αυτό του Ι, τότε ο J ξεκινάει να κάνει την ίδια διαδικασία.
5) Η διαδικασία αυτή συνεχίζεται μέχρι να βρεθεί ένα ζευγάρι δίδυμων κόμβων.

Προτεινόμενες Βελτιώσεις
Προτείνονται διάφοροι τρόποι για τη βελτίωση της απόδοσης του CAN, και ως προς το μήκος του μονοπατιού αναζήτησης, και ως προς τη βελτίωση της διαθεσιμότητας των δεδομένων. Για παράδειγμα, με την θεώρηση ενός πολυδιάστατου χώρου συντεταγμένων, αποδεικνύεται ότι μειώνεται το μήκος του μονοπατιού αναζήτησης χωρίς να επιβαρύνονται σημαντικά οι πίνακες δρομολόγησης. Για την αύξηση της διαθεσιμότητας των δεδομένων θεωρούμε ότι υπάρχουν πολλοί ανεξάρτητοι χώροι συντεταγμένων (realities), οπότε ένα ζευγάρι (κλειδί, τιμή) αποθηκεύεται σε περισσότερους από έναν κόμβους.

Συνολική αποτίμηση
Το CAN εγγυάται ότι αν υπάρχει η τιμή που ψάχνουμε θα την εντοπίσει σε χρόνο $O(n^{1/d})$. Ο χρόνος αυτός μπορεί να μειωθεί περεταίρω αν θέσουμε d=logn/2 οπότε γίνεται O(logn) όπου n ο αριθμός των κόμβων στο δίκτυο. Επιπλέον, καθώς χρησιμοποιεί hashing για την ανάθεση των ζευγαριών (κλειδί, τιμή) στους κόμβους, επιτυγχάνει εξισορρόπηση φορτίου ενώ με κάποιες βελτιώσεις επιτυγχάνει και μεγάλη διαθεσιμότητα δεδομένων. Η εισαγωγή και η έξοδος ενός κόμβου γίνεται σε σταθερό χρόνο, ενώ οι πίνακες δρομολόγησης κρατούν πληροφορίες μόνο για τους d γειτονικούς κόμβους. Τέλος, είναι ανθεκτικό σε επιθέσεις και έχει μεγάλο scalability, καθώς μπορεί να υποστηρίξει τη σύνδεση εκατομμυρίων κόμβων και η τοπολογία του δικτύου που σχηματίζεται είναι τοπολογία πλέγματος.

Συγκριτική Παρουσίαση

Στους παρακάτω πίνακες, παρουσιάζονται συγκριτικά διάφορα χαρακτηριστικά των αλγορίθμων Gnutella, Gnutella2, Chord και CAN.

Πίνακας 3: Χαρακτηριστικά P2P Συστημάτων

	Gnutella	**Gnutella2**	**Chord**	**CAN**
Προσαρμοστικότητα	✓	✓	✓	✓
Εγγύηση εύρεσης δεδομένων	-	-	✓	✓
Scalability	μέτριο	καλό	πολύ καλό	πολύ καλό
Εξισορρόπηση Φορτίου	-	-	✓	✓
Ανθεκτικότητα σε Denial of Service (DoS)	-	-	✓	✓

Όπως φαίνεται στον Πίνακα 3 και οι τέσσερεις αλγόριθμοι παρουσιάζουν μεγάλη προσαρμοστικότητα στις αλλαγές του δικτύου. Ωστόσο, οι αλγόριθμοι Chord και CAN, σε αντίθεση με τους αλγόριθμους Gnutella και Gnutella2, εγγυώνται την εύρεση των δεδομένων, την εξισορρόπηση του φορτίου του P2P δικτύου, ενώ παρουσιάζουν και ανθεκτικότητα σε επιθέσεις DoS. Τέλος, το Chord και το CAN μπορούν να υποστηρίξουν πολλούς κόμβους, σε αντίθεση με το Gnutella2 που έχει πιο περιορισμένες δυνατότητες, και το Gnutella που έχει ακόμα λιγότερες.

Πίνακας 4: Χαρακτηριστικά P2P Συστημάτων

	Gnutella	**Gnutella2**	**Chord**	**CAN**
Τοπολογία	power law	power law	grid (δακτύλιος)	grid (torus)
Αποθήκευση δεδομένων	Αυθαίρετη	αυθαίρετη	hashing	hashing
Δρομολόγηση μηνυμάτων	Broadcasting	περιορισμένο broadcasting	απευθείας	απευθείας

Στον Πίνακα 4 οι τέσσερις αλγόριθμοι συγκρίνονται ως προς την τοπολογία του P2P δικτύου που σχηματίζεται, καθώς και τους μηχανισμούς αποθήκευσης και δρομολόγησης που χρησιμοποιούν. Στο Gnutella και στο Gnutella2 η τοπολογία που παρατηρείται παρουσιάζει power law χαρακτηριστικά, ενώ στο Chord και στο CAN είναι grid. Ειδικότερα στο Chord σχηματίζεται ένας δακτύλιος και στο CAN ένα torus. Η αποθήκευση των δεδομένων στο Gnutella και στο Gnutella2 χαρακτηρίζεται στον πίνακα ως «αυθαίρετη» [16]. Δηλαδή, δε γίνεται με κάποιο μηχανισμό αλλά κάθε κόμβος αποθηκεύει τα δικά του δεδομένα. Αντίθετα, στο Chord και στο CAN η αποθήκευση γίνεται με κατακερματισμό και κάθε κόμβος αποθηκεύει κάποια δεδομένα, όχι κατ' ανάγκη αυτά που μοιράζεται. Τέλος, η δρομολόγηση των μηνυμάτων στο Gnutella γίνεται με broadcasting και στο Gnutella2 με περιορισμένο broadcasting (στέλνονται μόνο στους κόμβους που ανήκουν στο ίδιο hub cluster). Στο Chord και στο CAN ένα μήνυμα στέλνεται απευθείας ακολουθώντας κάποιο μονοπάτι, όπως αυτό προκύπτει από τους πίνακες δρομολόγησης.

Πίνακας 5: Πολυπλοκότητες Βασικών Λειτουργιών

	Gnutella	**Gnutella2**	**Chord**	**CAN**
Stabilize	O(n)	O(1)	$O(\log^2 n)$	O(1)
Lookup	O(n)	$O(\sqrt{n})$	O(logn)	$O(n^{1/d})$
Join	O(n)	O(1)	$O(\log^2 n)$	O(1)

Στον Πίνακα 5 παρουσιάζονται οι πολυπλοκότητες των βασικών λειτουργιών των τεσσάρων αλγορίθμων, θεωρώντας ότι έχουμε ένα δίκτυο με n κόμβους. Το Gnutella2 και το CAN έχουν την πιο «απλή» Stabilize καθώς αυτή πραγματοποιείται σε σταθερό χρόνο. Τη μεγαλύτερη πολυπλοκότητα παρουσιάζει η Stabilize του Gnutella καθώς, στη χειρότερη περίπτωση, θα πρέπει να στείλει O(n) μηνύματα. Όσα ισχύουν για τη Stabilize ισχύουν και για τη Join. Την πιο αποτελεσματική Lookup έχει το Chord. Ωστόσο, η αναζήτηση του CAN, με κάποιες βελτιώσεις-επεκτάσεις μπορεί να γίνει εξίσου αποτελεσματική, δηλαδή O(logn).

2 Προσομοίωση

2.1 Συστήματα και Μοντέλα

Ένα *σύστημα* ορίζεται ως μία συλλογή οντοτήτων που ενεργούν και αλληλεπιδρούν, με στόχο κάποιο λογικό τερματισμό. Στην πράξη, η έννοια του συστήματος εξαρτάται από τους στόχους της εκάστοτε μελέτης [15].

Ορίζουμε την *κατάσταση* ενός συστήματος ως τη συλλογή των μεταβλητών που είναι απαραίτητες για την περιγραφή του σε μια χρονική στιγμή, σε σχέση πάντοτε με το περιβάλλον – στόχους του.

Τα συστήματα μπορούν να είναι *διακριτά* (discrete) ή *συνεχή* (continuous). Διακριτό σύστημα είναι αυτό στο οποίο οι μεταβλητές κατάστασης μπορούν να αλλάξουν σε διακεκριμένες στιγμές του χρόνου. Συνεχές είναι το σύστημα του οποίου οι μεταβλητές κατάστασης αλλάζουν συνεχώς στο χρόνο.

Λίγα συστήματα στην πράξη είναι εξ ολοκλήρου διακριτά ή συνεχή, αλλά επειδή συνήθως μία από τις δύο ιδιότητες κυριαρχεί, μπορούμε τις περισσότερες φορές να χαρακτηρίσουμε το σύστημα ως διακριτό ή συνεχές.

Κατά τη διάρκεια της ζωής ενός συστήματος, παρουσιάζεται η ανάγκη μελέτης του, ώστε να αποκτήσουμε γνώση για τις σχέσεις μεταξύ των διαφόρων τμημάτων του, ή για να προβλέψουμε την απόδοσή του κάτω από νέες ή ειδικές συνθήκες. Παρακάτω, παρουσιάζονται διάφοροι τρόποι με τους οποίους μπορεί να μελετηθεί ένα σύστημα.

- ***Πείραμα με το Πραγματικό Σύστημα ή Πείραμα με Μοντέλο του Συστήματος***: Σε αυτή την περίπτωση, απαιτείται να έχουμε φυσική πρόσβαση στο σύστημα. Αυτό ενδέχεται να είναι δαπανηρό ή ακόμα και μή εφικτό στην περίπτωση στην οποία το σύστημα δεν έχει ακόμα ολοκληρωθεί. Ωστόσο, στην περίπτωση που είναι δυνατή η εφαρμογή του τρόπου αυτού, τα αποτελέσματα είναι ακριβή και δεν μπορούν να αμφισβητηθούν.
- ***Φυσικό Μοντέλο ή Μαθηματικό Μοντέλο***: Ένα μαθηματικό μοντέλο αναπαριστά το σύστημα με βάση λογικές και ποσοτικές σχέσεις, οι οποίες χειριζόμενες και μεταβαλλόμενες κατάλληλα, επιτρέπουν να δούμε πώς αντιδρά το μοντέλο, και κατά συνέπεια, πώς θα αντιδρούσε το σύστημα, υπό την προϋπόθεση πάντα ότι το μαθηματικό μοντέλο είναι έγκυρο.
- ***Αναλυτική Λύση ή Προσομοίωση***: Από τη στιγμή που έχει δημιουργηθεί ένα μαθηματικό μοντέλο, θα πρέπει να εξετασθεί ο τρόπος χρήσης του, ώστε να μπορεί να απαντήσει στα ερωτήματα που μας ενδιαφέρουν για το σύστημα, που υποτίθεται ότι αντιπροσωπεύει. Αν το μοντέλο είναι αρκετά απλό, τα ερωτήματα αυτά μπορούν σχετικά εύκολα να απαντηθούν. Στις περισσότερες περιπτώσεις, όμως, η διαδικασία αυτή είναι αρκετά πολύπλοκη και απαιτεί μεγάλη υπολογιστική ισχύ. Όταν υπάρχει αναλυτική λύση για ένα μαθηματικό μοντέλο και είναι υπολογιστικά εφικτή, συνήθως την προτιμούμε από την προσομοίωση. Τα περισσότερα συστήματα όμως είναι πολύπλοκα και τα μαθηματικά μοντέλα που τα αναπαριστούν δεν έχουν αναλυτική λύση. Στην περίπτωση αυτή το μοντέλο πρέπει να μελετηθεί με τη χρήση προσομοίωσης, δηλαδή με την εκτέλεση αριθμητικών πειραμάτων στο μοντέλο για τις εισόδους (δεδομένα) που μας ενδιαφέρουν, για να δούμε πώς αυτά επηρεάζουν τις εξόδους (μέτρα απόδοσης) του συστήματος. Η προσομοίωση είναι επίσης πολύ χρήσιμη στις περιπτώσεις που μπορούμε να πάρουμε μόνο προσεγγιστική αναλυτική

λύση, οπότε θέλουμε να επιβεβαιώσουμε την προσέγγιση, να βρούμε το σχετικό λάθος κ.λ.π.[15]

2.2 Τα Είδη Μοντέλων Προσομοίωσης.

Τα Μοντέλα Προσομοίωσης μπορούν να κατηγοριοποιηθούν με βάση τέσσερις διαφορετικές έννοιες:

Στατικά ή Δυναμικά Μοντέλα Προσομοίωσης: Ένα στατικό μοντέλο προσομοίωσης, αναπαριστά ένα σύστημα σε μία συγκεκριμένη χρονική στιγμή, ή αναπαριστά ένα σύστημα στο οποίο ο χρόνος δεν έχει σημασία. Αντίθετα, ένα δυναμικό μοντέλο προσομοίωσης αναπαριστά ένα σύστημα, όπως αυτό εξελίσσεται με την πάροδο του χρόνου.

Ντετερμινιστικά ή Στοχαστικά Μοντέλα Προσομοίωσης: Αν ένα μοντέλο προσομοίωσης δεν περιλαμβάνει στοχαστικά (δηλαδή "τυχαία") τμήματα, ονομάζεται ντετερμινιστικό. Στα ντετερμινιστικά μοντέλα, για δεδομένη είσοδο, η έξοδος είναι καθορισμένη. Όμως, πολλά συστήματα πρέπει να χρησιμοποιήσουν στοχαστικά μοντέλα προσομοίωσης, δηλαδή μοντέλα που θα έχουν τουλάχιστον ορισμένα τμήματα με "τυχαία" είσοδο.

Αυτο-οδηγούμενα ή Ιχνο-οδηγούμενα Μοντέλα Προσομοίωσης: Σε ένα αυτο-οδηγούμενο (self-driven) μοντέλο, υπάρχει μία εσωτερική πηγή τυχαίων αριθμών. Οι τυχαίοι αριθμοί οδηγούν τα τμήματα του μοντέλου, δηλαδή χρησιμοποιούνται για τον προσδιορισμό των στιγμών εμφανίσεως των γεγονότων του συστήματος. Το βασικό χαρακτηριστικό του αυτο-οδηγούμενου μοντέλου είναι ότι αποτελεί ένα αυτάρκες μοντέλο το οποίο δεν χρειάζεται εξωτερικές εισόδους για να λειτουργήσει. Αντίθετα, ένα ιχνο-οδηγούμενο (trace-driven) μοντέλο καθοδηγείται από ακολουθίες εισόδου που προέρχονται από δεδομένα (trace data) που έχουν δημιουργηθεί από τη λειτουργία ενός πραγματικού συστήματος. Τέτοια δεδομένα μπορούν να παραχθούν στα περισσότερα υπολογιστικά συστήματα που διαθέτουν ενσωματωμένα προγράμματα ιχνηλάτησης (tracing programs) που παρακολουθούν και καταγράφουν τις δραστηριότητες του συστήματος. Τα ιχνο-οδηγούμενα μοντέλα έχουν ορισμένα πλεονεκτήματα, όπως το γεγονός ότι αποφεύγονται οι δυσκολίες της πιθανοτικής ανάλυσης που χρειάζεται για τη χρήση κατανομών στην περιγραφή των εισόδων του μοντέλου, και επίσης το γεγονός ότι τα μοντέλα αυτά είναι εύκολο να επιβεβαιωθούν. Το πρόβλημα με τα ιχνο-οδηγούμενα μοντέλα είναι το μικρό εύρος εφαρμογών που μπορούν να αντιμετωπίσουν. Οι εφαρμογές αυτές πρακτικά περιορίζονται σε υπολογιστικά συστήματα και μάλιστα μόνο για τη μελέτη μετατροπών σε ένα σύστημα που ήδη λειτουργεί [15].

Συνεχή ή Διακριτά Μοντέλα Προσομοίωσης: Οι ορισμοί των συνεχών και διακριτών μοντέλων προσομοίωσης, είναι ανάλογοι με τους ορισμούς των συνεχών και διακριτών συστημάτων που αναφέρθηκαν στην ενότητα 2.1. Πάντως, πρέπει να σημειωθεί ότι ένα διακριτό μοντέλο δεν χρησιμοποιείται μόνο για την αναπαράσταση ενός διακριτού συστήματος, και ένα διακριτό σύστημα δεν αναπαρίσταται μόνο από ένα διακριτό μοντέλο προσομοίωσης. Η απόφαση για τη χρήση ενός διακριτού ή ενός συνεχούς μοντέλου για ένα συγκεκριμένο σύστημα, εξαρτάται από τους ιδιαίτερους στόχους της μελέτης.

Τα μοντέλα προσομοίωσης που θα μας απασχολήσουν στη συνέχεια, θα είναι διακριτά, δυναμικά, στοχαστικά και αυτο-οδηγούμενα και θα ονομάζονται Μοντέλα Προσομοίωσης Διακριτών Γεγονότων (discrete event simulation models).

2.3 Ο Μηχανισμός Εξέλιξης του Χρόνου

Λόγω του δυναμικού χαρακτήρα των μοντέλων προσομοίωσης διακριτών γεγονότων, πρέπει να έχουμε τη δυνατότητα αποθήκευσης της τρέχουσας τιμής του προσομοιωμένου χρόνου, ενώ χρειαζόμαστε και ένα μηχανισμό αύξησής του από μία τιμή σε μία άλλη. Η μεταβλητή του μοντέλου προσομοίωσης που μας δίνει την τρέχουσα τιμή του χρόνου, ονομάζεται *ρολόι προσομοίωσης* (simulation clock). Έχουν επικρατήσει δύο βασικές μέθοδοι για την εξέλιξη του ρολογιού προσομοίωσης: Η *Εξέλιξη με βάση το Χρόνο του Επομένου Γεγονότος* (next-event time advance) και η *Εξέλιξη Σταθερής Αύξησης του Χρόνου* (fixed-increment time advance).

- ***Εξέλιξη με βάση το Χρόνο του Επομένου Γεγονότος***: Στη μέθοδο εξέλιξης με βάση το χρόνο του επομένου γεγονότος, το ρολόι προσομοίωσης αρχικοποιείται στο μηδέν, ενώ καθορίζονται και οι στιγμές εμφάνισης των μελλοντικών γεγονότων. Το ρολόι αυξάνει στο χρόνο εμφάνισης του πιο κοντινού στο μέλλον, από τα γεγονότα αυτά. Τη στιγμή αυτή η κατάσταση του συστήματος ενημερώνεται ώστε να λάβει υπόψη της το γεγονός που εμφανίστηκε, ενώ ενημερώνεται επίσης η γνώση μας για τις χρονικές στιγμές εμφάνισης των μελλοντικών γεγονότων. Στη συνέχεια, το ρολόι αυξάνει, ώστε να δείχνει τη στιγμή εμφάνισης του νέου, πιο κοντινού στο μέλλον γεγονότος, η κατάσταση του συστήματος ενημερώνεται, καθορίζονται οι χρονικές στιγμές εμφάνισης των μελλοντικών γεγονότων κ.ο.κ. Η διαδικασία αυτή εξέλιξης του ρολογιού προσομοίωσης από το ένα γεγονός στο άλλο, συνεχίζεται μέχρι να ικανοποιηθεί κάποια προκαθορισμένη συνθήκη τερματισμού της προσομοίωσης. Αφού όλες οι αλλαγές κατάστασης γίνονται μόνο στις χρονικές στιγμές εμφάνισης των γεγονότων, οι ενδιάμεσες ανενεργοί περίοδοι δεν λαμβάνονται υπόψη και το ρολόι μετακινείται αυτόματα στη στιγμή εμφάνισης του επομένου γεγονότος.
- ***Εξέλιξη Σταθερής Αύξησης του Χρόνου:*** Στη μέθοδο εξέλιξης σταθερής αύξησης του χρόνου, το ρολόι προσομοίωσης εξελίσσεται με σταθερές αυξήσεις ακριβώς Δt μονάδων χρόνου κάθε φορά. Μετά από κάθε ενημέρωση του ρολογιού, γίνεται ένας έλεγχος για να εξακριβωθεί εάν θα έπρεπε να έχουν εμφανισθεί κάποια γεγονότα κατά το προηγούμενο χρονικό διάστημα Δt. Αν εμφανίσθηκαν γεγονότα στο διάστημα αυτό, θεωρούμε ότι αυτά εμφανίζονται στο τέλος του χρονικού διαστήματος και η κατάσταση του συστήματος ενημερώνεται κατάλληλα.

2.4 Συστατικά ενός Μοντέλου Προσομοίωσης Διακριτών Γεγονότων

Τα περισσότερα μοντέλα προσομοίωσης διακριτών γεγονότων που χρησιμοποιούν τη μέθοδο εξέλιξης με βάση το χρόνο του επομένου γεγονότος, περιλαμβάνουν τα παρακάτω τμήματα:

- ***Κατάσταση Συστήματος*** (system state): Η συλλογή των μεταβλητών κατάστασης που είναι απαραίτητες για την περιγραφή του συστήματος σε μία χρονική στιγμή.
- ***Ρολόι Προσομοίωσης*** (simulation clock): Μία μεταβλητή που περιέχει την τρέχουσα τιμή του προσομοιωμένου χρόνου.
- ***Λίστα Γεγονότων*** (event list): Μία λίστα που περιέχει την επόμενη χρονική στιγμή εμφάνισης κάθε τύπου γεγονότος.
- ***Μετρητές Στατιστικών*** (statistical counters): Μεταβλητές που χρησιμοποιούνται για την αποθήκευση στατιστικών μετρήσεων της απόδοσης του συστήματος.
- ***Ρουτίνα Αρχικοποίησης*** (initialization routine): Ένα υποπρόγραμμα που αρχικοποιεί το μοντέλο προσομοίωσης τη χρονική στιγμή μηδέν.
- ***Ρουτίνα Χρονισμού*** (timing routine): Ένα υποπρόγραμμα που αναγνωρίζει το επόμενο γεγονός από τη λίστα γεγονότων και ακολούθως αυξάνει το ρολόι προσομοίωσης στη χρονική στιγμή που το γεγονός αυτό θα εμφανισθεί.
- ***Ρουτίνες Γεγονότων*** (event routines): Υποπρογράμματα που ενημερώνουν την κατάσταση συστήματος όταν εμφανίζεται ένα συγκεκριμένο είδος γεγονότος (υπάρχει μία τέτοια ρουτίνα για κάθε είδος γεγονότος).
- ***Ρουτίνες Βιβλιοθήκης*** (library routines): Σύνολο υποπρογραμμάτων που δημιουργούν τυχαίες εμφανίσεις τιμών από πιθανοτικές κατανομές, που έχουν ορισθεί ως μέρος του μοντέλου προσομοίωσης.
- ***Γεννήτρια Αναφορών*** (report generator): Υποπρόγραμμα που υπολογίζει εκτιμήσεις των επιθυμητών μέτρων απόδοσης από τους μετρητές στατιστικών και παράγει αναφορές όταν τελειώσει η εκτέλεση του προσομοιωτή.
- ***Κυρίως Πρόγραμμα*** (main program): Το πρόγραμμα που καλεί τη ρουτίνα χρονισμού για να καθοριστεί το επόμενο γεγονός, και στη συνέχεια μεταφέρει τον έλεγχο στην αντίστοιχη ρουτίνα γεγονότος για να ενημερωθεί κατάλληλα η κατάσταση του συστήματος. Ελέγχει επίσης αν πρέπει να τερματιστεί η προσομοίωση και καλεί τότε τη γεννήτρια αναφορών.

2.5 Δημιουργία Τυχαίων Τιμών από Πιθανοτικές Κατανομές: Η κατανομή Poisson

Προκειμένου να υλοποιηθεί μια προσομοίωση με τη χρήση τυχαίων εισόδων (όπως οι χρόνοι μεταξύ διαδοχικών αφίξεων), πρέπει να καθορίσουμε τις πιθανοτικές κατανομές που τις περιγράφουν. Η προσομοίωση εξελίσσεται με τη δημιουργία και χρήση τυχαίων τιμών από τις κατανομές αυτές.

Στη συνέχεια θα ασχοληθούμε με την κατανομή Poisson. Η κατανομή Poisson χρησιμοποιείται στη μοντελοποίηση του αριθμού εμφανίσεων γεγονότων κατά τη διάρκεια μιας συγκεκριμένης περιόδου. Στην περίπτωσή μας, μας ενδιαφέρει να μοντελοποιήσουμε το ρυθμό υποβολής αιτήσεων αναζήτησης κατά τη διάρκεια ζωής ενός κόμβου. Αυτό το μοντελοποιούμε με τη βοήθεια της κατανομής Poisson [1].

Η κατανομή Poisson έχει συνάρτηση πυκνότητας πιθανότητας $f(x) = \lambda e^{-\lambda x}$. Σε χρονικό διάστημα T οι αφίξεις ακολουθούν την κατανομή Poisson και η πιθανότητα για k αφίξεις σε T (sec) δίνεται από τη σχέση

$$P(k) = P[k\_\alpha\phi\acute{\iota}\xi\varepsilon\iota\varsigma\_\sigma\varepsilon\_\mathrm{T}\sec] = \frac{(\lambda\mathrm{T})^k e^{-\lambda\mathrm{T}}}{k!}$$

Η συνάρτηση κατανομής της είναι $F(x) = P(X \le x) = \sum_{x_i \le x} p(x_i) = \sum_{x_i \le x} \frac{(\lambda\mathrm{T})^x e^{-\lambda\mathrm{T}}}{x!}$.

Αποδεικνύεται ότι όταν οι αφίξεις ακολουθούν την κατανομή Poisson, τα χρονικά διαστήματα μεταξύ αφίξεων είναι τυχαίες μεταβλητές που ακολουθούν εκθετική κατανομή. Η συνάρτηση πυκνότητας πιθανότητας της εκθετικής κατανομής δίνεται από τον τύπο $f(x) = \frac{1}{\beta} e^{-x/\beta}$ όπου β η μέση τιμή. Επομένως ο μέσος χρόνος μεταξύ διαδοχικών αφίξεων δίνεται από τη σχέση

$$E(\tau) = \int_0^\infty \tau f(\tau) d\tau = \frac{1}{\lambda}$$

Ο αλγόριθμος για τη δημιουργία τυχαίων τιμών από την κατανομή Poisson(λ), βασίζεται στη σχέση μεταξύ των κατανομών Poisson(λ) και expo(1/λ) και είναι ο εξής:

Θέσε $p = e^{-\lambda}, q = 1$, και $N = 0$.

1. Δημιούργησε ένα $U_{N+1} \sim U(0,1)$ και αντικατάστησε το q με το qU_{N+1}. Αν $q < p$, επίστρεψε το $X = N$ και τερμάτισε. Αλλιώς πήγαινε στο Βήμα 2.
2. Αντικατάστησε το i με N+1 και πήγαινε πίσω στο Βήμα 2.

Η ορθότητα του αλγορίθμου αποδεικνύεται, παρατηρώντας ότι $X = i$, αν και μόνο αν:

$$\sum_{j=1}^{i} Y_j \le 1 < \sum_{j=1}^{i+1} Y_j$$

όπου $Y_j = (-1/\lambda)\ln U_j \sim \exp(1/\lambda)$ και τα Y_j είναι ανεξάρτητα μεταξύ τους. Δηλαδή, $X = \max\{i : \sum_{j=1}^{i} Y_j \le 1\}$, έτσι ώστε η $X \sim \mathrm{Poisson}(\lambda)$.

3 Ο ορισμός του προβλήματος

Η ευρεία αποδοχή που γνωρίζει το P2P μοντέλο επικοινωνίας μέσω κυρίως των εφαρμογών διαμοιρασμού αρχείων, καθιστά απαραίτητη τη μελέτη των P2P πρωτοκόλλων.

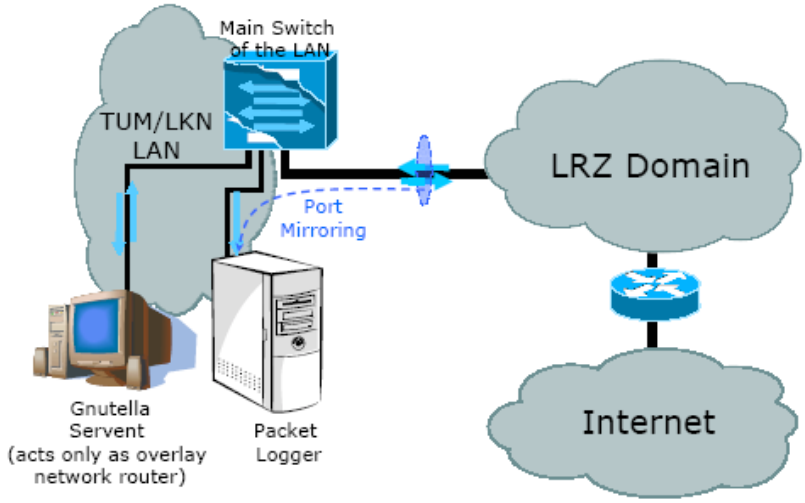
3.1 Προηγούμενες Μελέτες

Για το σκοπό αυτό έχουν εργαστεί πολλοί και με διάφορους τρόπους. Παρακάτω παρουσιάζονται επιγραμματικά κάποιες από αυτές τις μελέτες.

Στο [2] αντικείμενο μελέτης είναι το Napster και το Gnutella. Για κάθε ένα από τα δύο συστήματα δημιουργήθηκε ένας crawler. Σκοπός του είναι η ανακάλυψη όσο το δυνατόν περισσότερων συνδεδεμένων κόμβων και η συγκέντρωση στοιχείων για αυτούς. Για το Napster, επειδή δεν είναι δυνατή η απευθείας πρόσβαση στους κεντρικούς servers για τη συλλογή στοιχείων των συνδεδεμένων χρηστών, ο crawler υποβάλλει αιτήσεις για διάφορα δημοφιλή αρχεία. Για κάθε κόμβο που απαντά, ρωτάει τον κεντρικό server προκειμένου να πάρει στοιχεία, όπως το bandwidth, τον αριθμό, τα ονόματα και το μέγεθος των αρχείων που μοιράζεται, τον αριθμό των αρχείων που κατεβάζει (download) ή που κάποιος άλλος κατεβάζει από αυτόν (upload) εκείνη τη στιγμή, την IP διεύθυνση και άλλα. Για το Gnutella, ο crawler εκμεταλλεύεται τα ping-pong μηνύματα. Αρχικά, συνδέεται με έναν γνωστό κόμβο και αρχίζει να στέλνει ping μηνύματα με μεγάλο TTL σε άλλους γνωστούς κόμβους. Από τα pong που λαμβάνει μαθαίνει για άλλους κόμβους. Τα pong μηνύματα περιέχουν διάφορα στοιχεία που μας ενδιαφέρουν, όπως την IP διεύθυνση, τον αριθμό των διαμοιραζόμενων αρχείων και άλλα. Από την επεξεργασία των στοιχείων που συγκεντρώθηκαν προέκυψαν διάφορα στατιστικά που χαρακτηρίζουν κυρίως τους χρήστες των εφαρμογών αυτών και λιγότερο το P2P δίκτυο αυτό καθεαυτό.

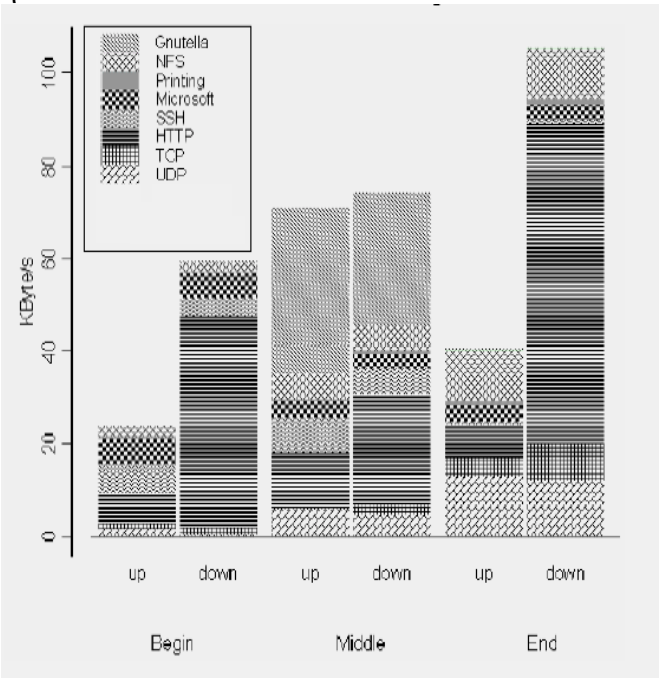
Στο [10] χρησιμοποιείται το NetFlow της Cisco για τη συγέντρωση δεδομένων από δρομολογητές κατά μήκος ενός ISP backbone. Για κάθε ροή (δηλαδή για κάθε ακολουθία πακέτων μεταξύ μιας πηγής και ενός προορισμού) κρατάει στοιχεία όπως: την IP διεύθυνση, network prefix και AS (Autonomus System). Τα συστήματα που μελετώνται είναι το Gnutella, το FastTrack και το DirectConnect. Τα στοιχεία συγκεντρώνονται, στέλνονται με UDP και αποθηκεύονται. Από την ανάλυσή τους προκύπτουν στοιχεία και για την κίνηση που προκαλείται από τις P2P εφαρμογές, αλλά και από το υπόλοιπο δίκτυο. Επίσης, επειδή δεν είναι απαραίτητη η γνώση των πρωτοκόλλων που μελετώνται, μπορούν να συγκεντρωθούν στοιχεία και για πρωτόκολλα τα οποία δεν είναι ανοιχτά (open source) όπως για παράδειγμα το FastTrack. Τα στοιχεία αναφέρονται σε μεγάλα χρονικά διαστήματα και χωρίς να επηρεάζουν τους κόμβους του δικτύου, αλλά συγκεντρώνονται από ένα μικρό τμήμα δρομολογητών.

Στο [3] μελετάται το Gnutella 0.6. Η διάταξη που χρησιμοποιείται φαίνεται στην Εικόνα 14 και σε γενικές γραμμές είναι η εξής: Τοποθετείται ένας κόμβος Gnutella σε ένα τοπικό δίκτυο (LRZ domain), με δημόσιο IP, ο οποίος άλλοτε λειτουργεί σαν φύλλο και άλλοτε σαν ultrapeer. Επίσης, τοποθετείται και ένας καταγραφέας πακέτων (Packet Logger) στον κεντρικό μεταγωγέα του δικτύου (Main Switch of the LAN).



Εικόνα 14: Διάταξη μετρήσεων [3]

Στην Εικόνα 15 παρουσιάζονται τα αποτελέσματα που συγκεντρώθηκαν από μετρήσεις οι οποίες έγιναν για το Gnutella 0.6. Οι δύο πρώτες στήλες παρουσιάζουν την κίνηση πριν ξεκινήσει η P2P εφαρμογή, στις δύο επόμενες, ενώ έχει ξεκινήσει και στις δύο τελευταίες, ενώ έχει σταματήσει. Όπως μπορούμε να δούμε, όσο τρέχει η P2P εφαρμογή, το μεγαλύτερο μέρος της κίνησης προκαλείται από αυτή και όχι από το HTTP ή το TCP.



Εικόνα 15: Μετρήσεις για το Gnutella 0.6 [3]

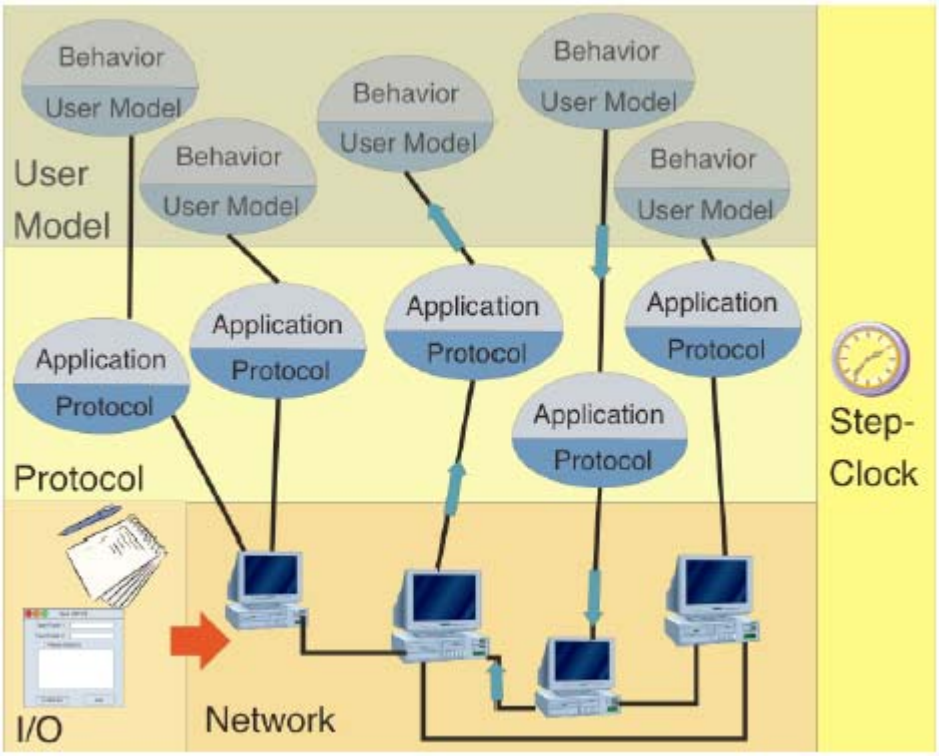
3.2 Προηγούμενες Μελέτες Συστημάτων με Προσομοίωση

Η άμεση μελέτη των P2P συστημάτων σε πραγματικό περιβάλλον μας δίνει στοιχεία για διάφορα χαρακτηριστικά των χρηστών και την κίνηση που αναπτύσσεται, αλλά δεν μας δίνουν στοιχεία για κάποια από τα χαρακτηριστικά που μας ενδιαφέρουν, όπως για παράδειγμα το scalability ή την αντοχή τους σε διαφόρων ειδών επιθέσεις, καθώς κάτι τέτοιο θα προϋπέθετε την εισαγωγή «κακών» κόμβων στο δίκτυο. Ένας τρόπος που μας επιτρέπει να μελετήσουμε τα στοιχεία που θέλουμε σε μεγάλη κλίμακα είναι η προσομοίωση. Ακόμα, με τον τρόπο αυτό είναι δυνατό να μελετηθούν βασικές λειτουργίες των συστημάτων, όπως για παράδειγμα η αναζήτηση-lookup, να συγκριθούν οι πολυπλοκότητές τους υπό τις ίδιες συνθήκες, καθώς και να δοκιμαστούν προτεινόμενες βελτιώσεις τους.

Στις παρακάτω ενότητες παρουσιάζονται κάποια συστήματα προσομοίωσης P2P συστημάτων. Παράλληλα, ορισμένα από αυτά συγκρίνονται σε κάποια βασικά τους σημεία με το σύστημα που αναπτύχθηκε.

3LS

Πρόκειται για έναν time-stepped προσομοιωτή που χρησιμοποιεί ένα κεντρικό ρολόι. Το ρολόι αυτό συγχρονίζει τα διάφορα γεγονότα. Η αρχιτεκτονική του φαίνεται στην Εικόνα 16. Έχει τρία επίπεδα: το επίπεδο του Δικτύου, το επίπεδο Πρωτοκόλλου και το επίπεδο Χρήστη. Χαρακτηριστικό του 3LS προσομοιωτή είναι ότι ο διαχωρισμός των επιπέδων επιτρέπει την προσομοίωση διαφόρων τοπολογιών του δικτύου, ανάλογα με το P2P πρωτόκολλο που προσομοιώνεται. Επιπλέον, μόνο τα άμεσα συνδεδεμένα επίπεδα μπορούν να επικοινωνήσουν άμεσα μεταξύ τους. Δηλαδή, το επίπεδο Πρωτοκόλλου λειτουργεί σαν διεπαφή μεταξύ του επιπέδου του Χρήστη και του Δικτύου. Ο προσομοιωτής παράγει log αρχεία με τα στοιχεία που προκύπτουν κάθε φορά ενώ υπάρχει και η δυνατότητα οπτικοποίησης [19].



Εικόνα 16: Αρχιτεκτονική 3LS [19]

P2PSim

Πρόκειται για έναν προσομοιωτή που αναπτύχθηκε στα πλαίσια του IRIS Project με σκοπό τη μελέτη της απόδοσης των P2P πρωτοκόλλων. Χαρακτηριστικά του συστήματος είναι ότι η προσομοίωση είναι discrete-event και το ότι υποστηρίζει μεν διάφορα P2P πρωτόκολλα, αλλά τα πρωτόκολλα αυτά είναι σχετικά απλοποιημένα σε σύγκριση με τα πραγματικά και συνδέονται άμεσα με το συγκεκριμένο σύστημα. Προς το παρόν υποστηρίζονται το Chord, Koorde, Kelips, Tapestry, και το Kademlia.[20]

The Query Cycle Simulator

Στο σύστημα αυτό προσομοιώνεται τόσο η συμπεριφορά των συνδεδεμένων κόμβων όσο και τα περιεχόμενα των αρχείων που μοιράζονται. Συγκεκριμένα, προσομοιώνεται ο τρόπος με τον οποίο ένας κόμβος υποβάλλει ή απαντά σε μία αίτηση αναζήτησης, η διάρκεια σύνδεσής του με το P2P δίκτυο, καθώς και η ποσότητα, το πόσο «δημοφιλές» είναι ένα αρχείο και το περιεχόμενο των αρχείων που προσφέρει στο δίκτυο. Το περιεχόμενο προσομοιώνεται με σκοπό να αναδειχθούν οι ομαδοποιήσεις (clusters) χρηστών που δημιουργούνται λόγω παρόμοιου περιεχομένου των αρχείων τους. Η διαδικασία προσομοίωσης προχωρά με κύκλους αναζήτησης (query cycles). Δηλαδή, σε κάθε κύκλο ένας κόμβος μπορεί να υποβάλλει μια αίτηση αναζήτησης και να περιμένει απάντηση και μετά να κατεβάζει το αρχείο, να απαντά σε μια αίτηση ή να είναι ανενεργό. Σε κάθε κόμβο συγκεντρώνονται στατιστικά. Γενικά, θα μπορούσε να ειπωθεί ότι η διαδικασία που προσομοιώνεται έχει κοινά στοιχεία με αυτά που προβλέπει το

πρωτόκολλο Gnutella. Τέλος, δεν υπάρχει διαχωρισμός επιπέδων, αλλά όλες οι διαδικασίες γίνονται στο ίδιο επίπεδο.

SimP2

Το SimP^2 είναι ένα εργαλείο προσομοίωσης που προτείνεται σε συνδυασμό με ένα μοντέλο κατασκευής τυχαίου γράφου για τη μελέτη P2P κοινοτήτων (π.χ. Gnutella, Freenet). Το μοντέλο αυτό παράγει έναν μη κανονικό γράφο και επιτρέπει τον έλεγχο της κατανομής των βαθμών των κόμβων του γράφου. Ο προσομοιωτής αποτελείται από δύο τμήματα. Το πρώτο περιλαμβάνει την κατασκευή ενός συνόλου στιγμιότυπων του P2P δικτύου βασιζόμενο στο μοντέλο κατασκευής τυχαίου γράφου και το δεύτερο την προσομοίωση της αναζήτησης δεδομένων και της διαδικασίας ανάκτησης τους. Με το δεύτερο κομμάτι είναι δυνατόν να μελετηθεί τόσο η τοπολογία που αναπτύσσεται όσο και πολλά στοιχεία που σχετίζονται με την απόδοση των υπό μελέτη P2P συστημάτων. Αναφέρουμε ότι προσομοιώνονται και τα περιεχόμενα των αρχείων και η διαδικασία μεταφοράς τους. Τέλος, το Sump^2 δεν μελετά κάποιο από τα υπάρχοντα P2P σύστημα, αλλά ένα μοντέλο που μοιάζει με το Gnutella.

PLP2P

Ο Packet-Level Peer-to-Peer [21] προσομοιωτής έχει σχεδιαστεί έτσι ώστε να μπορεί να τρέχει πάνω από μια πλατφόρμα προσομοίωσης δικτύου σε επίπεδο πακέτου (packet level network simulator) καθώς θεωρείται απαραίτητο να λαμβάνεται υπόψη η κίνηση του (underlying) δικτύου. Αποτελείται από τρία επίπεδα: το PeerAgent, το PeerApp και το SocketAdaption. Το SocketAddaption παρέχει τις βασικές λειτουργίες των Sockets (π. χ. bind, listen, connect) και αποτελεί τη διεπαφή μέσω της οποίας επικοινωνεί η πλατφόρμα προσομοίωσης του δικτύου με τον PLP2P. Το PeerApp παρέχει τη διεπαφή των χρηστών (π.χ. join, leave, search). Επίσης, αυτό το επίπεδο διαχειρίζεται τις συνδέσεις μεταξύ των κόμβων και λαμβάνει μηνύματα Queryhit από το PeerAgent επίπεδο. Το PeerAgent είναι υπεύθυνο για την επεξεργασία και δρομολόγηση των μηνυμάτων. Τέλος, ένα ξεχωριστό τμήμα το ActivityController, που σχετίζεται με το PeerApp επίπεδο, προσομοιώνει τη συμπεριφορά των χρηστών με βάση κάποιο μοντέλο. Υπάρχει η δυνατότητα, τροποποιώντας τα επίπεδα PeerAgent και PeerApp, να επεκταθεί ώστε να υποστηρίζει διαφορετικά P2P συστήματα. Σημειώνουμε ότι οι p2p clients που χρησιμοποιούνται είναι και αυτά συστήματα προσομοίωσης που προσομοιώνουν τον τρόπο λειτουργίας του εκάστοτε συστήματος και όχι πραγματικοί clients. Τέλος, καθώς ο προσομοιωτής δουλεύει σε χαμηλό επίπεδο-επίπεδο πακέτου, η κλιμάκωσή του δεν είναι πολύ καλή. Ωστόσο, αυτό μπορεί να βελτιωθεί με την παράλληλη εκτέλεση του προσομοιωτή σε περισσότερα μηχανήματα.

4 Περιγραφή Συστήματος

Το σύστημα που αναπτύχθηκε είναι ουσιαστικά μία πλατφόρμα πάνω από την οποία μπορούν να τρέξουν p2p clients/ αλγόριθμοι. Τα βασικά χαρακτηριστικά του συστήματος είναι ότι

- Προσομοιώνει τη συμπεριφορά των κόμβων και εξομοιώνει τη λειτουργία του δικτύου. Συνεπώς, δεν θα εφαρμοστούν ακριβώς όσα αναφέρθηκαν παραπάνω για την προσομοίωση, αλλά μόνον όσα χρειάζονται για τη μοντελοποίηση των κόμβων.
- Διαχειρίζεται τα μηνύματα που διακινούνται μεταξύ των κόμβων.
- Οι clients/ αλγόριθμοι που τρέχουν πάνω από αυτό μπορούν να τρέξουν χωρίς αλλαγές και πάνω από το πραγματικό δίκτυο.
- Η αρχιτεκτονική του είναι πολυεπίπεδη και μεταξύ των επιπέδων υπάρχει ανεξαρτησία. Με τον τρόπο αυτό, είναι δυνατό να τρέξει πάνω από την πλατφόρμα οποιαδήποτε P2P εφαρομγή.
- Ο διαχωρισμός των επιπέδων επιτρέπει και την αλλαγή του μοντέλου «συμπεριφοράς» των κόμβων.
- Μόνο τα άμεσα συνδεδεμένα επίπεδα μπορούν να μιλήσουν μεταξύ τους, δηλαδή ένα επίπεδο μπορεί να επικοινωνήσει άμεσα μόνο με το ανώτερο ή κατώτερο ως προς αυτό επίπεδο.

Μας ενδιαφέρει να μελετήσουμε:

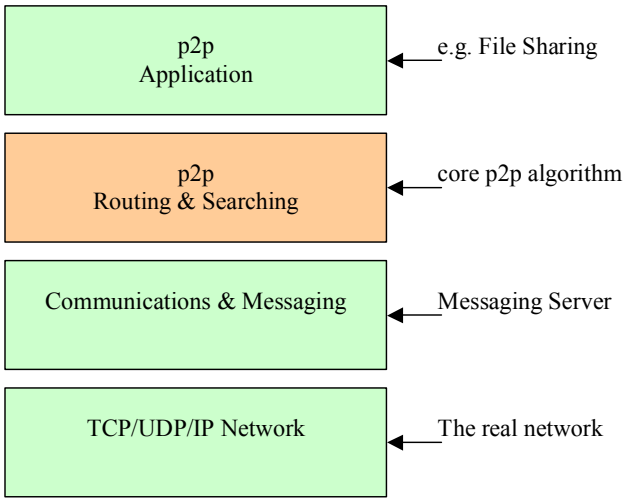
- Τη lookup του εκάστοτε P2P client. Συγκεκριμένα, μας ενδιαφέρει να μελετήσουμε πώς επηρεάζεται η απόδοση της αναζήτησης καθώς αυξάνεται το πλήθος των συνδεδεμένων κόμβων.
- Την κίνηση που προέρχεται αποκλειστικά από τις P2P εφαρμογές και όχι από το HTTP ή το TCP/ UDP/ IP. Δηλαδή μας ενδιαφέρουν τα πακέτα που περιέχουν πληροφορία σχετική με την αναζήτηση ενός αρχείου και όχι τα πακέτα που μεταφέρουν το αρχείο αυτό καθαυτό. Και αυτό γιατί όπως αναφέρθηκε και παραπάνω, αυτά είναι που προκαλούν το μεγαλύτερο ποσοστό της κίνησης στο πραγματικό δίκτυο.
- Στοιχεία που χαρακτηρίζουν το P2P δίκτυο και όχι στοιχεία που εξαρτώνται κυρίως από τη συμπεριφορά του χρήστη, όπως είναι για παράδειγμα η συχνότητα εμφάνισης κάποιων συγκεκριμένων queries, ή το ποσοστό των αναζητήσεων που είχαν θετικό αποτέλεσμα.

4.1 Αρχιτεκτονική Συστήματος

Η αρχιτεκτονική του συστήματος είναι τέτοια ώστε:

- ✓ Να χρησιμοποιεί την ίδια υλοποίηση των P2P clients και με αυτήν που θα μπορούσαμε να χρησιμοποιήσουμε σε ένα πραγματικό P2P δίκτυο.
- ✓ Να μπορούμε να «σηκώνουμε» πολλούς κόμβους στο ίδιο μηχάνημα.

- ✓ Να υπάρχει ανεξαρτησία μεταξύ επιπέδων έτσι ώστε αλλαγές σε κάποιο από αυτά να μην επηρεάζει το σχεδιασμό και τη λειτουργία των υπολοίπων.
- ✓ Κάθε επίπεδο να μπορεί να επικοινωνήσει μόνο με το ανώτερο ή κατώτερο επίπεδό του.
- ✓ Να μπορεί να υποστηρίξει οποιονδήποτε P2P client χωρίς να απαιτούνται αλλαγές στα υπόλοιπα επίπεδα.
- ✓ Να λειτουργεί σε επίπεδο μηνύματος, δηλαδή όλη η πληροφορία που διακινείται είναι με τη μορφή διαφόρων ειδών μηνυμάτων.



Εικόνα 17: Αρχιτεκτονική Συστήματος

Όπως φαίνεται και στην Εικόνα 16, το σύστημα αποτελείται από 4 επίπεδα:

1**. P2P Application**: Στο ανώτερο επίπεδο βρίσκεται η P2P εφαρμογή η οποία μπορεί να είναι για παράδειγμα ένα σύστημα διαμοιρασμού αρχείων όπως είναι το eMule ή το KaZaA.

2. **P2P Routing and Searching**: Στο αμέσως επόμενο βρίσκεται ο P2P αλγόριθμος-πρωτόκολλο. Στην περίπτωση του eMule το πρωτόκολλο που χρησιμοποιείται είναι το Kademlia και για το KaZaA το FastTrack.

3. **Communications and Messaging**: Το επίπεδο αυτό είναι υπεύθυνο για την επικοινωνία μεταξύ των κόμβων (p2pclients) που συμμετέχουν στο P2P δίκτυο, την παρακολούθηση του bandwidth των p2pclients που είναι συνδεδεμένοι μαζί του και τη συλλογή στατιστικών.
4. **Network – TCP/ UDP/ IP**: Στο κατώτερο επίπεδο βρίσκεται το πραγματικό δίκτυο πάνω από το οποίο σχηματίζεται το P2P δίκτυο και το οποίο είναι υπεύθυνο για τη μεταφορά των πακέτων που περιέχουν τα αρχεία αυτά καθαυτά.

4.2 Περιγραφή Επιπέδου Routing and Searching

Στο επίπεδο αυτό μπορεί να βρίσκεται οποιοσδήποτε P2P αλγόριθμος. Μπορεί να είναι για παράδειγμα ένας Gnutella, Chord ή CAN client. Κάθε κόμβος που θα συνδέεται με το δίκτυο θα τρέχει έναν τέτοιο P2P αλγόριθμο. Θεωρούμε ότι κάθε ένας από αυτούς τους αλγορίθμους υλοποιείται ακολουθώντας ένα συγκεκριμένο interface το οποίο φαίνεται στο Παράρτημα Α.

Προς το παρόν θεωρούμε έναν πολύ απλό Client με τον οποίο θα δοκιμαστεί το σύστημα. Ο Client αυτός παρουσιάζεται αναλυτικά παρακάτω.

Ένας Client χαρακτηρίζεται από το όνομά του (clientname), τη διεύθυνσή του, τη διεύθυνση του επόμενου και του προηγουμένου του και έναν αναγνωριστικό αριθμό. Επίσης, διατηρεί μια λίστα με αναγνωριστικά αρχείων (IDs), κάθε ένα από τα οποία αντιστοιχεί σε κάθε ένα αρχείο που διαθέτει. Για την επικοινωνία μεταξύ των p2p clients, καθιερώνεται ένα σύστημα ονοματολογίας το οποίο αξιοποιεί την ύπαρξη των Messaging Servers (βλ. 4.3). Έτσι η διεύθυνση κάθε p2p client έχει την μορφή:

p2p://hostname:port/clientname

- **p2p**: Υποδεικνύει το πρωτόκολλο επικοινωνίας, προς το παρόν το μοναδικό
- **hostname**: Η διεύθυνση του PC, μπορεί να είναι σκέτο IP ή της μορφής www.msgsrv.com
- **port**: Δείχνει το port του Messaging Server. Εάν έχουμε πολλά Messaging Servers σε κάθε PC τότε αυτά έχουν διαφορετικό port το καθένα.
- **clientname**: Το τοπικό (μέσα στο ίδιο Messaging Server) όνομα του p2p client

Για την επικοινωνία μεταξύ των p2pclients ανταλλάσσονται δύο ειδών μηνύματα:

- **FindReqMessage**: Στέλνεται για μια αίτηση αναζήτησης. Περιλαμβάνει τις εξής πληροφορίες: τη διεύθυνση του αποστολέα, το ID του μηνύματος που ψάχνει και ένα αναγνωριστικό του μηνύματος. Το αναγνωριστικό αυτό υπάρχει για να μπορέσει να ξεχωρίσει την απάντηση όταν τη λάβει, καθώς ένας p2pclient μπορεί να δέχεται ταυτόχρονα πολλές απαντήσεις.

- **FindResMessage**: Στέλνεται σαν απάντηση σε ένα FindReqMessage. Περιλαμβάνει τις εξής πληροφορίες: τη διεύθυνση του αποστολέα, τη διεύθυνση του επόμενου του, το ID για το οποίο έψαξε, το αναγνωριστικό του μηνύματος στο οποίο απαντάει και ένα πεδίο found που είναι αληθές όταν το ID βρέθηκε.

Ο συγκεκριμένος Client, για λόγους απλότητας, δεν υλοποιεί όλες τις βασικές λειτουργίες που περιγράφηκαν στην εννότητα 1.2.3. Από αυτές, υλοποιείται μόνο η αναζήτηση και κάποιες άλλες βοηθητικές. Στη συνέχεια περιγράφονται αναλυτικότερα οι λειτουργίες αυτές:

- **Παραγωγή ID Διαθέσιμων αρχείων** (createFileList): Όπως αναφέρθηκε, κάθε φορά που δημιουργείται ένας client παράγει και μια λίστα από ID, δηλαδή από αναγνωριστικά αρχείων τα οποία θα θεωρούμε ότι μοιράζεται. Αυτά παράγονται τυχαία (χρησιμοποιώντας την κλάση Random της Java).

- **Ορισμός επόμενου**: Όταν ένας p2pclient εισάγεται στο σύστημα ορίζεται και η διεύθυνση του επομένου του.

- **Ορισμός προηγούμενου**: Όταν ένας p2pclient εισάγεται στο σύστημα ορίζεται και η διεύθυνση του προηγουμένου του.

- **Αναζήτηση** (lookup)**:** Η αναζήτηση γίνεται με βάση το αναγνωριστικό του αρχείου που ψάχνουμε. Έστω ότι ο client1 ψάχνει για το ID. Τότε ετοιμάζει ένα μήνυμα αναζήτησης (FindReqMessage), εισάγει σε έναν πίνακα hash την τιμή null με κλειδί το αναγνωριστικό του μηνύματος αυτού, το στέλνει στον επόμενό του, τον client2 και περιμένει για απάντηση. Περιμένει για ένα χρονικό διάστημα π.χ. 500msec. Αν δεν λάβει απάντηση μέχρι τότε, η αναζήτηση σταματά. Αυτό το καταλαβαίνει ελέγχοντας τον πίνακα hash (βλέπε παρακάτω: Λήψη μηνύματος (getMessage)). Αν λάβει θετική απάντηση, δηλαδή το found είναι αληθές, η αναζήτηση σταματά και επιστέφεται η διεύθυνση του p2pclient που έχει το ID. Διαφορετικά, εξάγει από το FindResMessage τη διεύθυνση του επόμενου και στέλνει τώρα σε αυτόν ένα `FindReqMessage` Όπως αναφέρθηκε, η διαδικασία αυτή συνεχίζεται μέχρι να βρεθεί το ID ή μέχρι να μη ληφθεί έγκαιρα η απάντηση ή μέχρι ο επόμενος p2pclient που εξάγει από το απαντητικό μήνυμα να είναι ο εαυτός του.

- **Λήψη μηνύματος** (getMessage): Όταν λαμβάνεται ένα μήνυμα γίνονται τα εξής: αν το μήνυμα είναι FindReqMessage, ο παραλήπτης ψάχνει για το ID που του ζητείται. Σε κάθε περίπτωση απαντά με ένα FindReqMessage. Αν το έχει βρει το πεδίο found, τίθεται αληθές, διαφορετικά τίθεται ψευδές. Αν το μήνυμα είναι FindResMessage, τότε πρόκειται για απάντηση σε κάποια αίτησή του, εξάγει το αναγνωριστικό του και το εισάγει στον πίνακα hash του p2pclient που περιμένει αυτή την απάντηση με κλειδί το αναγνωριστικό αυτό. Στη συνέχεια αναλαμβάνει η lookup.

4.3 Περιγραφή Επιπέδου Communications and Messaging

Σε ένα τυπικό σύστημα που συμμετέχει σε ένα P2P network, θα υπάρχει (τουλάχιστον) ένα P2P client και (τουλάχιστον) ένα Messaging Server, τα οποία θα επικοινωνούν μεταξύ τους απευθείας, χωρίς να χρησιμοποιείται RMI, sockets ή κάποιος άλλος μηχανισμός, το οποίο σημαίνει ότι θα εκτελούνται μέσα στο ίδιο πρόγραμμα (JVM instance για την ακρίβεια). Υπάρχει όμως και η δυνατότητα να δημιουργηθούν περισσότεροι P2P clients στο ίδιο σύστημα (PC ή για την ακρίβεια JVM instance). Αυτά τα P2P clients θα επικοινωνούν απευθείας με το ίδιο Messaging Server, ώστε να μην υπάρχει υπερχρησιμοποίηση των sockets για ένα σύστημα.

Βασικό ρόλο στο επίπεδο αυτό παίζει ο Messaging Server. Ένας Messaging Server έχει ένα δημόσιο όνομα (msgServerPublicName) το οποίο χρησιμοποιούν οι άλλοι Messaging Servers όταν θέλουν να επικοινωνήσουν μαζί του και μία θύρα (port) στην οποία ακούει αιτήσεις σύνδεσης. Εκτός από το msgServerPublicName έχει και μια σειρά άλλα ονόματα Τα ονόματα αυτά χρησιμοποιούνται μόνο τοπικά, δηλαδή μόνον

από τους p2pclients που είναι συνδεδεμένοι με αυτόν. Παραδείγματα τέτοιων ονομάτων είναι "localhost", 127.0.0.1, www.msgsrv.com. Αυτά τα κρατάει σε μια λίστα (msgServerName). Ακόμα, κρατάει και δύο λίστες για τον έλεγχο του bandwidth (bWcheckListIn, bWcheckListOut) και έναν πίνακα hash (msgClient) για τους συνδεδεμένους p2pclients.

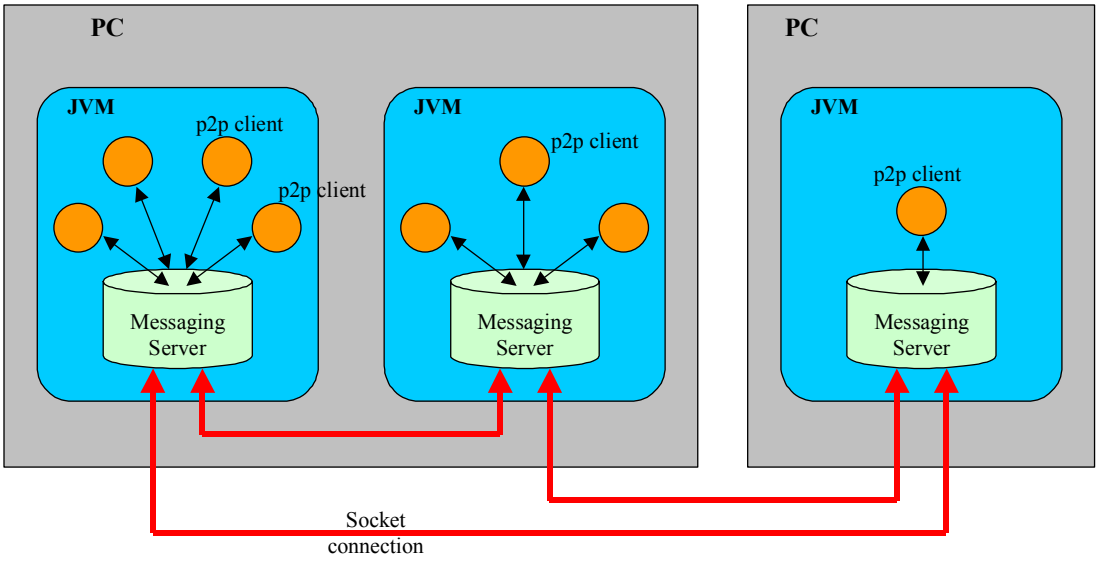
Παρακάτω παρουσιάζονται αναλυτικά οι λειτουργίες του:

- **Σύνδεση p2pclient** (registerClient): Κάθε p2pclient που εισέρχεται στο σύστημα συνδέεται με έναν Messaging Server και μόνο μέσω αυτού μπορεί να επικοινωνήσει με ένα άλλο p2pclient, είτε αυτό ανήκει στον ίδιο Messaging Server είτε σε διαφορετικό. Ο p2pclient εισάγεται στον πίνακα hash msgClient με κλειδί το όνομά του το οποίο είναι μοναδικό σε αυτόν τον Messaging Server.

- **Αποσύνδεση p2pclient** (unregisterClient): Κάθε p2pclient που εξέρχεται από το σύστημα αποσυνδέεται από τον Messaging. Ο p2pclient διαγράφεται από το msgClient.

- **Αποστολή Μηνύματος** (sendMessage): Αναλαμβάνει την αποστολή ενός μηνύματος από έναν p2pclient σε έναν άλλο. Έστω ότι ο p2pclientA θέλει να στείλει ένα μήνυμα στον p2pclientB. Τότε ο p2pclientA δεν του το στέλνει απευθείας. Αν ανήκουν και οι δύο στον ίδιο Messaging Server, τότε μεσολαβεί και του το παραδίδει αυτός. Διαφορετικά, αν ανήκουν σε διαφορετικούς Messaging Servers τότε επικοινωνούν μέσω αυτών. Συγκεκριμένα, δημιουργείται μεταξύ τους μία σύνδεση μέσω Socket. Ο Messaging Server του p2pclientA «πακετάρει» το μήνυμα. Το νέο πακέτο περιλαμβάνει το μήνυμα, το μέγεθός του και το μέγεθος της διεύθυνσης προορισμού και αποστολής. Με αυτόν τον τρόπο ο Messaging Server που παίρνει το μήνυμα μπορεί να καταλάβει αν το μήνυμα που έλαβε είναι αυτό που στάλθηκε ή αλλοιώθηκε κατά τη μετάδοση. Σε κάθε περίπτωση, πριν την αποστολή ελέγχεται η ορθότητα της διεύθυνσης προορισμού, ώστε να ακολουθεί τους κανόνες ονοματολογίας (βλ. 4.5).

Τα παραπάνω παρουσιάζονται στην Εικόνα 18:



Εικόνα 18: Αποστολή μηνύματος

- **Έλεγχος bandwidth** (bWcheckIn, bWcheckOut): Ένας p2pclient μπορεί να στείλει ή να λάβει ένα μήνυμα υπό την προϋπόθεση ότι δεν έχει εξαντλήσει το εισερχόμενο (incoming) bandwidth για την λήψη και το εξερχόμενο (outgoing) bandwidth του για την αποστολή. Τα bandwidth αυτά εξαρτώνται από το είδος της σύνδεσης του p2pclient (π.χ. 56kbps, DSL κ.λ.π.). Κάθε φορά που ξεκινά η αποστολή ή η λήψη ενός μηνύματος, ο Messaging Server ελέγχει το αντίστοιχο bandwidth. Αν δεν έχει ξεπεραστεί το bandwidth, ολοκληρώνει τη διαδικασία κανονικά, όπως αναφέρθηκε παραπάνω. Σε αντίθετη περίπτωση, μηνύματα απορρίπτονται και δε στέλνονται/ λαμβάνονται.

- **Συγκέντρωση και καταγραφή στατιστικών:** Κάθε Messaging Server κρατάει στατιστικά στοιχεία που αφορούν τα μηνύματα που διακινούνται μεταξύ των p2pclients που είναι συνδεδεμένοι με αυτόν. Αυτά αποθηκεύονται σε ένα αρχείο (monitorOutputport.txt, όπου port είναι η θύρα του Messaging Server). Στο αρχείο αυτό αποθηκεύονται τα εξής στοιχεία: η χρονική στιγμή (σε Milliseconds) που λαμβάνεται ή στέλνεται το μήνυμα, η διεύθυνση του αποστολέα, η διεύθυνση προορισμού, το μέγεθος του μηνύματος (bytes) και το είδος του μηνύματος. Ένα μήνυμα μπορεί να είναι μήνυμα που στάλθηκε (msgS), μήνυμα που λήφθηκε (msgR), μήνυμα σύνδεσης (rgs) ενός client με τον Messaging Server, μήνυμα αποσύνδεσης (urgs) ενός client με τον Messaging Server. Ακόμα, κρατάει τα ίδια στατιστικά και για τα λανθασμένα πακέτα, δηλαδή πακέτα των οποίων η διεύθυνση αποστολέα ή προορισμού δεν υπάρχει, το καταγράφει (hostErr αν το πρόβλημα δημιουργείται λόγω του localhost clErr. λόγω του clientname). Τέλος, καταγράφονται και τα μηνύματα που απορρίπτονται λόγω υπέρβασης του ορίου bandwidth(drIn για το εισερχόμενα και drOut για το εξερχόμενα). Τελικά, παράγεται ένα log αρχείο για κάθε Messaging Server με όλα τα παραπάνω στοιχεία. Μια τυπική εγγραφή του αρχείου αυτού είναι:

  1091009727921, msgS, p2p://localhost:1010/client0,
  p2p://localhost:1010/client1, 424

Κάθε φορά που συνδέεται ή αποσυνδέεται ένας p2pclient, στέλνεται ή λαμβάνεται ένα μήνυμα προστίθεται μία τέτοια εγγραφή στο αρχείο.

Παραγωγή ID p2pclient: Όπως αναφέρθηκε, κάθε φορά που δημιουργείται ένας p2pclient και συνδέεται με έναν Messaging Server, αυτό του παράγει και ένα ID, δηλαδή έναν αναγνωριστικό αριθμό.

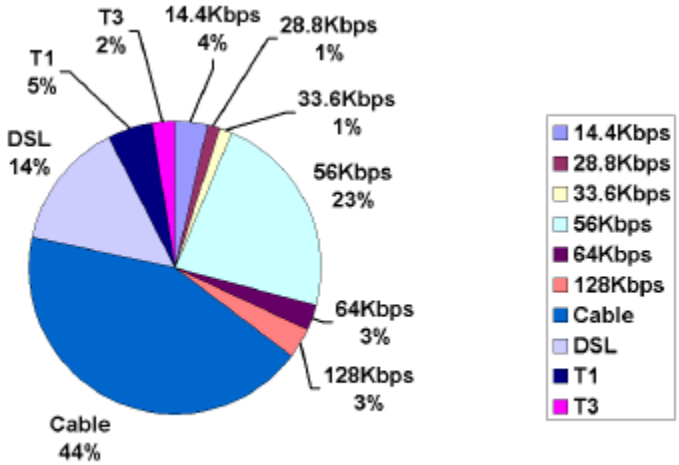
4.4 Μοντελοποίηση Κόμβων

Πρέπει να προσομοιώσουμε τη συμπεριφορά ενός κόμβου ως προς:

1. Τη διάρκεια παραμονής του στο σύστημα.
2. Τον τρόπο με τον οποίο υποβάλλει και απαντά σε αιτήσεις αναζήτησης (lookup queries).
3. Το bandwidth που του ανατίθεται κατά την αρχικοποίηση του.
4. Τον τρόπο με τον οποίο συνδέεται με άλλους κόμβους όταν εισάγεται στο σύστημα.

Αναλυτικότερα:

1. Η προσομοίωση θα σταματήσει όταν κάθε client θα έχει υποβάλει ένα συγκεκριμένο αριθμό από αιτήσεις αναζήτησης. Ο αριθμός ορίζεται εξωτερικά.
2. Όπως αναφέρεται στο [2] οι χρόνοι μεταξύ διαδοχικών αιτήσεων αναζήτησης θα μοντελοποιηθούν ως ανεξάρτητες τυχαίες μεταβλητές, που περιγράφονται από την κατανομή Poisson. Η μέση τιμή καθορίζεται εξωτερικά. Ο τρόπος παραγωγής αυτών των αριθμών περιγράφεται αναλυτικά στις ενότητες 2.5.
3. Το bandwidth που θα διαθέτει κάθε κόμβος θα καθορίζεται από πραγματικά δεδομένα, όπως αυτά παρουσιάζονται στο [2] και φαίνονται στο διάγραμμα της Εικόνας 19.



Εικόνα 19: Μετρήσεις Bandwidths για το Gnutella

Στο διάγραμμα αυτό παρουσιάζεται η κατανομή των bandwidths στους χρήστες του Gnutella. Θεωρούμε ότι μπορούν να χρησιμοποιηθούν και για κάθε άλλο P2P σύστημα. Οι τιμές αυτές προέκυψαν με τον τρόπο που περιγράφηκε στην ενότητα 3.1. Στον Πίνακα 7 φαίνεται αναλυτικά το bandwidth που αντιστοιχεί σε κάθε είδος σύνδεσης (upstream και downstream). Έτσι, στο σύστημά μας, αν υποθέσουμε ότι έχουμε 100 κόμβους, σύμφωνα με την Εικόνα 19, 23 από αυτούς θα έχουν σύνδεση στα 56Kbps δηλαδή θα έχουν upstream όριο 56,000 bytes/sec και downstream όριο 33,000 bytes/sec.

Πίνακας 6: Bandwidths για τους κόμβους του Gnutella

Είδος Σύνδεσης	**Downstream kbit/sec**	**Bandwidth (bytes/sec) downstream**	**Upstream kbit/sec**	**Bandwidth (bytes/sec) Upstream**
14.4 kbps	14.4	14,000	14,4	14,400
28.8 kbps	28.8	28,800	28,8	28,800
33.6 kbps	33.6	33,600	33.6	33,600
56 kbps	56	56,000	33	33,000
64 kbps	64	64,000	64	64,000
128 kbps	128	128,000	128	128,000
Cable	187	187,000	32	32,000
Cable	187	187,000	64	64,000
DSL	256	256,000	128	128,000
DSL	384	384,000	91	91,000
DSL	512	512,000	91	91,000
DSL	512	512,000	128	128,000
DSL	640	640,000	90	90,000
T1	1500	1,500,000	1500	1,500,000
T3	44Mbps	44,000,000	44Mbps	44,000,000

4. Θεωρούμε ότι οι κόμβοι συνδέονται με τέτοιο τρόπο ώστε να σχηματίζουν έναν δακτύλιο. Συγκεκριμένα, αφού συνδεθούν με έναν Messaging Server, κάθε ένας αποκτά έναν επόμενο και έναν προηγούμενο. Ο επόμενος του τελευταίου κόμβου ενός Messaging Server είναι ο πρώτος κόμβος ενός άλλου Messaging Server. Αντίστοιχα, ο προηγούμενος του πρώτου κόμβου ενός Messaging Server είναι ο τελευταίος ενός άλλου Messaging Server. Έστω ότι έχουμε δύο Messaging Servers τους localhost:1010 και localhost: 2020. Έστω ακόμα ότι σε κάθε έναν από τους αυτούς συνδέονται τρεις p2pclients. Τότε, σύμφωνα με τα παραπάνω, ο επόμενος του p2p://localhost:2020/client3 είναι ο p2p://localhost:1010/client1 και ο προηγούμενος του ο p2p: //localhost:1010/client1 είναι ο p2p:// localhost: 2020/ client3.

4.5 Παράδειγμα Λειτουργίας Του Συστήματος

Αρχικοποίηση του συστήματος:
Έστω ότι έχουμε δύο Messaging Servers:

- localhost:1010
- localhost:2020

Με τον localhost:1010 έχουν συνδεθεί (register) δύο clients:

- p2p://localhost:1010/client1
- p2p://localhost:1010/client2

Με τον localhost:2020 έχει συνδεθεί ένας:

- p2p://localhost:2020/client1

Για κάθε έναν από τους συνδεδεμένους p2pclients ξεκινάει ένα thread (App). Για κάθε thread η μεταβλητή strartme τίθεται true, ενώ στη συνέχεια ξεκινά να κάνει αναζήτηση (lookup). Όταν έχει πλέον ξεκινήσει ένα thread για κάθε p2pclient η main περιμένει (sleep) για ορισμένο χρονικό διάστημα (MAIN_SLEEPS_FOR). Μόλις ολοκληρωθεί το διάστημα αυτό, για κάθε thread η μεταβλητή startme τίθεται false και «κοιμάται» (sleep). Η main στέλνει interrupt σε κάθε thread οπότε το thread γίνεται έτοιμο (Ready) και μόλις «ξυπνήσει» διακόπτεται από ένα `InterruptedException`. Σημειώνουμε ότι όσα περιγράψαμε στην παράγραφο αυτή γίνονται στις κλάσεις SampleApp και App της υλοποίησης.

Σενάριο 1: Επικοινωνία clients που ανήκουν στον ίδιο Messaging Server

Έστω ότι ο client με διεύθυνση p2p://localhost:1010/client1 θέλει να στείλει ένα μήνυμα στον p2p://localhost:1010/client2. Το μήνυμα φεύγει από το routing/ searching layer και πάει στο Messaging Server. Αυτός κοιτάει αν p2p://localhost:1010/client1 έχει υπερβεί το όριο του εισερχόμενου bandwidth. Αν δεν το έχει υπερβεί κοιτάζει το hostname και το port του προορισμού και, αν είναι ο ίδιος, κοιτάζει τις τοπικές του καταχωρήσεις να δει εάν υπάρχει το αντίστοιχο clientname. Ο client2 υπάρχει, αλλά πριν του προωθήσει το μήνυμα, καταγράφει στο log του ότι έστειλε ένα μήνυμα (msgS) και στη συνέχεια ελέγχει το εισερχόμενο bandwidth του client2. Έστω ότι δεν το έχει υπερβεί, οπότε του το παραδίδει και καταγράφει ότι ο client2 έλαβε ένα μήνυμα (msgR).

Σενάριο 2: Επικοινωνία clients που ανήκουν σε διαφορετικό Messaging Server

Έστω τώρα ότι ο client με διεύθυνση p2p://localhost:1010/client1 θέλει να στείλει ένα μήνυμα στον p2p://localhost:2020/client1. Το μήνυμα φεύγει από το routing/searching layer και πάει στο Messaging Server. Αυτός κοιτάει αν p2p://localhost:1010/client1 έχει υπερβεί το όριο του εξερχόμενου bandwidth του. Έστω ότι δεν το έχει υπερβεί, οπότε κοιτάζει το hostname και το port του προορισμού. Εφόσον δεν είναι ο ίδιος, προσπαθεί να επικοινωνήσει με τον localhost:2020 ανοίγοντας ένα socket. Εφόσον ο localhost αυτός υπάρχει και ακούει σε αυτό το port, το socket θα δημιουργηθεί, ο localhost:2020 θα πάρει το μήνυμα και ο localhost:1010 θα καταγράψει ότι έστειλε ένα μήνυμα (msgS). Στη συνέχεια, ο localhost:2020 κοιτάζει τις τοπικές του καταχωρήσεις να δει εάν υπάρχει το αντίστοιχο clientname. Ο client1 υπάρχει, αλλά πριν

του προωθήσει το μήνυμα, ελέγχει το εισερχόμενο bandwidth του. Έστω ότι δεν το έχει υπερβεί, οπότε του το παραδίδει και καταγράφει ότι ο client1 έλαβε ένα μήνυμα (msgR).

Σενάριο 3: Υπέρβαση Bandwidth

Έστω ότι ο client με διεύθυνση p2p://localhost:1010/client1 θέλει να στείλει ένα μήνυμα στον p2p://localhost:1010/client2. Το μήνυμα φεύγει από το routing/ searching layer και πάει στο Messaging Server. Αυτός κοιτάει αν p2p://localhost:1010/client1 έχει υπερβεί το όριο του εξερχόμενου bandwidth. Έστω ότι το έχει υπερβεί. Τότε ο p2p://localhost:1010 καταγράφει ότι απέρριψε το μήνυμα (drOut).

Σενάριο 3: Πιθανό σφάλμα στο localhost

Έστω τώρα ότι ο client με διεύθυνση p2p://localhost:1010/client1 θέλει να στείλει ένα μήνυμα στον p2p://localhost:3030/client1. Το μήνυμα φεύγει από το routing/searching layer και πάει στο Messaging Server. Αυτός κοιτάει αν p2p://localhost:1010/client1 έχει υπερβεί το όριο του εξερχόμενου bandwidth του. Αν δεν το έχει υπερβεί, κοιτάζει το hostname και το port του προορισμού. Εφόσον δεν είναι ο ίδιος, προσπαθεί να επικοινωνήσει με τον localhost:3030 ανοίγοντας ένα socket. Ο localhost όμως δεν ακούει σε αυτό το port, οπότε το socket δε δημιουργείται και το μήνυμα απορρίπτεται. Ο localhost:1010 το καταγράφει στο log του ως σφάλμα (hostErr).

Σενάριο 4: Πιθανό σφάλμα στο clientname

Έστω τώρα ότι ο client με διεύθυνση p2p://localhost:1010/client1 θέλει να στείλει ένα μήνυμα στον p2p://localhost:1010/client3. Το μήνυμα φεύγει από το routing/ searching layer και πάει στο Messaging Server. Αυτός κοιτάει αν p2p://localhost:1010/client1 έχει υπερβεί το όριο του εξερχόμενου bandwidth του. Αν δεν το έχει υπερβεί κοιτάζει το hostname και το port του προορισμού. Εφόσον είναι ο ίδιος, κοιτάζει τις τοπικές του καταχωρήσεις να δει εάν υπάρχει το αντίστοιχο clientname. Ο client3 δεν υπάρχει, οπότε το μήνυμα απορρίπτεται και ο localhost:1010 το καταγράφει στο log του ως σφάλμα.

4.6 Σύγκριση Συστημάτων Προσομοίωσης

Λαμβάνοντας υπόψη τις περιγραφές των προσομοιωτών που προηγήθηκαν στην ενότητα 3.2, παρουσιάζονται τα συστήματα αυτά σε σύγκριση με το δικό μας σύστημα, ως προς κάποια βασικά χαρακτηριστικά (Πίνακας 7).

Πίνακας 7: Ομοιότητες και Διαφορές μεταξύ του συστήματός μας και άλλων προσομοιωτών.

Σύστημα	*Ομοιότητες*	*Διαφορές*
3LS	Αρχιτεκτονική πολλών επιπέδων. Δυνατότητα μελέτης πολλών p2p clients. Δυνατότητα υποστήριξης διαφόρων μοντέλων συμπεριφοράς κόμβων-χρηστών	Αυστηρή προσομοίωση (time stepped) Δεν χρησιμοποιούνται «πραγματικοί» clients
P2Psim	Δυνατότητα μελέτης πολλών p2p clients.	Δεν χρησιμοποιούνται «πραγματικοί» clients. Discrete Event προσομοίωση.
The Query Cycle Model		Δεν υπάρχει πολυεπίπεδη αρχιτεκτονική. Προσομοιώνει μόνο το Gnutella. Προσομοίωση των περιεχομένων και της ζήτησης των αρχείων.
SimP^2		Δημιουργία της τοπολογίας του δικτύου με το μοντέλο κατασκευής τυχαίου γράφου. Δεν υπάρχει πολυεπίπεδη αρχιτεκτονική Προσομοιώνει μόνο ένα είδος Gnutella
PLP2P	Αρχιτεκτονική πολλών επιπέδων Δυνατότητα μελέτης πολλών p2p clients. Δυνατότητα υποστήριξης διαφόρων μοντέλων συμπεριφοράς κόμβων-χρηστών	Discrete Event προσομοίωση Προσομοίωση σε επίπεδο πακέτου. Δεν χρησιμοποιούνται πραγματικοί clients.

Αναλύοντας περισσότερο τα στοιχεία του παραπάνω πίνακα, θα μπορούσαμε να πούμε τα εξής. Το 3LS όπως και το δικό μας σύστημα αποτελείται από πολλά επίπεδα. Κάθε ένα από αυτά επικοινωνεί μόνο με τα επίπεδα που συνδέονται άμεσα μεταξύ τους. Παρόμοια αρχιτεκτονική έχει και το PLP2P. Αυτό τους δίνει τη δυνατότητα υποστήριξης πολλών διαφορετικών P2P clients τροποποιώντας ή αντικαθιστώντας απλά κάποιο από τα επίπεδα, καθώς και διαφορετικά μοντέλα «συμπεριφοράς» κόμβων-χρηστών. Αντίθετα, τα συστήματα Query Cycle Model και SimP^2 είναι σχεδιασμένα σε ένα επίπεδο ώστε να προσομοιώνουν ένα συγκεκριμένο P2P client (π. χ. τον Gnutella ή κάποιον που μοιάζει με Gnutella). Κοινή διαφορά όλων των συστημάτων με το δικό μας

είναι ότι είναι σχεδιασμένα με αυστηρή προσομοίωση, καθώς επίσης και ότι οι P2P clients που υποστηρίζουν δεν θα μπορούσαν να τρέξουν σε πραγματικό δίκτυο, σε αντίθεση με το τους P2P client που θα τρέχουν πάνω από το δικό μας σύστημα. Τέλος, ο PLP2P λειτουργεί σε επίπεδο πακέτου ενώ το δικό μας σε επίπεδο μηνύματος (message-driven).

5 Πειράματα - Κατανομές

5.1 Σκοπός Των Πειραμάτων

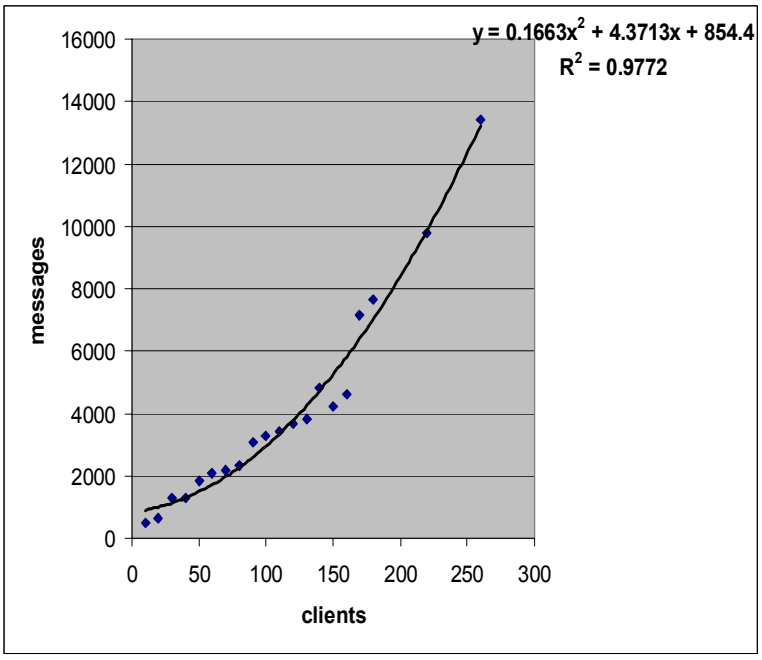
Σκοπός μας είναι να μελετήσουμε τη συμπεριφορά της βασικής λειτουργίας ενός p2p client, της αναζήτησης (lookup) και να προσδιορίσουμε την πολυπλοκότητά της. Μας ενδιαφέρει δηλαδή, να βρούμε το πλήθος των μηνυμάτων που ανταλλάσσονται και πώς αυτό μεταβάλλεται με την αύξηση του πλήθους των κόμβων που είναι κάθε φορά συνδεδεμένοι στο δίκτυο.

Στα παρακάτω πειράματα μελετάμε τον client που περιγράφεται στην ενότητα 4.2.

5.2 Περιγραφή Διαδικασίας - Συμπεράσματα

Για το σκοπό αυτό χρειάζεται να συγκεντρώσουμε στοιχεία για διαφορετικό πλήθος κόμβων. Θεωρούμε την περίπτωση στη οποία έχουμε μόνο έναν messaging server. Το πρόγραμμα ξεκινά να τρέχει π.χ για 10 κόμβους. Σε κάθε επανάληψη του προγράμματος το πλήθος των κόμβων αυξάνεται κατά 10. Σε κάθε επανάληψη, παράγεται ένα log αρχείο που κρατάει στοιχεία για τα μηνύματα που ανταλλάχτηκαν (βλ. Ενότητα 4.3). Η διαδικασία συνεχίζεται μέχρι και για 300 συνδεδεμένους κόμβους. Τελικά, μετά την ολοκλήρωση των 300 επαναλήψεων, υπολογίζεται ο συνολικός αριθμός των μηνυμάτων που στάλθηκαν, διαιρείται με το πλήθος των κόμβων, και το αποτέλεσμα αποθηκεύεται σε ένα άλλο αρχείο (stat.txt) που θα χρησιμοποιηθεί στο τέλος για την παραγωγή των γραφικών παραστάσεων.

Ολη την παραπάνω διαδικασία επαναλαμβάνουμε 3 φορές. Τελικά, από τα 3 stat αρχεία που παρήχθησαν, υπολογίζουμε το μέσο όρο των μηνυμάτων που ανταλλάχθηκαν για κάθε τιμή client, οπότε και παράγεται ένα τελικό αρχείο στατιστικών.



Εικόνα 20: Γραφική παράσταση Αναζήτησης (Lookup)

Θεωρούμε μέση τιμή για την Poisson λ= 4. Στην Εικόνα 20 απεικονίζονται τα σημεία από το τελικό αρχείο στατιστικών και η καμπύλη που περνάει από αυτά, χρησιμοποιώντας τη μέθοδο των ελαχίστων τετραγώνων (που υπάρχει ως επιλογή στο MS-Excel). Οι αριθμητικές τιμές φαίνονται στον Πίνακα 8.

Clients	Messages
10	478
20	623
30	1307
40	1292
50	1843
60	2063
70	2186
80	2318
90	3062
100	3272
110	3426
120	3692
130	3819
140	4827
150	4240
160	4612
170	7139
180	7640
220	9784
260	13395

Πίνακας 8: Αριθμητικές τιμές

Η αναμενόμενη πολυπλοκότητα της lookup για τον συγκεκριμένο client είναι $O(n^2)$ καθώς στη χειρότερη περίπτωση, αν n είναι το πλήθος των συνδεδεμένων κόμβων και κάθε κόμβος θέλει να επικοινωνήσει με όλους αυτούς, θα πρέπει να στείλει n-1 μηνύματα. Η καμπύλη που προκύπτει προσεγγίζεται από τη συνάρτηση $y = 0.1663x^2 + 4.3713x + 854.4$ η οποία είναι $O(n^2)$.

6 Επίλογος/ Μελλοντικές επεκτάσεις

Συμπερασματικά, το σύστημα που υλοποιήθηκε είναι μία πλατφόρμα για τη μελέτη P2P πρωτοκόλλων ως προς τους μηχανισμούς δρομολόγησης και αναζήτησης που αυτά προβλέπουν, καθώς και ενδεχόμενων βελτιώσεων ή επεκτάσεών τους. Θα μπορούσαμε να πούμε ότι παρέχει μία κοινή βάση για σύγκριση μεταξύ των πρωτοκόλλων αυτών και την για αξιολόγησή τους.

Το σύστημα δοκιμάστηκε μόνο για έναν πολύ απλό client. Σκοπός του είναι να μελετηθούν πραγματικοί p2p clients οι οποίοι θα μπορούσαν να τρέξουν χωρίς τροποποιήσεις και στο πραγματικό δίκτυο, όπως για παράδειγμα ένας Chord ή CAN client. Επιπλέον, εκτός από την αναζήτηση, ο εκάστοτε αλγόριθμος θα πρέπει να μπορεί να μελετηθεί και ως προς τις υπόλοιπες λειτουργίες του, όπως για παράδειγμα τη Stabilize ή τη Join.

Επίσης, μπορεί να μοντελοποιηθεί η συμπεριφορά «κακών» κόμβων οι οποίοι θα συνδέονται στο δίκτυο, προκειμένου να μελετηθεί η ανθεκτικότητα του εκάστοτε αλγορίθμου σε διάφορες μορφές επιθέσεων (π. χ. Denial of Service Attacks).

Τέλος, θα μπορούσαν να γίνουν τροποποιήσεις ώστε να μετατραπεί σε προσομοίωση διακριτού χρόνου, όπως αυτή περιγράφηκε στην Ενότητα 2.

6. Παράρτημα Α: Interfaces

```
/*
Provides the prototype functions that will be used to communicate between clients.
*/
public interface IP2PMessaging {

/*
Send a message to the designated destination using a reliable protocol.
@param destination - The name of the destination node
@param data - The bytes to send
*/
public void sendMessage(IAddress sender, IAddress destination, byte[] data) throws
URISyntaxException;

/*
Send a message to the designated destination using an unreliable protocol.
@param destination - The name of the destination node
@param data - The bytes to send
*/

public void sendMessageU(IAddress sender, IAddress destination, byte[] data) throws
URISyntaxException;

/*
Registers a client name for receiving messages
@param client - The client fingerprint
*/
public void registerClient(IClient client);

/*
Unregisters a previously registered client
@param client - The client fingerprint
*/
public void unregisterClient(IClient client);

/*
Returns the list of registered clients that can receive requests
@return - The names of the clients
*/
public String[] getClientNames();
/*
```

```
Adds another name for the Messaging Server. This is typically used to define multiple
names for the server, such as "localhost", "127.0.0.1" or "www.msgsrv.com". This
information is only used for requests coming from the local clients and not by requests
arriving through the socket interface.
@param name - The name to add
*/
public void addServerName(String name);

/*
Retrieves all the Messaging Server names
@return - The server names
*/
public String[] getServerNames();

/*
Returns the address of the client
@param - the client
@return - The address of the client
*/
public IAddress getClientAddress(IClient client);


/*
Retrieves the Messaging Server port
@return - The server port
*/
public int getServerPort();

/*
Retrieves the Messaging Server public name. Other Messaging
Servers should use this name to connect to this over the Internet.
@return - The server public name
*/
public String getServerPublicName();

}


/*
Provides the functions that clients must implement in order to participate in the p2p
network.
*/
public interface IP2PMessageReceiver {

/*
This method is called whenever a message is received for the node.
```

```
@param data - The received data
*/
public void getMessage(byte[] data);

}


/*
This interface provides the core functions required for accessing p2p ids in an algorithm-independent way.
*/
public interface IId {

/*
Converts an id to a series of bytes to be used for serialization purposes.
@return the serialized format of the id
*/
public byte[] toBytes();


/*
Converts an id to a String representation to be used for display/serialization purposes
@return the serialized format of the id
*/
public String toString();

/*
Converts a byte array to an Id, performing the reverse operation compared to the toBytes method.
@param data - the byte array
@return the corresponding Id
@throws Exception - when the byte array cannot be properly converted
*/
public IId toId(byte[] data) throws Exception;

/*
Converts a String to an Id, performing the reverse operation compared to the toString method.
@param data - the string representation
@return the corresponding Id
@throws Exception - when the string cannot be properly converted
*/
public IId toId(String data) throws Exception;

/*
Converts a byte array to an Id by hashing the
```

```
provided data.
@param data - the byte array
@return the corresponding Id
*/
public IId hash(byte[] data);

/*
Converts a string to an Id by hashing the provided data.
@param data - the byte array
@return the corresponding Id
*/
public IId hash(String st);

public int toInt() throws Exception ;

}


/*
Factory pattern for client creation
*/
public interface IClientFactory {

/*
Creates a new Client instance, with a local name, and connects the instance to a
messaging server. The messaging server is then responsible for composing the address of
the client by using messaging server specific data and the local name. The local name
may be something like 'myclient' or something like a port number (e.g. '1234'). The exact
convention is implementation specific.
@param localname - the local name/port used by the client
@param ms - the messaging server
@return an instance of the newly created client
*/
public IClient newInstance(String localname, IP2PMessaging ms);

}


/*
This interface provides an abstraction of the generic DHT client functionality.
*/
public interface IClient extends IP2PMessageReceiver {

/*
```

```
Connects the client to a given p2p network. This is done through one or more other
nodes, whose addresses are calculated using other means. If the array is empty or null, the
node forms a new, initial, standalone p2p network by itself.
@param addr - the array of addresses used for bootstrapping
@throws Exception when the node failed to bootstrap for the given addresses.
*/
public void bootstrap(IAddress[] addr) throws Exception;

/*
The primary operation offered by p2p clients. This function returns an array of addresses
which are capable of providing the content associated with the given id.
@param id - the id of the requested content
@return an array of addresses for the available nodes. This is an empty array if none
found
*/
public IAddress[] lookup(IId id);

/*
Looks up for the id in the client's FileList.
@param id
@return true if found
*/
public boolean find(IId id);

/*
Notifies the client that a specific content is now available for lookups by other nodes.
Invoked by the end user application.
@param id
*/
public void addContent(IId id);

/*
Notifies the client that a specific content is no longer available for lookups by other
nodes. Invoked by the end user application.
@param id
*/
public void removeContent(IId id);

/*
Get the Id assigned to the client
@return the client id
*/
public IId getId();

/*
```

```
Local name of the client. This is the name that is used for registering the client. Each client must have a distinct name.
@return - The name of the client
*/
public String getLocalName();

/*
 Full address of the client. This is the name that will be used for providing the client with a network-wide unique name.
@return - The address of the client
*/
public IAddress getAddress();

}


/*
Factory interface for address generation
*/
public interface IAddressFactory {
/*

Converts a byte array to an address, performing the reverse operation compared to the toBytes method.
@param data - the byte array
@return the corresponding Id
@throws Exception - when the byte array cannot be properly converted
*/
public IAddress toAddress(byte[] data) throws Exception;

/*
Converts a String to an address, performing the reverse operation compared to the toString method.
@param data - the string representation
@return the corresponding Id
@throws Exception - when the string cannot be properly converted
*/
public IAddress toAddress(String data) throws Exception;

}


import java.io.Serializable;
/*
Provides an abstraction of a peer address
*/
```

```
public interface IAddress extends Serializable{

/*
Converts an address to a series of bytes to be used for serialization purposes
@return the serialized format of the address
*/
public byte[] toBytes();

/*
Converts an address to a String representation to be used for display/serialization purposes
@return the string format of the address
*/
public String toString();

}
```

7. Παράρτημα Β: Χρήσιμοι Ορισμοί

Overlay Network: Πρόκειται για ένα εικονικό δίκτυο που σχηματίζεται επάνω από ένα υπάρχον .

Μοντέλο επικοινωνίας Client-Server: Σε ένα μοντέλο επικοινωνίας Client-Server υπάρχει διάκριση μεταξύ των υπολογιστών που διαθέτουν τους πόρους (Server) του δικτύου και των υπολογιστών που κάνουν χρήση αυτών των πόρων (Client). Δηλαδή, σε ένα δίκτυο που ακολουθεί αυτό το μοντέλο, η διαχείριση και ο έλεγχος όλων των διαθέσιμων πόρων όπως αρχεία, κατάλογοι, εφαρμογές και κοινόχρηστες συσκευές γίνεται κεντρικά από τους Servers ενώ οι Clients ζητούν και προσπελαύνουν τους πόρους αυτούς.

Bottleneck (συμφόρηση): Έτσι ονομάζεται το σημείο στο δίκτυο όπου ο ρυθμός μετάδοσης έχει περιοριστεί λόγω αυξημένης κίνησης.

DHT: Ένας κατανεμημένος πίνακας κατακερματισμού (DHT) κάνει ότι κάνει και ένας πίνακας κατακερματισμού, με τη διαφορά ότι ο πίνακας δε βρίσκεται σε έναν κόμβο αλλά είναι κατανεμημένος σε όλους τους κόμβους του δικτύου. Δηλαδή, σε κάθε κόμβο αποθηκεύονται ζευγάρια (κλειδί, τιμή). Η αναζήτηση μιας τιμής γίνεται με βάση ένα κλειδί.

Broadcast: Μια μετάδοση που στέλνεται σε όλους τους κόμβους ενός δικτύου.

Τοπολογία power law: Σε μια power law τοπολογία, οι κόμβοι που συνδέονται στο δίκτυο έχουν την τάση να συνδέονται με κόμβους με μεγάλο βαθμό.(Βαθμός ενός κόμβου είναι ο αριθμός των κόμβων με τους οποίους συνδέεται)

Τοπολογία Πλέγματος (Grid): Σε μία κανονική τοπολογία πλέγματος (regular grid topology) ένας κόμβος συνδέεται με δύο γειτονικούς κόμβους. Σε περίπτωση που έχουμε περισσότερες από μία διαστάσεις συνδέεται με δύο κόμβους σε κάθε διάσταση. Αν έχουμε μία διάσταση και οι κόμβοι συνδέονται κυκλικά τότε η τοπολογία πλέγματος ονομάζεται ειδικά *δακτύλιος*. Αν έχουμε περισσότερες διαστάσεις, και οι κόμβοι συνδέονται κυκλικά σε κάθε μια από αυτές, σχηματίζεται ένα *torus* και η τοπολογία ονομάζεται toroidal [17].

SHA-1 (Secure Hash Algorithm - 1): Αλγόριθμος που μετατρέπει ένα μήνυμα μέγιστου μήκους 2^{64} bits σε μία «σύνοψη» του μηνύματος (message digest) μήκους 160 bits.

Denial Of Service Attack [16]**:** Σε μια τέτοια επίθεση ένας υπολογιστής/ χρήστης του δικτύου δεν μπορεί να κάνει χρήση κάποιων υπηρεσιών (π. χ. email, internet). Μια τέτοια επίθεση, μπορεί να έχει ως στόχο ένα δίκτυο ή ένα λειτουργικό σύστημα. Υπάρχουν πολλοί τρόποι για την πραγματοποίηση μιας τέτοιας επίθεσης. Για παράδειγμα, μπορεί κάποιος να προκαλέσει μεγάλη κίνηση στο δίκτυο, να καταναλώνει όλο το bandwidth ή άλλους πόρους του «θύματος».

8. Αναφορές

[1] Simulating a File – Sharing P2P Network - Mario T. Schlosser, Tyson E. Condie, and Sepandar D. Kamvar. *First Workshop on Semantics in P2P and Grid Computing*, December, 2002.
[2] A Measurement Study of Peer – to – Peer Sharing Systems. by Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19)*, Bolton Landing, New York, October 2003.
[3] Peer – to – Peer Traffic Characteristics - Rüdiger Schollmeier, Antoine Dumanois. *Proceedings of the 9th EUNICE Open European Summer School (EUNICE'03)*. Budapest, September 2003.
[4] The Gnutella Specifications: http://www.clip2.com
[5] Mapping the Gnutella Network: Properties of a large scale Peer-to-Peer Systems and Implications for System Design – Matei Ripeanu, Ian Foster, Adriana Iamnitchi. *IEEE Internet Computing Journal, 6(1),* 2002.
[6] Gnutella2 Developer Network: http://www.gnutella2.com/index.php/Main_Page
[7] Looking up data in P2P Systems – Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica. *Communications of the ACM*, February 2003.
[8] Chord: A Scalable Peer-to-Peer Lookup Service for Internet - Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan. *Proceedings of theACM SIGCOMM Conference*, 2001.
[9] A Scalable Content- Addressable Network Applications - Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp. *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications,* 2001.
[10] Analyzing peer- to peer traffic across large networks - Subhabrata Sen and Jia Wang. *IEEE/ACM Transactions on Networking (TON),* 2004.
[11] Σημειώσεις μαθήματος «Διανομή Περιεχομένου στο Διαδίκτυο» – Παναγιώτης Τριανταφύλλου.
[12] Decentralized Peer-to-Peer Network Architecture: Gnutella and Freenet. IEEE Internet Computing, vol 6(1), January-February, 2002.
[13] A Quantitative Analysis of the Gnutella Network Traffic - Demetris Zeinalipour and Theodoros Folias
[14] Experimenting with Gnutella communities - Jean Vaucher, Peter Kropf, Gilbert Babin, and Thierry Jouve. *In Conference on Distributed Communities on the Web*, 2002.
[15] Σημειώσεις Μαθήματος «Προσομοίωση Πληροφοριακών Συστημάτων», Γ. Γαροφαλάκη.
[16] http://www.darwinmag.com
[17] http://www.fact-index.com
[18] Open Problems in Data Sharing Peer-To-Peer Systems - Hector Garcia-Molina. *ICDT,* 2003
[19] 3LS - A Peer-to-Peer Network Simulator**,** Nyik San Ting, Ralph Deters. *Third International Conference on Peer-to-Peer Computing p. 212*, 2003.
[20] P2PSIM: Home: http://www.pdos.lcs.mit.edu/p2psim/

[21] Mapping the behavior to packet-Level Details: A framework for Packet-level Simulation of Peer-to-Peer Systems – Qi He, Mostafa Ammar, George Riley, Himanshu Raj, Richard Fujimoto. *11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems p. 71*, 2003.