Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή εργασία

Instant Messenger πάνω από ομότιμο δίκτυο.

Τσακνής Χρήστος (ΑΜ: 835)

Ηράκλειο – Νοέμβριος 2010

Επόπτης Καθηγητής: Φραγκοπούλου Παρασκευή

Κατάλογος περιεχομένων

Πρόλογος

Η παρούσα πτυχιακή εκπονήθηκε κατά το ακαδημαϊκό έτος 2009 – 2010 με επιβλέποντα καθηγητή του τμήματος Εφαρμοσμένης Πληροφορικής και Πολυμέσων του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Κρήτης, κυρίας Φραγκοπούλου Παρασκευής. Ιδιαίτερη προσοχή δόθηκε, εκτός του παρόντος κειμένου, στην εφαρμογή που δημιουργήθηκε παράλληλα με αυτό και εφαρμόζει πολλές από της τεχνικές που περιγράφονται εντός αυτού για την επίτευξη του στόχου της πτυχιακής που είναι να παράγουμε μία εφαρμογή ανταλλαγής άμεσων μηνυμάτων η οποία θα ανήκει σε κόμβο ομότιμου δικτύου που θα επικοινωνεί με τούς υπόλοιπους μέσω αυτού και θα αξιοποιεί της δυνατότητες επικοινωνίας που του παρέχει. Η εφαρμογή υπάρχει στο συνοδευτικό CD και συμπληρώνει την πτυχιακή εργασία.

Διάρθρωση της εργασίας

Τα θέματα που θα αναλύσουμε στην πτυχιακή αυτή ακολουθούν ένα σταθερό μεταβατικό στάδιο από ένα θεωρητικό προς ένα πιο τεχνικό κομμάτι με κατάληξη στην περιγραφή της τελικής εφαρμογής που εφαρμόζει τις τεχνικές που αναλύθηκαν καθώς και στα συμπέρασμα και κριτική των βιβλιοθηκών που χρησιμοποιήθηκαν για την παραγωγή του αλλά και μία εκτίμηση των λύσεων των προβλημάτων που προσπαθήσαμε να επιλύσουμε με αυτό. Αναλυτικότερα, αρχίζουμε με το θεωρητικό υπόβαθρο και το τι είναι ένα ομότιμο δίκτυο , ποιες αρχές και ποια κριτήρια πρέπει να ικανοποιεί για να χαρακτηριστεί ως τέτοιο καθώς και την μετάβαση στους κατανεμημένους πίνακες κατακερματισμού. Στην συνέχεια αναλύουμε το θεωρητικό κομμάτι της Pastry, του ομότιμου δικτύου επικάλυψης που θα χρησιμοποιήσουμε, τις εφαρμογές PAST και SCRIBE που την πλαισιώνουν και στο επόμενο στάδιο την τεχνική ανάλυση με παραδείγματα της FreePastry, του ελεύθερου λογισμικού πού υλοποιεί το πρότυπο της Pastry στην γλώσσα προγραμματισμού της Java. Το τελικό στάδιο της πτυχιακής αναφέρεται στην εφαρμογή και τα χαρακτηριστικά της καθώς και στο πρωτόκολλο επικοινωνίας που έχει φτιαχτεί πάνω από την FreePastry και που χρησιμοποιεί καθώς και περιγραφή των χαρακτηριστικών επικοινωνίας και των λειτουργιών που προσφέρει στους χρήστες.

Με βάση αυτή τη θεματική μετάβαση τα θέματα που θα διαπραγματευτούμε ανά κεφάλαιο περιγραφικά είναι:

- **Κεφάλαιο 1:** Στο πρώτο κεφάλαιο θα μιλήσουμε για το τι είναι ένα ένα ομότιμο δίκτυο , με ποια κριτήρια το κατατάσσουμε στον χαρακτηρισμό αυτό και και τι πλεονεκτήματα προσφέρουν σε σχέση με τα καθιερωμένα μοντέλα δικτύωσης. Επίσης γίνεται αναφορά και περιγραφή λειτουργίας από δύο δημοφιλής υπηρεσίες ανταλλαγής άμεσων μηνυμάτων, το Skype και τον Windows Live Messenger.
- **Κεφάλαιο 2:** Στο δεύτερο κεφάλαιο θα ρίξουμε μια θεωρητική ματιά στο πρότυπο δικτύου επικάλυψης (overlay network) της Pastry που επιλέξαμε για να κατασκευάσουμε την τελική εφαρμογή επεξηγώντας τα χαρακτηριστικά της και κάνοντας μία ανάλυση στις δύο εφαρμογές που την συνοδεύουν. Η μία είναι ο PAST και είναι ο κατανεμημένος πίνακας κατακερματισμού (Distributed Hash Table) που χρησιμοποιεί και η δεύτερη το SCRIBE που είναι ένα σύστημα δημοσίευσης/συνδρομής (Publish/Subscribe) με multicasting και anycasting δυνατότητες.
- **Κεφάλαιο 3**: Το τρίτο κεφάλαιο είναι μία φυσική συνέχεια του δευτέρου με την διαφορά ότι από το θεωρητικό κομμάτι της Pastry περνάμε σε καθαρά τεχνικής φύσεως ζητήματα περιγράφοντας και παραθέτοντας σύντομα παραδείγματα της FreePastry , της βιβλιοθήκης σε Java που υλοποιεί την Pastry και των διαδικασιών που περιγράφτηκαν στο δεύτερο κεφάλαιο.
- **Κεφάλαιο 4:** Σε αυτό το κεφάλαιο παρουσιάζεται η εφαρμογή FreePastry DHT Messenger, η εφαρμογή που κατασκευάστηκε σαν αποτέλεσμα της μελέτης που έγινε στα προηγούμενα κεφάλαια. Αναλύουμε τις λειτουργίες που παρέχει αλλά και πως αξιοποιείται η FreePastry για την παροχή αυτών των υπηρεσιών.

Εισαγωγή

Οι υπηρεσίες ανταλλαγής άμεσων μηνυμάτων έχουν αρκετά χρόνια έχουν εμφανιστεί. Στην αρχή βέβαια αποτελούσαν την επικοινωνία μεταξύ εφαρμογών υπολογιστών και βοηθούσαν στις υπηρεσίες μεταξύ τους αλλά στην πορεία άρχισαν δειλά δειλά να παίρνουν την μορφή ξέρουμε σήμερα. Σε αυτό συνετέλεσε το γεγονός ότι προσέφεραν γρήγορο,άμεσο, φθηνό άλλα κάπως περιορισμένων δυνατοτήτων είδος επικοινωνίας από το άλλο κυρίαρχο μέσο σε αυτήν την κατηγορία, τις τηλεφωνικές επικοινωνίες. Το κυρίαρχο μοντέλο δικτύωσης ήταν αυτό του πελάτη - εξυπηρετητή (client-server)με ορισμένες εξαιρέσεις πού χρησιμοποίησαν ομότιμη τεχνολογία (πχ talk, ntalk and ytalk) [9].

Από την άλλη τα κατανεμημένα (ομότιμα) δίκτυα λόγω της φύσεως τους να προσδίδουν σε ένα δίκτυο πόρους όπως αποθηκευτικές , επεξεργαστικές δυνατότητες αλλά και εύρος δικτύου χρησιμοποιήθηκαν ευρέως στην ανταλλαγή αρχείων και μετά την υπόθεση του Napster [7, 10], ενός κατανεμημένου συστήματος ανταλλαγής αρχείων, έγιναν ιδιαίτερες προσπάθειές για την εξέλιξη τους με αποτέλεσμα να φτάσουμε στο μοντέλο ομότιμων συστημάτων που χρησιμοποιούν κατανεμημένους πίνακες κατακερματισμού.

Το γεγονός όμως ότι η τεχνολογία προχωρούσε και μαζί της ανέβαιναν και οι απαιτήσεις για νέες υπηρεσίες που θα πρόσδιδαν στην εμπειρία χρήσης των χρηστών αλλά και την διευκόλυνση τους προσφέροντας τους νέα χαρακτηριστικά, αυτές οι δύο τεχνολογίες ολοένα και έρχονταν πιο κοντά.

Η μεγάλη ευκαιρία για αυτές για να συνεργαστούν ήταν όταν οι ταχύτητες του Διαδικτύου ήταν αρκετά γρήγορες για να μεταφέρουν βίντεο και ήχο σε ικανοποιητική ποιότητα. Το μοντέλο πελάτη – εξυπηρετητή κρίθηκε ανεπαρκές για τους λόγους ότι δεν ήταν ανθεκτικό και είχε περιορισμένο εύρος δικτύου για αυτού του είδους υπηρεσίες. Στην καλύτερη των περιπτώσεων , ήταν απλά ακριβό. Οπότε η ευκαιρία για τα ομότιμα δίκτυα είχε έρθει. Προσέφεραν όλα όσα δεν μπορούσε το μοντέλο πελάτη εξυπηρετητή με κύρια χαρακτηριστικά την ανθεκτικότητα , την κλιμάκωση την διαθεσιμότητα και το εύρος δικτύου και όλα αυτά οικονομικότερα.

Σήμερα δύο από τις δημοφιλέστερες υπηρεσίες ανταλλαγής μηνυμάτων που υλοποίησαν τις παραπάνω δυνατότητες μέσω της ομότιμης τεχνολογίας σχεδόν εξολοκλήρου ή για για μέρος των υπηρεσιών αυτών είναι το Skype και ο Windows Live Messenger. Ο τρόπος λειτουργίας τους περιγράφεται στο τέλος του πρώτου κεφαλαίου της πτυχιακής.

Ποίες είναι όμως οι δυσκολίες που αντιμετωπίζουν τα ομότιμα δίκτυα όταν υλοποιούν υπηρεσίες στις οποίες πρέπει να κάποιος χρήστης να εντοπίζει κάποιον άλλο που μπορεί να βρίσκεται σε οποιονδήποτε υπολογιστή του δικτύου; Πώς μπορεί ο ένας να ενημερώνεται για την παρουσία του άλλου και αυτοί οι δύο να επικοινωνούν με επιτυχία; Αυτό ονομάζεται “Rendezvous problem” (πρόβλημα συνάντησης) και αποτελεί τον στόχο της πτυχιακής και σε συνδυασμό με μία λύση του το χτίσιμο μίας ικανοποιητικής και πλήρους λειτουργικής υπηρεσίας ανταλλαγής άμεσων μηνυμάτων.

Κεφάλαιο 1 - Δίκτυα ομότιμων κόμβων

1.1 Τα δίκτυα ομότιμων κόμβων (Peer-to-Peer Networks)

Τα δίκτυα ομότιμων κόμβων ικανοποιούν τον ορισμό των δικτύων επικοινωνιών: "*Ένα δίκτυο υπολογιστών (computer network), ορίζεται σαν σύνολο από αυτόνομους υπολογιστές οι οποίοι είναι συνδεδεμένοι μεταξύ τους με μία συγκεκριμένη τεχνολογία"*[7], μπορούν δηλαδή να επικοινωνήσουν και να ανταλλάξουν πληροφορίες μεταξύ τους. Για να δώσουμε έμφαση στο μοντέλο των ομότιμων δικτύων και να αναδείξουμε την διαφορετικότητα τους είναι καλό να το αντιπαραβάλλουμε με ένα από τα επικρατέστερα μοντέλα δικτύωσης που καθιερώθηκαν μέχρι σήμερα , το μοντέλο πελάτη – εξυπηρετητή (Client-Server).

1.2 Το μοντέλο πελάτη – εξυπηρετητή και το μοντέλο ομότιμων κόμβων

Ένα από τα κεντρικά μοντέλα δικτύωσης που συναντάμε πολύ συχνά είναι το μοντέλο πελάτη – εξυπηρετητή (Client-Server Model) [7]. Οι περισσότερες σημερινές εφαρμογές χρησιμοποιούν αυτό το μοντέλο καθώς και το πρωτόκολλο διαδικτύου. Συγκεκριμένα μια διαδικτυακή εφαρμογή ξεχωρίζει τα επιμέρους μέρη της που είναι συνδεδεμένα σε αυτή σε πελάτες και εξυπηρετητές. Ο πελάτης στέλνει μια αίτηση την οποία αναλαμβάνει να διεκπεραιώσει ο εξυπηρετητής. Ο εξυπηρετητής λαμβάνει την αίτηση , την επεξεργάζεται και επιστρέφει τα αποτελέσματα της αίτηση αυτής πίσω στον πελάτη. Καθιερώνεται έτσι μια αυστηρή ιεραρχία στα στοιχεία που αποτελούν το δίκτυο και του επιμέρους ρόλου τους.

Τα δίκτυα ομότιμων κόμβων[7] δεν ακολουθούν την νοοτροπία αυτή του αυστηρού μοντέλου πελάτη – εξυπηρετητή και της διαδικασίας αίτησης , επεξεργασίας και αποστολής πίσω του αποτελέσματος της αίτηση που στάλθηκε δηλ. ενός δικτύου με προκαθορισμένες συμπεριφορές του καθενός μέρους του και δεν επιβάλουν οπωσδήποτε ότι ο ένας θα είναι αυστηρά και μόνο ο πελάτης και ο άλλος ο εξυπηρετητής . Η αρχιτεκτονική και ο σχεδιασμός τους τους δεν κατατάσσει τους κόμβους σε κατηγορίες που θα διαχωρίζονται σε πελάτες και εξυπηρετητές αλλά θεωρεί ότι οποιοσδήποτε κόμβος του δικτύου θα μπορεί να εναλλάσσεται μεταξύ αυτών των δύο ρόλων. Έτσι αφού οι κόμβοι αποκτούν μία "ισοτιμία" ο ένας απέναντι στον άλλο και για τον λόγο αυτό στα δίκτυα αυτά δόθηκε η ονομασία ομότιμα.

Λόγω αυτής της κατανεμημένης φύσης που έχουν τα δίκτυα ομότιμων κόμβων και σύμφωνα λοιπόν με τον Tanenbaum: "ένα κατανεμημένο σύστημα μπορεί να οριστεί σαν *ένα σύνολο*

αυτόνομων υπολογιστών το οποίο παρουσιάζεται στους χρήστες του σαν ένα ενιαίο σύστημα"[7]. Αυτή είναι και η βασική διαφορά ενός κατανεμημένου συστήματος από ένα δίκτυο υπολογιστών , ότι στην δεύτερη περίπτωση η μη ενιαία εικόνα και ή έκθεση των επιμέρους στοιχείων του δικτύου γίνεται αντιληπτή.

Είδαμε το πώς ορίζεται ένα ομότιμο σύστημα σε σχέση με τη διαφάνεια ενός υπολογιστικού συστήματος . Ένας ορισμός ως προς την ισότητα (ομοτιμία) που περιγράψαμε προηγμένος θα ήταν:

"*Ένα ομότιμο δίκτυο ορίζεται σαν ένα σύνολο υπολογιστών συνδεδεμένων μεταξύ τους οι οποίοι έχουν ίσες υποχρεώσεις αλλά και δικαιώματα ο ένας απέναντι στον άλλο*".

Σε αυτό το σημείο θα ήταν χρήσιμο να αναφέρουμε και έναν ακόμη ορισμό ο οποίος μας είναι αναγκαίος για την συνεχεία. Αυτόν των δικτύων επικάλυψης (overlay networks) [7, 10] καθώς με αυτόν μπορούμε να προσδιορίσουμε και τα δίκτυα ομότιμων κόμβων. Ένα δίκτυο επικάλυψης είναι ένα τεχνητό δίκτυο σε επίπεδο εφαρμογής με λογικές ζεύξεις το οποίο δομείται πάνω από ένα υπάρχον δίκτυο. Μπορούμε να πούμε ότι τα περισσότερα δίκτυα ομότιμων κόμβων είναι δίκτυα επικάλυψης γιατί προσφέρουν τις υπηρεσίες τους πάνω από υπάρχον δίκτυο είτε αυτό είναι κάποιο τοπικό δίκτυο η το Ιντερνέτ. Θα δούμε σε επόμενο κεφάλαιο ότι ένα τέτοιο είναι και η FreePastry την οποία χρησιμοποιούμε ως overlay network με οποίο θα επικοινωνεί το πρωτότυπο εφαρμογής Instant Messenger που κατασκευάστηκε ως μέρος την πτυχιακής.

1.3 Η εξάπλωση των δικτύων ομότιμων κόμβων

Ένα εκ των πρώτων, ευρείας χρήσης, δικτύων ομότιμων κόμβων είναι αυτό του Usenet [14] το οποίο βρίσκεται ακόμη σε λειτουργία. Το Usenet είναι ένα κατανεμημένο δίκτυο στο οποίο οι χρήστες του μπορούν να δημοσιεύσουν οι ίδιοι ή να διαβάζουν άρθρα άλλων χρηστών . Τα άρθρα αυτά κατατάσσονται σε κατηγορίες (newsgoups). Η ουσιαστική όμως επανάσταση τους ήρθε το 1999 και το δίκτυο του Napster [7, 12, 13]. Από τότε, πολλά γνωστά συστήματα, όπως το KaZaA, το Napster, το Gnutella , το Limewire αλλά και το Bit Torrent καθιερώθηκαν σαν δίκτυα διαμοιρασμού δεδομένων (File Sharing Systems), δημιουργώντας αναστάτωση στην μουσική βιομηχανία και στη συνέχεια και στην βιομηχανία παραγωγής ταινιών καθώς οι ταχύτητες downloading με την καθιέρωση της ADSL αυξήθηκαν θεαματικά και σε συνδυασμό με την πρόσβαση σε μεγάλα όγκο δεδομένων που προσέφεραν η απόκτηση παράνομου υλικού εξαπλώθηκε. Οι χρήστες των δικτύων ομότιμων μπορούσαν να διαμοιραστούν μεταξύ τους αρχεία μουσικής αλλά και ταινίες χωρίς να έχουν τα πνευματικά δικαιώματα για την διακίνηση των αρχείων. Από την στιγμή της διάδοσης τους και μετά ακολούθησαν χιλιάδες μηνύσεις κατά συστημάτων αλλά και χρηστών, με πιο γνωστές τις μηνύσεις κατά του Napster που και αν και ο δημιουργός του Shawn Fanning υποστήριξε ότι από την στιγμή που δεν αποθηκεύει το σύστημα δεδομένα τοπικά σε κάποιον server δεν λειτουργούσε με παράνομο τρόπο και η ευθύνη έπεφτε στην χρήση του από τον κάθε χρήστη ξεχωριστά. Το δικαστήριο όμως δεν πείστηκε και με την πίεση των δισκογραφικών εταιριών κυρίως αναγκάστηκε να διακόψει τις υπηρεσίες του το 2001.

1.4 Τα πλεονεκτίματα των δικτύων ομότιμων κόμβων

Τα δίκτυα ομότιμων κόμβων έχουν κάποια στοιχεία [5] τα οποία τα έχουν κάνει δημοφιλή. Οι κυρίως λόγοι για την ανάγκη χρησιμοποίηση τους είναι οι παρακάτω:

- Διάφορες δυσκολίες , περιορισμοί , οικονομικές απαιτήσεις αλλά και ο περιορισμένος αριθμός ανθρωπίνου δυναμικού, είναι χαμηλές αφού δεν απαιτούν ειδικές διοικητικές ή οικονομικές επενδύσεις όπως συμβαίνει αντίθετα αντίθετα με τα κεντρικοποιημένα συστήματα.(π.χ. Client - Server Μοντέλο)

- Οι υπολογιστές που συμμετέχουν στο δίκτυο ομότιμων κόμβων προσφέρουν μία ολοένα και θετικά κλιμακούμενη (scalability) αποθηκευτική αλλά και επεξεργαστική διαθεσιμότητα.

- Λόγω της κατανεμημένης και αποκεντρωμένης φύσεως έχουν την δυνατότητα για ευρωστία (robustness) να είναι ανθεκτικότερα σφάλματα, αποτυχίες αλλά και σε κακόβουλες επιθέσεις. Όπως επίσης την συνέχεια παροχής των υπηρεσιών του δικτύου ακόμα και σε περίπτωση αποτυχίας μεγάλου αριθμού των κόμβων που το αποτελούν.

1.5 Τα κριτήρια τα οποία θα πρέπει να ικανοποιούν τα δίκτυα ομότιμων κόμβων

Για να χαρακτηρίσουμε ένα δίκτυο ως ομότιμο θα πρέπει να ικανοποιεί κάποιες αρχές σύμφωνα με τους Karl Aberer και Manfred Hauswirth [1] αυτές είναι:

- *Η αρχή του διαμοιρασμού πόρων*: Όλα τα δίκτυα ομότιμων περιλαμβάνουν την έννοια του διαμοιρασμού. Οι πόροι που διαμοιράζονται μπορεί να είναι φυσικοί, όπως αποθηκευτικός χώρος σε ένα σκληρό δίσκο ή κάποιο άλλο αποθηκευτικό μέσο, το εύρος ζώνης κάποιου δικτύου, ή μέρος από την επεξεργαστική ισχύ κάποιου κόμβου. Με βάση αυτή την αρχή, μπορούν να υλοποιηθούν εφαρμογές οι οποίες θα ήταν αδύνατο να υλοποιηθούν πάνω από ένα και μόνο υπολογιστή. Επίσης η ιδέα αυτή αποτέλεσε την κινητήρια δύναμη πίσω από τη δημιουργία του Napster.

- *Η αρχή της αποκέντρωσης*: Η αρχή αυτή αποτελεί συνέπεια της αρχής διαμοιρασμού πόρων. Μέρη του συστήματος, ή και ολόκληρο το σύστημα καθίσταται κάτω από κάποιο από κεντρικό έλεγχο. Έτσι, αποφεύγονται πιθανές περιοχές συμφόρησης στο δίκτυο (bottle nects) , στοιχείο το οποίο προσδίδει στην ανθεκτικότητά (robustness) του συστήματος (έλεγχος δεν εξαρτάται από ένα και μοναδικό κόμβο και δεν είναι κεντρικός). Πλήρως αποκεντρωμένα συστήματα, αποτελούν το Gnutella και το Freenet.

- *Η αρχή της αυτό οργάνωσης*: Όταν ένα σύστημα είναι πλήρως αποκεντρωμένο, δεν υπάρχει κεντρικό σημείο έλεγχου. Αυτό, σημαίνει ότι κάθε κόμβος θα πρέπει να οργανώνει και να συντονίζει τη συμπεριφορά του, βασιζόμενος στις εσωτερικές πληροφορίες που κατέχει, καθώς και στις πληροφορίες που μπορεί να ανακτήσει από μια γειτονία κόμβων. Η

συμπεριφορά του συστήματος μπορούμε να πούμε είναι συνάρτηση της αλληλόσυνεργαζόμενης συμπεριφοράς των κόμβων που το αποτελούν.

1.6 Οι γενιές των δικτύων ομότιμων κόμβων

Τα ομότιμα δίκτυα μπορούν να χωριστούν σε κατηγορίες ανάλογα το χρονικό στάδιο της εξέλιξης τους και τον αναγκών ύπαρξης τους την εκάστοτε δεδομένη χρονική στιγμή. Ως τώρα μπορούμε να διακρίνουμε 4 κατηγορίες [19] από αυτά.

1.6.1 Πρώτη γενιά

Η πρώτη γενιά δικτύων ομότιμων κόμβων είχε κεντρικό έλεγχο από εξυπηρετητές. Ο κεντρικός έλεγχος αυτός συντόνιζε την κίνηση μεταξύ των χρηστών. Στους εξυπηρετητές αποθηκεύονταν λίστες από αρχεία των χρηστών και ανανεώνονταν όταν αυτοί ξανά εισέρχονταν στο σύστημα. Στο κεντρικοποιημένο αυτό δίκτυο ομότιμων κόμβων ένας χρήστης έστελνε την αίτηση του για το αρχείο πού έψαχνε στον εξυπηρετητή και αυτός τού έστελνε πίσω μία λίστα με τους κόμβους πού το είχαν και μετά η επικοινωνία γινόταν μεταξύ τους για το κατέβασμα του αρχείου. Αυτό το client – server μοντέλο ήταν αρκετά αποδοτικό γιατί ο κεντρικός κατάλογος ανανεωνόταν συνεχώς και οι χρήστες έπρεπε να ήταν εγγεγραμμένοι για να χρησιμοποιήσουν το πρόγραμμα. Ωστόσο είχε ένα μοναδικό σημείο αποτυχίας (single point of failure) το οποίο μπορούσε να οδηγήσει σε κατάρρευση του συστήματος σε περίπτωση που ο εξυπηρετητής έβγαινε εκτός λειτουργίας. Επίσης μπορούσε να έχει εκπρόθεσμες πληροφορίες η σπασμένους συνδέσμους (links) αν ο εξυπηρετητής δεν ανανέωνε τις λίστες του σωστά.

Δύο παραδείγματα που χρησιμοποιούσαν αυτό το μοντέλο ήταν το Napster και το eDonkey.

1.6.2 Δεύτερη γενιά

Μετά τα νομικά προβλήματα που αντιμετώπισε το Napster, ο Justin Frankel της Nullsoft ξεκίνησε να δημιουργεί ένα ομότιμο δίκτυο κόμβων χωρίς την ύπαρξη κεντρικού εξυπηρετητή που να διατηρεί κατάλογους με τελικό αποτέλεσμα το Gnutella. Δυστυχώς το μοντέλο του Gnutella στο ποίο όλοι οι κόμβοι ήταν ίσοι αντιμετώπισε προβλήματα με σημεία συμφόρησης (bottlenecks) καθώς το δίκτυο μεγάλωνε. Το σύστημα FastTrack έλυσε αυτό το πρόβλημα παρέχοντας την δυνατότητα σε κάποιους κόμβους να είναι περισσότερο ίσοι από τους άλλους.

Αυτό γινόταν με το να επιλέγονται κάποιοι κόμβοι με μεγαλύτερη χωρητικότητα (σε αποθηκευτικό χώρο , εύρος δικτύου κτλπ) για να κρατάνε τις λίστες και με τους λιγότερης χωρητικότητας να συνδέονται σε αυτούς, παρέχοντας ένα δίκτυο το οποίο μπορούσε να εξαπλωθεί σε μεγαλύτερο μέγεθος. Το Gnutella γρήγορα υιοθέτησε και αυτό το ίδιο μοντέλο

Στην δεύτερη γενιά συμπεριλαμβάνεται και το μοντέλο των κατανεμημένων πινάκων κατακερματισμού (DHTs) οι οποίοι βοήθησαν στην αντιμετώπιση του προβλήματος της διαθεσιμότητας καθώς μέσω της αντιγραφής και διατήρησης αρχείων σε γειτονικούς κόμβους από κάποιον κόμβο η αναζήτηση γινόταν γρήγορα και αποτελεσματικά

1.6.3 Τρίτη γενιά

Η τρίτης γενιάς ομότιμων δικτύων είναι παρόμοια με την δεύτερη μόνο που εισαγάγει την έννοια της ανωνυμίας (anonymity).

Ένας βαθμός ανωνυμίας δημιουργείται με το να δρομολογείται ή κίνηση μας μέσω κόμβων άλλων χρηστών πράγμα που κάνει δύσκολο για κάποιον να προσδιορίσει ποιος κατεβάζει και ποιος προσφέρει αρχεία στο δίκτυο. Τα περισσότερα των προγραμμάτων αυτών έχουν δυνατή κρυπτογράφηση και προστασία από traffic sniffing (υποκλοπή κίνησης πακέτων).

Επίσης ένας ακόμα τύπος ομότιμων δικτύων τρίτης γενιάς που εμφανίστηκε είναι αυτός του φίλου προς φίλο (Friend to Friend) επικοινωνίας. Αυτό το μοντέλο επιτρέπει απευθείας σύνδεση για την την προώθηση αρχείων αιτήσεων των κόμβων των οποίων οι χρήστες γνωρίζονται μεταξύ τους και την επιτρέπουν ο ένας στον άλλο.

1.6.4 Τέταρτη γενιά

Πέρα από την παραδοσιακή λειτουργία της ανταλλαγής αρχείων υπάρχουν υπηρεσίας που στέλνουν ροές δεδομένων αντί για αρχεία εντός ενός ομότιμου δικτύου. Έτσι μπορεί κάποιος να ακούσει ράδιο ή να δει τηλεόραση χωρίς κάποιον εξυπηρετητή να παρεμβάλλεται. Ένα δημοφιλές παράδειγμα τέτοιου δικτύου είναι το Justin.tv (www.justin.tv) όπου κανείς μπορεί να δει online αθλητικά γεγονότα, ταινίες , διάφορες τηλεοπτικές εκπομπές καθώς και να δημιουργήσει το δικό του προσωπικό κανάλι στο οποίο μπορούν να συνδεθούν και να παρακολουθούσουν άλλοι χρήστες.

1.7 Καθαρά και υβριδικά δίκτυα ομότιμων κόμβων

Τα δίκτυα ομότιμων κόμβων διακρίνονται στα καθαρά (pure) και τα υβριδικά (hybrid) [15]. Στα καθαρά δίκτυα ομότιμων, δεν υπάρχει κανένας κεντρικός έλεγχος και όλοι οι κόμβοι παρέχουν και ζητούν της ίδιες υπηρεσίες από τους άλλους κόμβους του δικτύου (ομοτιμία). Το μοντέλο αυτό ανταποκρίνεται στον ορισμό των δικτύων ομότιμων κόμβων, θεωρώντας τον κάθε κόμβο μια αυτοτελή και ανεξάρτητη οντότητα του δικτύου. Ένα παραδείγματα αυτού του τύπου αποτελούν τα δίκτυα Gnutella και Freenet. Το υβριδικό μοντέλο ακολουθεί μία ελαφρώς διαφορετική φιλοσοφία και διαφοροποιείται ως εξής: Ένας κεντρικός κόμβος προσεγγίζεται σε αρχικό στάδιο έτσι ώστε να αντληθούν πληροφορίες . Αυτές οι πληροφορίες μπορεί να αφορούν θέματα ασφάλειας του

δικτύου, αλλά και πληροφορίες όπως σε ποιο κόμβο βρίσκεται αποθηκευμένη κάποια πληροφορία. Από εκείνο το σημείο εκείνο και μετά, το δίκτυο λειτουργεί σαν ένα καθαρό δίκτυο ομότιμων.

1.7.1 Το σύστημα Skype

Ό κεντρικοποιημένος αυτός έλεγχος, μπορεί να πάρει πολλές μορφές. Για παράδειγμα το Skype [16] το οποίο είναι ένα δημοφιλές σύστημα Instant Messaging το οποίο προσφέρει, υπηρεσίες VOIP (Voice Over IP) δυνατότητα αποστολής SMS, επίγειων τηλεφωνικών κλήσεων, αποστολής αρχείων καθώς και δυνατότητα Video συνδιάσκεψης.

Στο Skype συνυπάρχουν δύο διαφορετικές οντότητες, ένας απλός πελάτης (ordinary host ή αλλιώς Skype Client) και ο υπερκόμβος (super node). Ένας απλός πελάτης είναι ο υπολογιστής ενός χρήστη του Skype που έχει εγκαταστήσει την εφαρμογή και συνδέεται στο δίκτυο για να επικοινωνήσει με άλλους χρήστες. Οι υπερκόμβοι είναι τα σημεία εισαγωγής των clients στο δίκτυο στα και οποία συνδέονται για να γίνει εφικτή η επικοινωνία μεταξύ των χρηστών. Κάθε υπολογιστής με δημόσια IP διεύθυνση και κατάλληλο εξοπλισμό μπορεί να γίνει υπερκόμβος. Κάθε πελάτης πρέπει να συνδεθεί σε έναν υπερκόμβο και τα στοιχειά του χρήστη να ταυτοποιηθούν από τον Skype Login Server για να εισαχθεί η παρουσία του στο σύστημα. Ο Skype Login Server είναι το μόνο κεντρικό σύστημα σε όλο το δίκτυο. Μαζί με άλλα στοιχεία αποθηκεύει τα usernames και τα passwords των χρηστών. Όλοι οι υπερκόμβοι συνδέονται στον Skype Login Server και αυτό γίνεται για να προωθηθεί η αίτηση ταυτοποίησης των χρηστών που συνδέονται αρχικά σε αυτούς. Οι υπερκόμβοι είναι εξυπηρετητές διασκορπισμένοι σε διαφορετικά σημεία του πλανήτη αλλά ο πελάτης ενός χρήστη πρέπει να ξέρει σε ποιον υπερκόμβο να συνδεθεί . Για αυτό τον λόγο κάθε client διατηρεί τοπικά στην registry του συστήματος που είναι εγκατεστημένος του ένα πίνακα με τις IP διευθύνσεις και και τις αντίστοιχες θύρες κάθε υπερκόμβου. Κάθε φορά που κάποιος χρήστης τρέχει τον Skype πελάτη του πέρνει την πρώτη IP και θύρα και απο τον πίνακα και προσπαθεί να συνδεθεί στον υπερκόμβο με την συγκεκριμένη IP διευθύνση. Αν για οποιονδήποτε λόγο η σύνδεση αποτύχει (π.χ. Ο υπερκόμβος είναι εκτός λειτουργίας ή δεν είναι πια μέρος του δικτύου) τότε διαβάζει την επόμενη γραμμή του πίνακα και αν κανείς δεν είναι διαθέσιμος το Skype επιστρέφει ένα login error on start up. Οπότε για να μπορεί κάποιος να μπει στο δίκτυο του Skype θα πρέπει να υπάρχει τουλάχιστον μία εγγραφή που να αντιστοιχεί σε ενεργό και μη προβληματικό υπερκόμβο.

Σαν ιδέα , οι υπερκόμβοι εισήλθαν στα τρίτης γενιάς ομότιμα δίκτυα. Επιτρέπουν εξελιγμένη απόδοση στην αναζήτηση , μειωμένη καθυστέρηση στις μεταφορές αρχείων , την επεκτασιμότητα του δικτύου . και την ικανότητα κάποιος να μπορεί να συνεχίσει μία διακεκομμένη μεταφορά αρχείου αλλά και παράλληλο κατέβασμα κομμάτια ενός αρχείου από πολλούς κόμβους. Επίσης βοηθούν τους πελάτες να συνδεθούν μεταξύ τους και να μεταφέρουν αποτελεσματικά την κρυπτογραφημένη κίνηση τους.

Οι υπερκόμβοι είναι υπεύθυνοι και για την καταγραφή και συντήρηση των λιστών των χρηστών του συστήματος (Global Indexing). Η τεχνική αυτή επιτρέπει σε χρήστες να ψάχνουν άλλους χρήστες μέσω των υπερκόμβων. Η εταιρεία πίσω από το Skype εγγυάται ότι κάθε χρήστης μπορεί να γίνει ορατός από μία αναζήτηση ενός άλλου χρήστη μέσα σε μέγιστο χρονικό περιθώριο των 72 ωρών από την πρώτη του εισαγωγή στο δίκτυο.

Μία ενδιαφέρουσα ιδιότητα του Skype είναι ότι είναι αυτοπροσαρμοζόμενο. Αν έχουμε εγκαταστήσει την εφαρμογή μπορεί και ο υπολογιστής μας να γίνει υπερκόμβος, αν παρουσιαστεί

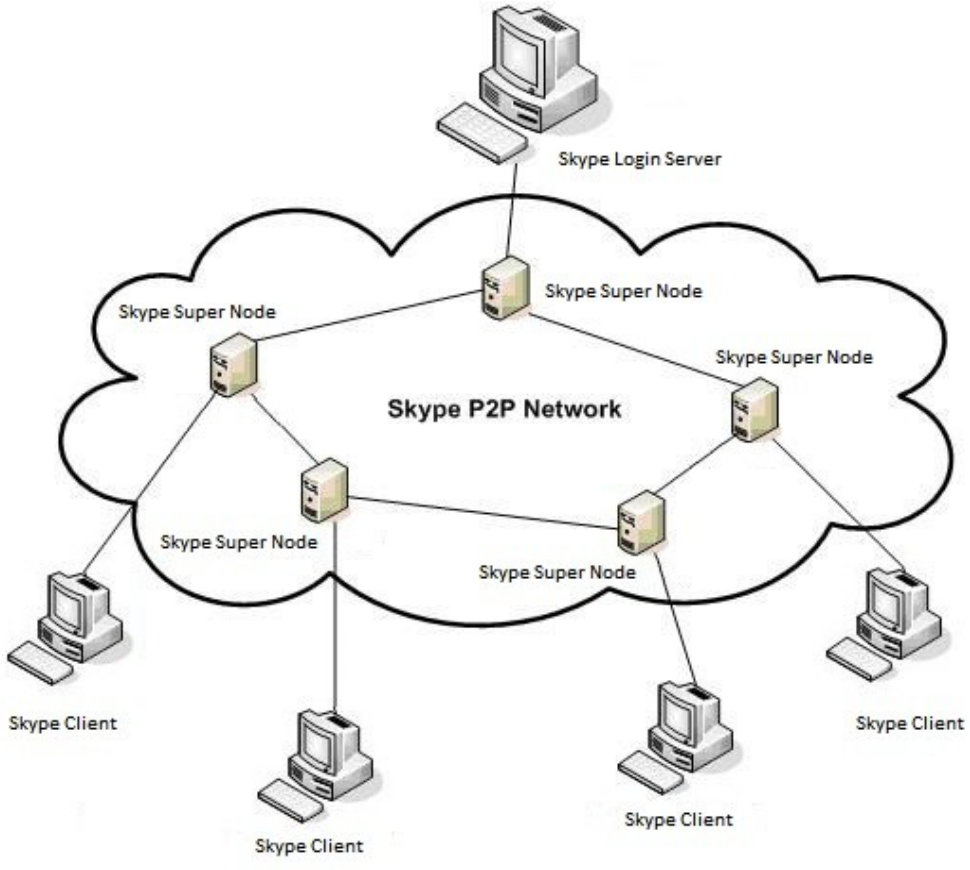
ανάγκη για ένα τέτοιο ενδεχόμενο, χωρίς καν να το πάρουμε είδηση καθώς αυτή η ιδιότητα δεν έχει παρατηρήσιμο αντίκτυπο στην επίδοση του υπολογιστή.

Οι υπερκόμβοι αποθηκεύουν τις διευθύνσεις πολλών χιλιάδων χρηστών , χωρίς όμως να αποθηκεύουν δεδομένα σχετικά με μεταφορές αρχείων ή φωνής. Υπό το σκεπτικό αυτό όσο περισσότεροι χρήστες του Skype είναι online τόσο περισσότερη είναι η αποθηκευτική δυνατότητα και το εύρος του δικτύου.

Το Skype δρομολογεί την κίνηση έξυπνα χρησιμοποιώντας το βέλτιστο μονοπάτι χρησιμοποιώντας είτε TCP ή UDP πρωτόκολλο και σπάει την ροή δεδομένων σε πολλά πακέτα τα οποία μπορούν να πάρουν και διαφορετικές διαδρομές προς τον προορισμό όπου και γίνεται η ανασυγκρότησή της.

Σχετικά με την ασφάλεια ο Skype χρησιμοποιεί το στάνταρ ένα εξελιγμένο πρότυπο κρυπτογράφησης με το όνομα Rijndel το όποιο χρησιμοποιούν και οργανισμοί και η κυβέρνηση των Ηνωμένων Πολιτειών της Αμερικής.

Σε παλαιότερες εκδόσεις του Skype η λίστα των επαφών αποθηκευόταν και αυτή στην Registry των Windows. Ήταν κρυπτογραφημένη ήταν είναι τοπική για κάθε υπολογιστή και δεν αποθηκευόταν κεντρικά στο δίκτυο. Πλέον όμως μετά τις εκδόσεις 1.2.0.XX προσφέρεται και κεντρική αποθήκευση.

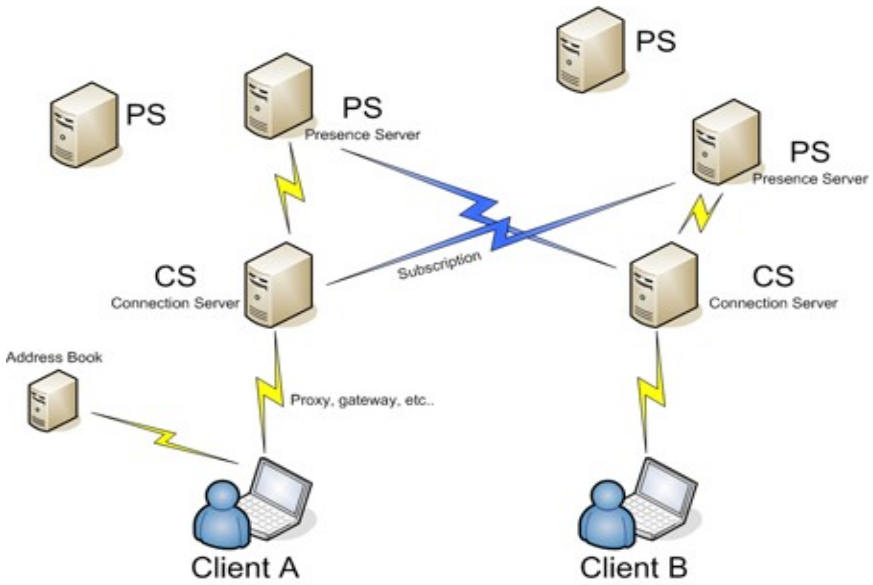


Σχήμα 1: Το δίκτυο του Skype

1.7.2 Ο Windows Live Messenger

Η υπηρεσία αυτή της Μicrosoft [17] είναι και αυτή μία υβριδική Instant Messaging υπηρεσία όχι όμως ως προς τους υπερκόμβους που χρησιμοποιεί το Skype και την ομοτιμία αλλά ώς προς την σύσταση των υπηρεσιών του. Προσφέρει δυνατότητες chat , offline μηνυμάτων, video/audio συνδιάσκεψης και διαμοιρασμού αρχείων. Η διαφορά του Live Messenger είναι ότι η ανταλλαγή μηνυμάτων μεταξύ δύο client είναι αποκλειστικά βασισμένη στο client-server μοντέλο. Συγκεκριμένα η κίνηση μεταφέρεται μέσα από ένα frame relay network (δίκτυο αναμετάδοσης πακέτων που τα δύο επικοινωνούντα μέρη μεταφέρουν τα πακέτα μέσω ενδιάμεσων σταθμών). Για την μεταφορά αρχείων και μετάδοση ήχου/βίντεο χρησιμοποιεί ομότιμη τεχνολογία η οποία όμως για αν γίνει δυνατή χρησιμοποιεί για την εγκαθίδρυση της σύνδεσης έναν ενδιάμεσο εξυπηρετητή. Θα μπορούσαμε να χαρακτηρίσουμε την λειτουργία αυτή σαν υβριδική ομότιμη καθώς μεσολαβεί ενδιάμεσος εξυπηρετητής στο αρχικό στάδιο της έναρξης της επικοινωνίας . Η διαφορά είναι ότι ο αυτός ο εξυπηρετητής δεν λειτουργεί σαν υπερκόμβος όπως στο Skype. Συγκεκριμένα ακολουθεί το αδόμητο κεντρικοποιημένο τύπο ομότιμου δικτύου.

Αν κάποιος χρήστης (χρήστης Α) θέλει να επικοινωνήσει με κάποιον άλλο χρήστη (χρήστη Β) θα πρέπει να συνδεθεί μεσώ ενός εξυπηρετητή σύνδεσης (Connection Server). Πίσω από τον εξυπηρετητή σύνδεσης υπάρχει ένας εξυπηρετητής παρουσίας (Presence Server). Κάθε χρήστης πάντα αντιστοιχίζεται στον ίδιο εξυπηρετητή παρουσίας ο οποίος αποθηκεύει προσωπικά μας στοιχεία όπως το μήνυμα κατάσταση μας , η φωτογραφία προφίλ μας κτλπ.

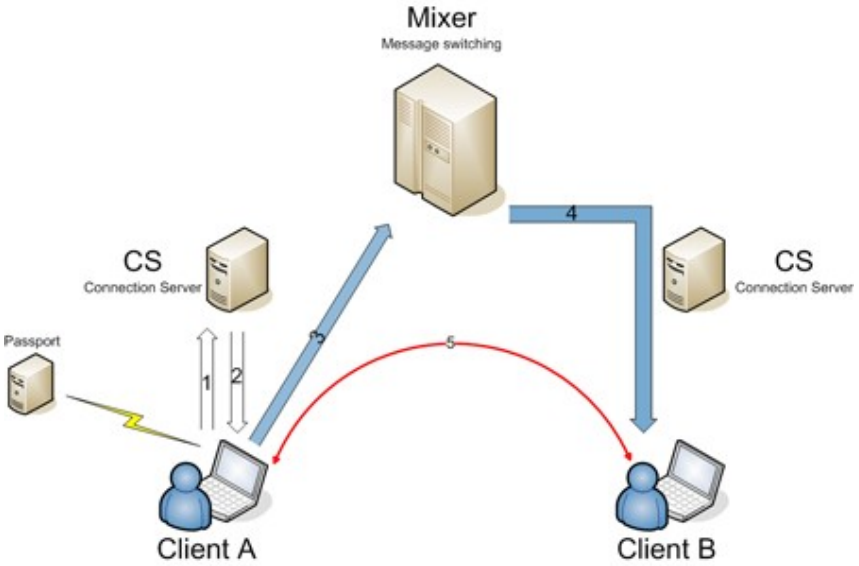


Σχήμα 2: Το frame relay δίκτυο του Windows Live Messenger

Ένα άλλο στοιχείο της αρχιτεκτονικής είναι το βιβλίο διευθύνσεων (Address Book) στο οποίο συνδέεται κατευθείαν και παίρνει την λίστα των επαφών του.

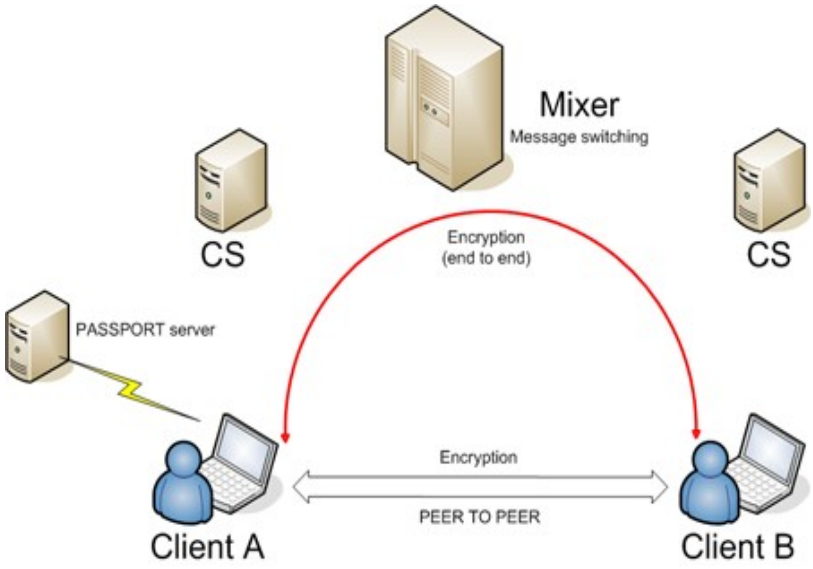
Τότε ο πελάτης του χρήστη Α στέλνει στον εξυπηρετητή σύνδεσης τις επαφές του ο οποίος τις μεταβιβάζει στον εξυπηρετητή παρουσίας και δημιουργεί συνδέσεις στις επαφές του επικοινωνώντας με του αντίστοιχους εξυπηρετητές παρουσίας των επαφών και πηγαίνοντας προς τα κάτω, στους εξυπηρετητές σύνδεσης, με τελικό στάδιο τους πελάτες της κάθε επαφής (όπως και του Β).

Αν τώρα ο χρήστης Α θέλει να μιλήσει με τον Β, λέει στον εξυπηρετητή παρουσίας του ότι θέλει να επικοινωνήσει με κάποιον και ο εξυπηρετητής παρουσίας λέει στον μίκτη (Mixer), ο οποίος διαχειρίζεται την κίνηση των μηνυμάτων προς ένα προορισμό, ότι θέλει να μιλήσει με τον Β. Τότε ο Α και Β μπορούν να συνομιλήσουν.



Σχήμα 3: Ο τρόπος αποστολής μηνυμάτων του Windows Live Messenger.

Όταν οι χρήστες Α και Β θέλουν να στείλουν μεγαλύτερα κομμάτια δεδομένων (Αρχεία, Ήχο/Βίντεο) χρησιμοποιείτε ομότιμη τεχνολογία. Εγκαθιστούν μία συνεδρία μέσω του μίκτη και εγκαθιστούν μία ασφαλή σύνδεση μεταξύ τους για την μεταφορά των δεδομένων.



Σχήμα 4: Τρόπος αίτησης εγκαθίδρισης ομότιμης ασφαλούς συνόδου μεταφοράς δεδομένων του Windows Live Messenger.

Για ήχο και βίντεο υπάρχουν συγκεκριμένοι εξυπηρετητές με παρόμοιες λειτουργίες όπως ο μίκτης. Η κίνηση είναι κρυπτογραφημένη και με την βοήθεια του Passport εξυπηρετητή, ο οποίος ταυτοποιεί τους χρήστες, εγκαθιδρύεται μια ασφαλής σύνδεση.

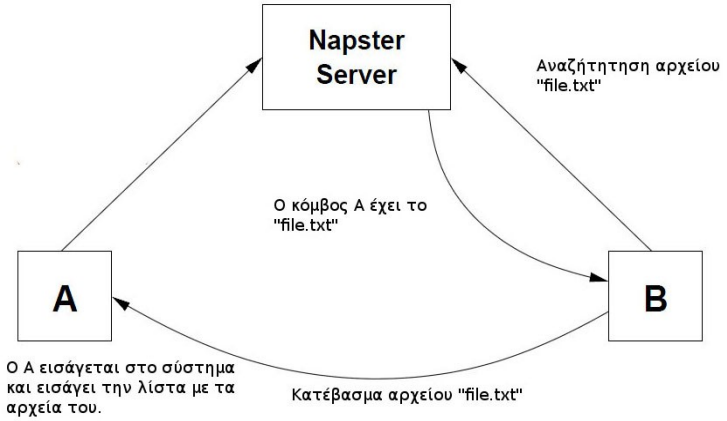
1.8 Η αναζήτηση στα ομότιμα δίκτυα

Στα πρώιμα δίκτυα ομότιμων κόμβων τα οποία διέθεταν πόρους όπως αποθηκευτικός χώρος και εύρος ζώνης στο δίκτυο, υιοθετούσαν διάφορες μεθοδολογίες για να αντιμετωπίσουν ένα βασικό πρόβλημα του παρουσιάστηκε: Το πώς ο κόμβος Χ, ο οποίος ζητούσε μία πληροφορία, θα ανακάλυπτε ποιος κόμβος (πχ ο Z) μπορούσε να του διαθέσει αυτή την πληροφορία. Το πρόβλημα αυτό ονομάζεται πρόβλημα αναζήτησης (lookup) [1,18] . Στη παράγραφο αυτή θα δούμε περιγραφικά τον τρόπο λειτουργίας μερικών από τις τεχνικές που προηγήθηκαν των κατανεμημένων πινάκων κατακερματισμού.

1.8.1 Μοντέλο Κεντρικού Καταλόγου (Centralized Directory Model)

Αυτό το μοντέλο χρησιμοποιήθηκε από το Napster. Το Napster είχε έναν κεντρικό εξυπηρετητή ο οποίος αποθήκευε μια λίστα αρχείων και τους κόμβους στους οποίους βρίσκονταν αυτά τα αρχεία. Ο κάθε κόμβος μπορούσε αναζητήσει κάποιο αρχείο και μαζί με τα αποτελέσματα να πάρει και τους κόμβους που θα πρέπει να επικοινωνήσει για να λάβει το αρχείο. Η αναζήτηση στο μοντέλο αυτό, ορίζεται σαν μια μέθοδος *δομημένης αναζήτησης* (structured lookup), δηλαδή αναζήτηση κατά την οποία ο κάθε κόμβος διατηρεί ορισμένο σύνολο πληροφοριών σχετικά με τους άλλους κόμβους του δικτύου. Η ύπαρξη του κεντρικού ελέγχου, έκανε το σύστημα ευάλωτο σε επιθέσεις

και παρουσίαζε προβλήματα ευρωστίας (robustness) και κλιμάκωσης (scalability). Η βάση δεδομένων του συστήματος μπορεί να έφτανε πάρα πολύ μεγάλα μεγέθη. Επίσης αποτελούσε ένα μοναδικό σημείο αποτυχίας (single point of failure), δηλαδή εάν σταματούσε να λειτουργεί ο εξυπηρετητής, τότε όλο το δίκτυο έβγαινε εκτός λειτουργίας.



Σχήμα 5: Το μοντέλο του ομότιμου δικτύου Napster.

1.8.2 Μοντέλο Πλημμύρας (Flooded Request Model)

Το μοντέλο αρχικά εφαρμόστηκε αρχικά από το δίκτυο Gnutella, δεν χρησιμοποιούσε κάποιο κεντρικό έλεγχο και έστελνε μηνύματα πλημμύρας (flooding), μηνύματα τα οποία στέλνονταν σε όλους τους κόμβους και ζητούσαν κάποια πληροφορία. Το πρωτόκολλο του Gnutella ήταν ένα απλό πρωτόκολλο για αναζήτησεις. Χρησιμοποιούσε πέντε βασικούς τύπους μηνυμάτων, τα οποία είχαν συγκεκριμένο "χρόνο ζωής" (Time To Live - TTL). Κάθε κόμβος που λάμβανε μήνυμα, μείωνε το χρόνο αυτό (αν ήταν μεγαλύτερος του 0), έλεγχε την τοπική λίστα του και σε περίπτωση που δεν είχε ξανασυναντήσει τον κόμβο πού έστειλε το μήνυμα, προωθούσε το μήνυμα σε όσους κόμβους του ήταν γνωστοί. Αν ο κόμβος είχε το ζητούμενο αρχείο σταματούσε την προώθηση του.

1.8.3 Μοντέλο Δρομολόγησης Εγγράφου (Document Routing Model)

Το μοντέλο αυτό χρησιμοποιήθηκε αρχικά από το δίκτυο Freenet. Αντιστοιχούσε ένα κλειδί σε κάθε αρχείο, και η δρομολόγηση στο δίκτυο γινότανε με βάση το κλειδί αυτό. Αρχεία με παρόμοια κλειδιά ομαδοποιούνταν σε μια συστάδα (cluster) κόμβων. Έτσι οι κόμβοι στους οποίους γινότανε αναζήτηση κάποιου αρχείου μειώνονταν. Για να το επιτύχει αυτό, διατηρούσε πίνακες δρομολόγησης (routing tables) σε κάθε κόμβο. Το Freenet ήταν το πρώτο σύστημα που εισήγαγε την έννοια της αντιγραφής (replication). Αρχεία τα οποία αναζητούνταν συχνά, αντιγράφονταν σε κόμβους στους οποίους υπήρχε η μεγαλύτερη πιθανότητα να βρεθούν.

1.8.4 Κατανεμημένοι πίνακες κατακερματισμού (Distributed Hash Tables)

Όλες οι προηγούμενες προσπάθειες και κυρίως αυτή του Freenet, με την εισαγωγή της έννοιας της αντιγραφής αρχείων, έδωσε το έναυσμα για περαιτέρω ανάπτυξη του μοντέλου αυτού ώστε να καταλήξουμε στο μοντέλο που ορίζουμε σήμερα ως *κατανεμημένος πίνακας κατακερματισμού* [11].

Οι κατανεμημένοι πίνακες κατακερματισμού αντιστοιχούν ένα κλειδί σε μία τιμή και για να δημιουργήσουν ένα λειτουργικό σύστημα αντιγραφής αρχείων και αναζήτησης αυτών, οφείλουν να ορίσουν δύο συναρτήσεις την lookup (για αναζήτηση) την και την insert (για εισαγωγή).

Σε μορφή ψευδοκώδικα οι δυο αυτές μέθοδοι περιγράφονται ως εξής:

- Διαδικασία Lookup

```
lookupKey = hashFunction (filename)
file = dht.lookup(filename)
```

- Διαδικασία Insert

```
lookupKey = hashFunction (filename)
dht.insert(lookupKey,file)
```

Η hashFunction είναι μία συνάρτηση κατακερματισμού που μας βοηθάει να δημιουργούμε τα κλειδιά πού αντιστοιχούμε στα δεδομένα που εισάγουμε στους κατανεμημένους πίνακες κατακερματισμού. Ενδεικτικά, δύο συνηθισμένες συναρτήσεις κατακερματισμού είναι και η SHA-1 και η MD5.

Κεφάλαιο 2 – Pastry

Η Pastry είναι το πρότυπο υλοποίησης ενός δικτύου επικάλυψης (overlay network) με την χρησιμοποίηση κατανεμημένου πίνακα κατακερματισμού (Distributed Hash Table – DHT) και . Αποτελεί το δίκτυο επιλογής για την κατασκευή της εφαρμογής που συνοδεύει την πτυχιακή. Ο κατανεμημένος πίνακας κατακερματισμού που χρησιμοποιεί ονομάζεται PAST.

Στο κεφάλαιο αυτό θα κάνουμε μία ανάλυση της Pastry, του αλγόριθμου δρομολόγησης και της συμπεριφοράς και της δόμησης των κόμβων πού το αποτελούν, την περιγραφή κατανεμημένου πίνακα κατακερματισμού της και των λειτουργιών του καθώς και ενός συστήματος δημοσίευσης/συνδρομής (publish/subscribe) με multicasting και anycasting δυνατότητες που ονομάζεται SCRIBE και έχει φτιαχτεί πάνω από την Pastry.

2.1 Η μορφολογία του δικτύου επικάλυψης της Pastry

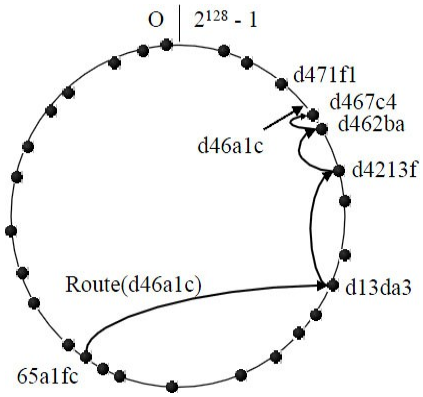
Η Pastry[6] αναθέτει σε κάθε κόμβο του δικτύου επικάλυψης ένα μοναδικό αναγνωριστικό (nodeId) 128 bit (η υλοποίηση της FreePastry [20] διαφέρει λίγο από το πρότυπο και χρησιμοποιεί 160 bit που είναι 20 δεκαεξαδικά ψηφία). Όταν κάποια εφαρμογή η οποία βρίσκεται πάνω από ένα κόμβο της Pastry, θέλει να μεταβιβάσει ένα μήνυμα χρησιμοποιώντας ένα αναγνωριστικό, η Pastry αναλαμβάνει να δρομολογήσει το μήνυμα στο κόμβο του δικτύου ο οποίος βρίσκεται πιο κοντά στο αναγνωριστικό.

Η τοπολογία της Pastry υποθέτει ότι το δίκτυο επικάλυψης της είναι ένας κυκλικό χώρο (δακτύλιος) ο οποίος εκτείνεται από το 0 έως το 2^{128} -1. Το αναγνωριστικό αυτό μπορεί να ανατίθεται τυχαία κατά την σύνδεση ενός κόμβου στο δίκτυο χρησιμοποιώντας μια κρυπτογραφική συνάρτηση κατακερματισμού.

Υποθέτοντας ότι έχουμε *N* κόμβους, το δίκτυο επικάλυψης μπορεί να δρομολογήσει ένα μήνυμα με ένα συγκεκριμένο αναγνωριστικό στον κόμβο ο οποίος έχει το αριθμητικά πλησιέστερο αναγνωριστικό σε (log_bN) βήματα, όπου το *b* είναι μια παράμετρος η οποία έχει τυπική τιμή 4. Η σωστή παράδοση στον κόμβο γίνεται επιτυχώς, εκτός εάν *L/2* γειτονικοί κόμβοι αποτύχουν ταυτόχρονα. Η παράμετρος *L* είναι ένας ακέραιος που έχει τυπική τιμή 16.

2.2 Η δρομολόγηση στην FreePastry

Τα αναγνωριστικά στην Pastry [6] θεωρούνται σαν μια σειρά ψηφίων στη βάση 2^b. Η Pastry για να εξασφαλίσει ότι το μήνυμα θα καταλήξει στο σωστό κόμβο, παρέχει έναν αλγόριθμο δρομολόγησης ο οποίος λειτουργεί με τον εξής τρόπο: Υποθέτουμε ότι έχουμε το μήνυμα με αναγνωριστικό *messageId* το οποίο βρίσκεται στο κόμβο με αναγνωριστικό *nodeIdA*. Έστω ότι το κοινό πρόθεμα του αναγνωριστικού *messageId* και του *nodeidA*, έχει μήκος *k* ψηφία. Ο επόμενος κόμβος στον οποίο θα δρομολογηθεί το μήνυμα θα έχει αναγνωριστικό το οποίο θα μοιράζεται τουλάχιστον *k +1* ψηφία. Αν έχει μόνο *κ* τότε ελέγχει τους πινάκες του και το στέλνει στον κόμβο με το πλησιέστερο ψηφίο στο κοντά στο *κ+1*. Στην περίπτωση που δεν βρει κανένα αναγνωριστικό κόμβου που να έχει κοινό πρόθεμα με το *messageId* το στέλνει σε αυτόν με το συνολικά πλησιέστερο αριθμητικά αναγνωριστικό.



Σχήμα 6: Η δρομολόγηση ενός μηνύματος από τον κόμβο με αναγνωριστικό node65a1fc με κλειδί d46a1c.

2.3 Πώς δομείται ένας κόμβος

Κάθε κόμβος στην Pastry[6] κρατάει εσωτερικά κάποιους πίνακες που τον βοηθά στην δρομολόγηση μηνυμάτων, να καθορίζει την θέση του πάνω στον δακτύλιο αλλά και να γνωρίζει τους γειτονικούς του κόμβους. Οι πίνακες αυτοί είναι ο πίνακας δρομολόγησης(Routing Table) ,ο πίνακας συνόλου γειτονίας (Neighborhood Set) και ο πίνακας συνόλου φύλλων (Leaf Set).

Για λόγους δρομολόγησης τα αναγνωριστικά στην Pastry θεωρούνται σαν μια σειρά ψηφίων με βάση 2^b (πχ b=4 , δεκαεξαδικό σύστημα).Ο πίνακας δρομολόγησης ενός κόμβου περιέχει $(log_{2^b}N)$γραμμές και $2^b - 1$ στήλες. Οι 2^b-1 στήλες σε μία γραμμή n αναφέρονται σε έναν κόμβο οποίου το αναγνωριστικό μοιράζεται η ψηφία με τον κόμβο στον οποίο ανήκει ο πίνακας, αλλά διαφέρουν στο $n+1$ ψηφίο στο οποίο αντιστοιχεί μία από τις $2^b -1$ υπόλοιπες τιμές, εκτός από την τιμή $n+1$ του ψηφίου του κόμβου που διατηρεί τον πίνακα.

Η παράμετρος b έχει ρυθμιστικό ρόλο ανταλλάσσοντας την αύξηση του μεγέθους του πίνακα δρομολόγησης με το μέγιστο αριθμό μεταβιβάσεων (hops) που χρειάζονται για τη δρομολόγηση μεταξύ ενός ζευγαριού κόμβων. Αν για παράδειγμα δει ότι γίνονται πολλά hops για να φτάσει ένα μήνυμα αυξάνετε το μέγεθος του πίνακα και γίνεται η δρομολόγηση πιο αποτελεσματική και έτσι μειώνονται τα hops.

NodeId 10233102

Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Σχήμα 8: Υποθετική κατάσταση κόμβου με Id 10233102, b = 2, και L = 8.

0x	1x	2x	3x	4x	5x		7x	8x	9x	ax	bx	cx	dx	ex	fx
60x	61x	62x	63x	64x		66x	67x	68x	69x	6ax	6bx	6cx	6dx	6ex	6fx
650x	651x	652x	653x	654x	655x	656x	657x	658x	659x		65bx	65cx	65dx	65ex	65fx
65a0x		65a2x	65a3x	65a4x	65a5x	65a6x	65a7x	65a8x	65a9x	65aax	65abx	65acx	65adx	65aex	65afx

Σχήμα 7: Ο πίνακας δρομολόγησης για ένα κόμβο με αναγνωριστικό 65a1χ, b=4. Τα ψηφία είναι στο δεκαεξαδικό σύστημα (base16).

Ο πίνακας του σύνολου γειτονίας περιλαμβάνει τους M κόμβους με αναγνωριστικά τα οποία βρίσκονται πιο κοντά στο κόμβο στον οποίο ανήκει ο πίνακας και είναι "γείτονες" του πάνω στον δακτύλιο, ενώ το πίνακας (είναι διπλός) του σύνολου διατηρεί τους $L/2$ κόμβους που είναι είναι αριθμητικά πλησιέστεροι (άμεσοι γείτονες) αλλά έχουν μεγαλύτερο αριθμητικά αναγνωριστικό από το αναγνωριστικό του κόμβου που έχει τον πίνακα και τους $L/2$ οι οποίοι είναι πλησιέστεροι αλλά με μικρότερο αναγνωριστικό. Τυπικές τιμές για τα M και L, είναι 2^b ή $2x2^b$.

2.4 Πώς αυτο-οργανώνεται η Free Pastry

Άφιξη κόμβου

Κατά την άφιξη ενός κόμβου [4] στο δίκτυο, ο κόμβος πρέπει να ενημερώσει τους πίνακές του και να πληροφορήσει τους άλλους κόμβους για την ύπαρξή του. Σταν κόμβο ανατίθεται ένα αναγνωριστικό nodeId και ήδη γνωρίζει τον κόμβο από τον οποίο έκανε την εισαγωγή του στο δίκτυο. Ο νέος κόμβος ζητάει από τον κόμβο εισαγωγής του (Bootstrap Node) να στείλει ένα μήνυμα, με την πρόθεση του να εισαχθεί στο δίκτυο, με αναγνωριστικό ίδιο με το αναγνωριστικό που του ανατέθηκε (nodeId) στον κόμβο με το πλησιέστερο αριθμητικά αναγνωριστικό σε αυτό. Όταν φτάσει το μήνυμα ο νέος κόμβος αντιγράφει τον πίνακα με το σύνολο φύλλων από τον κόμβο προορισμού του μηνύματος και συμπληρώνει την *i-οστή* γραμμή του πίνακα δρομολόγησης όπου *i* ο αριθμός του κόμβου (βηματικά) που συνάντησε το μήνυμα μέχρι να φτάσει στον προορισμό του. Στην συνέχεια ενημερώνει τους κόμβους που έχει στους πίνακές του για την άφιξή του.

Αποχώρηση Κόμβου

Ένας κόμβος θεωρείται ότι έχει αποχωρήσει [4] από το δίκτυο αν οι άμεσοι γείτονες του στο χώρο αναγνωριστικών της Pastry δεν μπορούν να επικοινωνήσουν μαζί του. Η επικοινωνία αυτή γίνεται περιοδικά μέσω UDP μηνυμάτων και σε περίπτωση που ένας κόμβος διαπιστώσει ότι ένας γειτονικός του κόμβος έχει αποχωρήσει τότε ελέγχει από ποια μεριά του συνόλου φύλλων (μικρότερων και μεγαλύτερων αναγνωριστικών) και παίρνει το σύνολο φύλλων του κόμβου με το μεγαλύτερο δείκτη (index) στον πίνακα και επιδιορθώνει το δικό του. Για τον πίνακα δρομολόγησης, ο κόμβος επικοινωνεί με τους άλλους κόμβους στην ίδια γραμμή με το κόμβο που αποχώρησε, με σκοπό να αντικαταστήσει τη στήλη που δεν αντιστοιχεί ποια σε ενεργό κόμβο. Το σύνολο γειτονίας αναβαθμίζεται ανταλλάσσοντας μηνύματα με άλλα μέλη του συνόλου γειτονίας.

Επανάκαμψη κόμβου

Ένας κόμβος που απέτυχε προσωρινά και επανακάμπτει [4] αρχικά προσπαθεί να επικοινωνήσει με τους κόμβους οι οποίοι βρίσκονταν στον πίνακα του πιο πρόσφατου συνόλου φύλλων του, ανακτώντας το σύνολο φύλλων του καθενός. Αν ανάμεσα σε όλα τα αναγνωριστικά των κόμβων που έλαβε, βρει τον εαυτό του, τότε ο κόμβος ανανεώνει το σύνολο φύλλων του βασίζοντας την ανανέωση στις πληροφορίες που λαμβάνει από τους κόμβους που βρήκε τον εαυτό του στο σύνολο φύλλων τους και στη συνέχεια ειδοποιεί τα μέλη του νέου σύνολου φύλλων του για τη παρουσία του. Στην περίπτωση που δεν βρει τον εαυτό του θα πρέπει ακολουθήσει την διαδικασία πρόσθεσης νέου κόμβου στο δίκτυο.

2.6 PAST - Ο κατανεμημένος πίνακας κατακερματισμού της Pastry

Το σύστημα PAST [2] είναι φτιαγμένο πάνω από Pastry και υλοποιεί τον κατανεμημένο πίνακα

κατακερματισμού της. Σε κάθε στιγμιότυπο του PAST ανατίθεται ένα αναγνωριστικό. Τα αναγνωριστικά αυτά βρίσκονται ομοιόμορφα διασκορπισμένα στον χώρο αναγνωριστικών (παρόμοια με τους κόμβους αλλά 160 bit). Οι βασικές λειτουργίες που ορίζονται στον PAST είναι:

- fileId = Insert(name, owner — credentials, k, file) : Αποτελεί την μέθοδο εισαγωγής στον PAST. Το αρχείο *file* αποθηκεύεται με το όνομα *name,* σε ένα αριθμό *k(Replication Level)* κόμβων στο δίκτυο *της* Pastry πάνω από το οποίο τρέχει η εφαρμογή PAST. Το αναγνωριστικό του αρχείου (*fileId*) που παράγεται από αυτήν την διαδικασία είναι 160 bit*,* το οποίο μπορεί να χρησιμοποιηθεί για τον προσδιορισμό του αρχείο στον PAST κατά την αναζήτηση. Για την παραγωγή του αναγνωριστικού χρησιμοποιείται η συνάρτηση κατακερματισμού *SHA-1*, με παραμέτρους το όνομα του αρχείου, το δημόσιο κλειδί του ιδιοκτήτη και ένα τυχαίο salt για την παραγωγή ενός ψευδοτυχαίου κλειδιού. Αυτό εγγυάται την μεγάλη πιθανότητα στην μοναδικότητα των αναγνωριστικών Στην αντίθετη και σπάνια περίπτωση που παραχθεί ένα δεύτερο αρχείο με ίδιο αναγνωριστικό και εισαχθεί το σύστημα φροντίζει για την απόρριψη του.

- *file* = Lookup(*fileId):* Η μέθοδος αυτή ανακτά ένα αντίγραφο του αρχείου με αναγνωριστικό *fileId* το οποίο δέχεται σαν όρισμα, πάντα με την προϋπόθεση ότι το αρχείο αυτό υπάρχει στο δίκτυο σε ένα τουλάχιστον από τους *k* κόμβους που το φιλοξενούν. Συνήθως το αρχείο θα ανακτηθεί από τον κόμβο ο οποίος βρίσκεται πιο κοντά στο κόμβο ο οποίος κάνει την αναζήτηση.

- *Reclaim(fileId, owner-credentials):* Ζητάει την αποδέσμευση του χώρου των k αντιγράφων που αναγνωρίζονται από το *fileId* από τους κόμβους που τα φιλοξενούν. Όταν η διαδικασία ολοκληρωθεί, ο PAST δεν εγγυάται πάντα ότι μία αναζήτηση θα επιστρέψει το αρχείο. Αντίθετα με μία διαδικασία διαγραφής η *reclaim* μπορεί να αφήσει υπολείμματα του αρχείου κάτω από ορισμένες συνθήκες και θεωρείται μη ασφαλής μέθοδος.

2.6.1 Εξισορρόπηση Φόρτου (Load Balancing)

Η εξισορρόπηση φόρτου (Load Balancing) [2], είναι μια τεχνική η οποία προσπαθεί να είναι δίκαια ως προς την κατανομή του φόρτου ενός ομότιμου συστήματος και να μεγιστοποιήσει την απόδοση του αποφεύγοντας την δημιουργία σημείων συμφόρησης (bottlenecks).

Το συγκεκριμένο πρόβλημα στα δίκτυα ομότιμων κόμβων, συναντάται με δύο τρόπους:

- *Με την κατανομή δεδομένων στους κόμβους*. Κάθε κόμβος, είναι ικανός να αποθηκεύει κάποια δεδομένα ή να υπεύθυνος για κάποια δεδομένα άλλου κόμβου, ανάλογα με το αναγνωριστικό του και αν είναι γειτονικός του. Αυτό το φροντίζει ο ίδιος ο PAST γιατί τα αναγνωριστικά αρχείων που παράγονται, είναι ομοιόμορφα κατανεμημένα στο χώρο αναγνωριστικών.

- *Με την ύπαρξη κάποιας δημοφιλούς πληροφορίας.* Αυτό το πρόβλημα αφορά κόμβους οι οποίοι περιέχουν κάποια πληροφορία για την οποία γίνονται πολλές αναζητήσεις για την απόκτηση της. Παρότι οι πληροφορίες είναι κατανεμημένες ομοιόμορφα στο δίκτυο, για μία

δημοφιλή πληροφορία μεγάλου μεγέθους (πχ κάποιο αρχείο), ο PAST παρέχει μεθόδους επίλυσης του προβλήματος. Έκτος το μηχανισμού αντιγραφής (replication), όπου ένα αρχείο βρίσκεται αποθηκευμένο σε πολλά σημεία για εύκολη πρόσβαση και καλύτερη συντήρηση του, εφαρμόζει ένα μηχανισμό "κρυφής μνήμης" (Cache), έτσι ώστε πληροφορίες στις οποίες υπάρχει συχνή πρόσβαση να αποθηκεύονται προσωρινά για γρήγορη ανάκτηση τους. Επιπλέον για την αντιγραφή παρέχει την μέθοδο "παράκαμψης αντιγραφής" (Replication Diversion) όπου σε υπερβολικές περιπτώσεις αναζήτησης κάποιας πληροφορίας, οι κόμβοι πού την φιλοξενούν, προσωρινά την αντιγράφουν στους γειτονικούς κόμβους τους ώστε να αυξήσουν την διαθεσιμότητα της και να υπάρξει αποσυμφόρηση στους ίδιους.

2.7 Scribe - Ένα σύστημα δημοσίευσης/συνδρομής πάνω από την Pastry.

Το Scribe[3] είναι ένα σύστημα δημοσίευσης/συνδρομής φτιαγμένο πάνω από την Pastry στο οποίο κάποιος κόμβος μπορεί να γραφτεί σε κάποια ομάδα και να δημοσιεύει μηνύματα ή να δημιουργήσει την δική του για τον ίδιο σκοπό στην οποία μπορούν να συμμετέχουν και άλλοι κόμβοι.

Το Scribe προσφέρει ένα σχετικά απλό API και οι λειτουργίες που παρέχει είναι :

- *create(credentials, groupId):* Δημιουργεί μία ομάδα με αναγνωριστικό *groupId*.
- *join(credentials, groupId, messageHandler):* ο κόμβος μπαίνει σε μία ομάδα με αναγνωριστικό groupId. Όλα τα multicast μηνύματα που έρχονται περνούνται στον messageHandler.
- *leave(credentials, groupId)* ο χρήστης αποχωρεί από την ομάδα με αναγνωριστικό *groupId*.
- *multicast(credentials, groupId, message)*: στέλνει ένα μύνημα πολυδιανομής στην ομάδα.

Το Scribe χρησιμοποιεί την Pastry για την δημιουργία των ομάδων, την εισαγωγή σε αυτές αλλά και για την δημιουργία του δέντρου της ομάδας που χρησιμοποιείται για την πολυεκπομπή (multicast) μηνυμάτων.

Η Pastry και το σύστημα Scribe είναι πλήρως αποκεντρωμένα. Όλες οι αποφάσεις βασίζονται σε τοπικές πληροφορίες και όλοι οι κόμβοι αντιμετωπίζονται με ισοτιμία. Κάθε κόμβος μπορεί να λειτουργήσει σαν ρίζα (root) του δέντρου πολυεκπομπής , σαν μέλος της της ομάδας, σαν κόμβος που ανήκει σε αυτό και κάθε συνδυασμός των παραπάνω.

2.7.1 Διαχείριση ομάδων

Κάθε ομάδα έχει ένα μοναδικό αναγνωριστικό (groupId). Ο κόμβος με το πλησιέστερο αναγνωριστικό (nodeId) στο αναγνωριστικό του group (groupId) λειτουργεί ως το σημείο συνάντησης (Rendezvous point) για το συγκεκριμένο group. Το σημείο συνάντησης λειτουργεί και ως ρίζα το δέντρου πολυδιανομής της ομάδας.

Για να δημιουργήσει μία ομάδα [3] ένας κόμβος ζητάει από την Pastry να δημιουργήσει ένα μήνυμα χρησιμοποιώντας το όνομα της ομάδας ως κλειδί. Η Pastry το δρομολογεί στον κόμβο με το πλησιέστερο αναγνωριστικό με το κλειδί. Το Scribe εφόσον ελέγξει ότι μπορεί να δημιουργηθεί η ομάδα την τοποθετεί στην λίστα με τις ομάδες που ήδη γνωρίζει.

Το αναγνωριστικό μιας ομάδας (groupId) είναι αποτέλεσμα της συνάρτησης κατακερματισμού SHA-1 με είσοδο το όνομα που του δίνει ο δημιουργός. Αυτό εγγυάται την ομοιόμορφη κατανομή των αναγνωριστικών των ομάδων όπως συμβαίνει και με τα αναγνωριστικά των κόμβων.

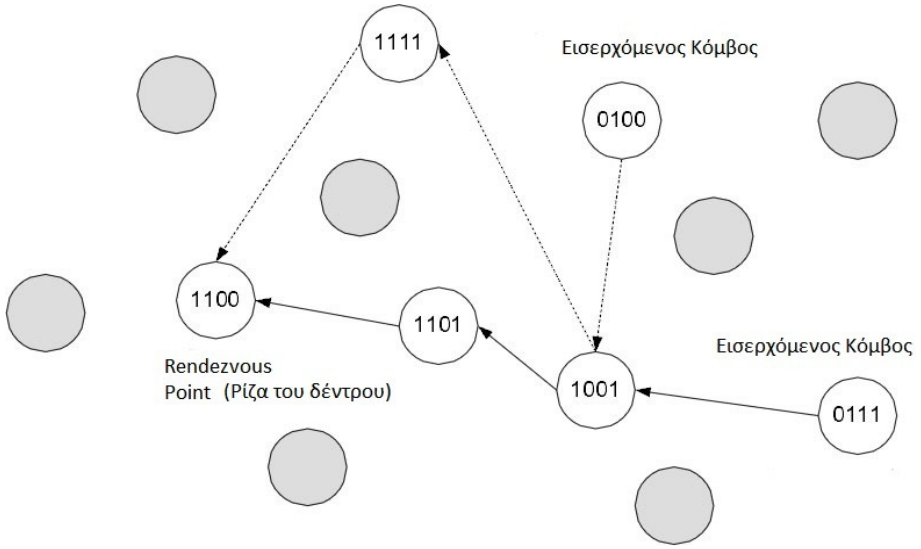
2.7.2 Διαχείριση μελών

Το Scribe δημιουργεί το δέντρο πολυδιανομής [4] με το σημείο συνάντησης να είναι η ρίζα όπου και τα μηνύματα θα μεταδίδονται στα κλαδιά του.

Το δέντρο πολυδιανομής δημιουργείται με μία μέθοδο παρόμοια με αυτή του αντίστροφου μονοπατιού. Σχηματίζεται με τα μονοπάτια που πήραν τα μηνύματα των κόμβων που είναι μέλη του μέχρι το σημείο συνάντησης (Rendezvous point). Κόμβοι οι οποίοι ανήκουν στο δέντρο λέγονται και "προωθητές" (forwarders) μιας ομάδας. Κάθε τέτοιος κόμβος διατηρεί ένα πίνακα από τα παιδιά του (children) για την ομάδα με την οποία σχετίζεται του οποίου οι εγγραφές αποτελούνται από την IP διεύθυνση και το αναγνωριστικό του κόμβου τους.

Όταν ένας κόμβος επιθυμεί να συμμετάσχει σε μία ομάδα, όπως έχουμε ήδη πει, στέλνεται ένα μήνυμα μεσώ της Pastry με το όνομα της ομάδας (πχ route (join, groupId)) αν κατά την πορεία του προς το σημείο συνάντησης συναντήσει ένα προωθητή της συγκεκριμένης ομάδας ο κόμβος αυτός γίνεται παιδί του προωθητή και τοποθετείται στον πίνακα με τα παιδιά του. Σε περίπτωση πού δεν είναι δημιουργεί τον πίνακα και τον τοποθετεί. Μετά στέλνει ο ίδιος ο προωθητής μήνυμα προς το σημείο συνάντησης και ακολουθείται η ίδια διαδικασία μέχρι να φτάσουμε στην ριζά του δέντρου.

Όταν ένας κόμβος επιθυμεί να φύγει από μία ομάδα και έχει παιδιά που ανήκουν σε αυτήν δηλώνει τοπικά ότι έχει φύγει από την ομάδα αλλά συνεχίζει το έργο του ως προωθητής. Σε αντίθετη περίπτωση στέλνει ένα μήνυμα αποχώρησης (leave) στον άμεσο προωθητή του και αποχωρεί από το δέντρο. Σε περίπτωση που κάθε προωθητής μέχρι την ρίζα εξυπηρετούσε το κανάλι επικοινωνίας του κόμβου που έφυγε, δεν ήταν μέλος της ομάδας και δεν είχε άλλα παιδιά παρά μόνο προωθητές (μη μέλη) τότε αποχωρούν όλοι τους με αντίστοιχο τρόπο.



Σχήμα 9: Διαχείρηση των μελών του δέντρου πολυδιανομης. Το διακεκομμένο κομμάτι θα ήταν το μονοπάτι του κόμβου 0100 με την ρίζα σε περίπτωση που ο κόμβος 1001 δεν ήταν προωθητής για την ομάδα που θέλει να εγγραφεί ο 0100.

Κεφάλαιο 3 – Η Βιβλιοθήκη FreePastry

Σε αυτό το κεφάλαιο θα δούμε κάποιες βασικές λειτουργίες , παραδείγματα σε κώδικα και επεξήγηση εννοιών της βιβλιοθήκης FreePastry που υλοποιεί το πρότυπο δικτύου επικάλυψης Pastry που είδαμε στο προηγούμενο κεφάλαιο και που αποτελούν βάση για τον κώδικα που χρησιμοποιήθηκε για την υλοποίηση του FreePastry DHT Messenger , της εφαρμογής που συμπληρώνει την πτυχιακή. Η FreePastry δημιουργήθηκε και συντηρείται από το πανεπιστήμιο Rice στο Χιούστον της πολιτείας του Τέξας των Ηνωμένων πολιτειών της Αμερικής. (Τα παραδείγματα του κεφαλαίου αυτού αποτελούν τροποποιημένες εκδόσεις των αντίστοιχων του tutorial της Freepastry [20] προσθέτοντας τους κάποια ιδιαίτερα χαρακτηριστικά που δεν καλύπτονται από αυτό και που μας χρειάστηκαν για την υλοποίηση της εφαρμογής).

3.1 Continuations

Η FreePastry παρέχει ένα συγκεκριμένο τρόπο για να ψάχνουμε και να τοποθετούμε δεδομένα στον κατανεμημένο της πίνακα κατακερματισμού (PAST), τις continuations.

Για να αντιληφθούμε καλύτερα τι είναι οι continuations μπορούμε να τις χαρακτηρίσουμε σαν

callbacks ή στην περίπτωση της Java σαν Listeners. Τυπικά χρησιμοποιούνται μία φορά για να αναζητήσουμε το αποτέλεσμα της λέξης κλειδί που τους δίνουμε ή για να αποθηκεύσουν στον PAST τα δεδομένα που επιθυμούμε κάτω από αυτό το κλειδί. Ένας επίσης] λόγος που τις χρησιμοποιούμε είναι γιατί είναι non blocking , ασύγχρονες διαδικασίες και η εφαρμογή μας δεν χρειάζεται να περιμένει για ένα αποτέλεσμα για να συνεχίσει την λειτουργία της όπως συμβαίνει συνήθως αν χρησιμοποιούσαμε κάποια RMI (Remote Method Invocation) διαδικασία. Αυτό μας επιτρέπει να έχουμε κάποια άνεση στο να αναζητούμε και να αποθηκεύουμε δεδομένα χωρίς να είναι απαραίτητο να περιμένουμε την επαλήθευση των αποτελεσμάτων των ενεργειών αυτών και παράλληλα η εφαρμογή μας να μπορεί να επικεντρωθεί και σε άλλες εργασίες που έχει να κάνει.

Στον παρακάτω κώδικα βλέπουμε το Interface πού πρέπει μία continuation που δημιουργούμε να υλοποιεί.

```
public interface Continuation {


  public void receiveResult(Object result);


  public void receiveException(Exception ex);
}
```

Το Interface Continuation μας υπαγορεύει ότι σε κάθε continuation που κατασκευάζουμε πρέπει να υλοποιούνται οι παρακάτω μέθοδοι:

- *receiveResult(Object result)* : Καλείται όταν βρει το αποτέλεσμα που ψάχνουμε και δέχεται σαν όρισμα το αντικείμενο που βρέθηκε. Αν δεν βρει αποτέλεσμα στην αναζήτηση μας επιστρέφει null τιμή. Η receiveResult καλείται για την αποθήκευση δεδομένων σε αυτήν όταν αυτό βρεθεί από την lookup μέθοδο του PAST. Χρειάζεται να υλοποιήσουμε δική μας μέθοδο για να πάρουμε το αποτέλεσμα το οποίο αναζητήσαμε από την continuation. Αυτός ο χειρισμός φαίνεται στο επόμενο κομμάτι κώδικα όπου παραθέτουμε μία ολοκληρωμένη.
- *receiveException(Exception ex)* : Καλείται όταν κάτι δεν πάει κάτι καλά με την αναζήτηση ή αποθήκευση που της ζητήσαμε να κάνει και μας επιστρέφει την εξαίρεση (Exception) που παράχθηκε.

Για να γίνει αντιληπτή η λειτουργιά μιας continuation παραθέτουμε μία ολοκληρωμένη κλάση που υλοποιεί το Interface Continuation και τις μεθόδους της:

```
public class myContinuation implements Continuation {

   boolean isReady;
   Object result;

   public void receiveResult(Object result) {
```

```
      if (result != null) {
         System.out.println("Successfully looked up " + result);
         this.result = result;
      }
      isReady = true;

   }

   public void receiveException(Exception ex) {
      System.out.println(ex);
   }

   public Object getResult() {
      return result;
   }

   public boolean isReady() {
      return isReady;
   }


}
```

Όπως παρατηρούμε στο προηγούμενο κομμάτι κώδικα , εκτός από τις μεθόδους receiveResult και receiveException που μας υποχρεώνει το Interface μιας continuation να υλοποιήσουμε, δημιουργήσαμε και δύο δικές μας τις :

- *Object getResult()*: Με αυτήν την μέθοδο μπορούμε να πάρουμε το αποτέλεσμα της αναζήτησης μας. Χρειάζεται βέβαια να κάνουμε casting στο αντικείμενο που αναμένουμε να μας επιστρέψει η continuation γιατί μας το επιστρέφει σαν αντικείμενο Object που επεκτείνουν όλες οι κλάσεις της java.

- *boolean ısReady()* : Είναι μία βοηθητική μέθοδος που την χρησιμοποιούμε για να δούμε αν έχει βρεθεί το αποτέλεσμα της αναζήτησης και βέβαια για προστασία από πρόωρη κλήση της getResult με αποτέλεσμα να πάρουμε null τιμή εφόσον δεν το έχει βρει ακόμα.

Ένας διαφορετικός τρόπος για να χρησιμοποιήσουμε μια continuation στην FreePastry είναι ως ανώνυμη εσωτερική κλάση αν δεν θέλουμε να την χρησιμοποιήσουμε όπως πριν. Όμως κάθε εξωτερικό αντικείμενο στο οποίο αναφέρεται θα πρέπει να δηλωθεί ως final. Παρακάτω δείχνουμε πώς γίνεται να την χρησιμοποιήσουμε με αυτόν τον τρόπο για ένα lookup.

```
   Past past = null; // Δημιουργείται σε άλλο σημείο
   Id id = null; // Δημιουργείται σε άλλο σημείο

   past.lookup(id, new Continuation() {

     public void receiveResult(Object result) {
```

```
    PastContent pc = (PastContent)result;
    System.out.println("Received a "+pc);
  }

  public void receiveException(Exception result) {
    System.out.println("There was an error: "+result);
  }
});
```

Οι μεταβλητές τύπου Past, Id και PastContent θα αναλυθούν σε επόμενη ενότητα αλλά πληροφοριακά σε αυτό το σημείο θα πούμε απλά σε τι αναφέρονται. Η μεταβλητή past, τύπου Past, αναφέρεται στον κατανεμημένο πίνακα κατακερματισμού της Freepastry, τον PAST. Η μεταβλητή pc, τύπου PastContent, αναφέρεται σε αντικείμενο περιεχομένου του PAST, το οποίο λειτουργεί ως υποδοχέας και περιέχει τα δεδομένα και το κλειδί στα οποία αντιστοιχούν. Η id, τύπου Id, είναι το κλειδί το οποίο παράγουμε και μαζί με την continuation τα περνάμε σαν ορίσματα στην μέθοδο lookup του past.

3.2 Το Περιβάλλον (Environment)

Το περιβάλλον μας βοηθάει να δημιουργούμε πολλαπλούς κόμβους στην ίδια εικονική μηχανή (Virtual Machine) της Java αναθέτοντας στον καθένα το δικό του περιβάλλον εργασίας έτσι ώστε ο καθένας να τρέχει το δικό του στιγμιότυπο (instance) της FreePastry.

Το περιβάλλον μας εξυπηρετεί με τους παρακάτω 6 τρόπους:

- ***Logging (Καταγραφή) -*** Η FreePastry παρέχει ένα δικό της σύστημα καταγραφής γεγονότων το οποίο είναι πιο απλό από αυτό της Java αλλά είναι συμβατά μεταξύ τους. Όποτε αν κάποιος θέλει να το χρησιμοποιήσει μπορεί να το κάνει άφοβα.

- **Parameters (Παράμετροι)** - Οι νέες εκδόσεις της Freepastry επιτρέπουν την αλλαγή των παραμέτρων της, που βρίσκονται στο αρχείο *freepastry.params,* κατά το τρέξιμο της εφαρμογής και δεν απαιτεί recompile. Αυτό γίνεται με εντολές μεσώ του περιβάλλοντος.

- **SelectorManager** - Ο SelectorManager είναι ένα νήμα το οποίο διαχειρίζεται όλη την κίνηση από δίκτυο (network I/O) και τα events του Timer της FreePastry. Αυτό το μοντέλο συνήθως δίνει καλύτερη απόδοση και συγχρονισμό μεταξύ των κόμβων από το να χρησιμοποιούσαμε πολλά άλλα νήματα για την συγκεκριμένη εργασία.

- **Processor/BlockingI/O** - Είναι σημαντικό όλες οι εργασίες που γίνονται στον SelectorManager να μην είναι χρονοβόρες όπως, για παράδειγμα, το νήμα που διαχειρίζεται την κίνηση που έρχεται από το δίκτυο (network I/O) και για τον λόγο να μην υποθέσουν οι

άλλοι κόμβοι ότι ο δικός μας είναι προβληματικός ή εκτός λειτουργίας. Κάποιες φορές όμως το να κάνουμε χρονοβόρες εργασίες είναι ανέφικτο όποτε για αυτήν την κατηγορία μπορούμε να χρησιμοποιήσουμε τον Processor ο οποίος είναι ένα διαφορετικό νήμα στο οποίο αναθέτουμε την διεκπεραίωση των εργασιών αυτών. Μια τυπική κλήση του είναι ή environment.getProcessor().processBlockingIO().

- **TimeSource** - Η FreePastry και οι εφαρμογές της (SCRIBE, PAST) καλούν την TimeSource.currentTimeMillis() και όχι την System.currentTimeMillis(). Το μεγάλο πλεονέκτημα αυτού είναι ότι προσφέρεται ένα virtual ρολόι για τον λόγο ότι όλοι οι υπολογιστές δεν μπορεί να είναι απόλυτα συγχρονισμένοι χρονικά μεταξύ τους και αν χρησιμοποιούμε τον GCPAST (GC = Garbage Collecting), στον οποίο τα περιεχόμενα έχουν προκαθορισμένη χρονική διάρκεια ζωής πριν διαγραφούν, δεν θέλουμε κάποιος κόμβος που έχει λανθασμένη ή μη καλά συγχρονισμένη ώρα με τον δικό μας να τα εκθέτει ενώ εμείς επιθυμούμε την διαγραφή τους.

- **RandomSource** – Έχει το ίδιο Interface με την java.util.Random αλλά μας επιτρέπει την ευκολότερη παραγωγή λαθών αν δώσουμε το κατάλληλο seed.

3.3 Ο ελάχιστος κώδικας για να δημιουργήσουμε και να συνδεθούμε σε έναν δακτύλιο

Όπως έχουμε αναφέρει ήδη στο κεφάλαιο 2 η Pastry θεωρεί το δίκτυο της ως ένα ιδεατό δακτύλιο όπου ο κάθε κόμβος διακρίνεται από τον άλλο από ένα μοναδικό αναγνωριστικό (Node Id) διασκορπισμένο ομοιόμορφα, μαζί με τα υπόλοιπα των άλλων κόμβων, πάνω στην περιφέρεια του. Το πως όμως αυτό υλοποιείται από την FreePatsry θα το δούμε στην παράγραφο αυτή. Όμως πριν εμβαθύνουμε στο παράδειγμα κώδικα επεξηγούμε κάποιες βασικές έννοιες που θα μας χρειαστούν.

Οι έννοιες που θα μας χρειαστούν είναι οι παρακάτω:

- **NodeId -** Το αναγνωριστικό του κάθε κόμβου στην FreePastry το οποίο ανατίθεται τυχαία σε κάθε κόμβο. Ένα τυπικό αναγνωριστικό κόμβου είναι μεγέθους 160 bit το οποίο αναπαρίσταται από 20 δεκαεξαδικά ψηφία (Hex digits).

- **NodeIdFactory –** Παράγει το αναγνωριστικό του κόμβου. Το χρειαζόμαστε γιατί είναι σημαντικό κάποιος κόμβος να μην μπορεί να επιλέξει ποιο θα είναι το αναγνωριστικό του καθώς κάνει το σύστημα πιο ανθεκτικό σε επιθέσεις. Αν διαφορετικά θέλουμε όμως μπορούμε να δημιουργήσουμε το Id χρησιμοποιώντας κάποιο ψηφιακό πιστοποιητικό. Προς το παρόν όμως δεν θα μας χρειαστεί και θα το δούμε πώς γίνεται αυτό όταν θα

αναφερθούμε στην ασφάλεια και κρυπτογράφηση της επικοινωνίας με την χρήση του SSL στρώματος.

- **PastryNode -** Αυτός είναι ένα κόμβος στο δίκτυο. Μία εφαρμογή μας στέλνει μηνύματα σε άλλες εφαρμογές μέσω του κόμβου και ο κόμβος μεταφέρει τα μηνύματα που λαμβάνει στην εφαρμογή.

- **NodeHandle** - Είναι η αναφορά σε ένα αντικείμενο PastryNode. Είναι ο τρόπος που αναφερόμαστε σε έναν συγκεκριμένο κόμβο στο δίκτυο. Ένα αντικείμενο NodeHandle περιέχει το NodeId και οποιαδήποτε πληροφορία χρειάζεται το στρώμα μεταφοράς (transport layer) για να βρει τον κόμβο αυτό. Αυτές οι πληροφορίες είναι η διεύθυνση IP και η θύρα του.

- **Bootstrap Node** - Ο κόμβος που χρησιμοποιούμε για να εισάγουμε τον δικό μας κόμβο στο δίκτυο . Όταν ένας κόμβος ενεργοποιείται μπορεί ή να γίνει μέλος ενός υπάρχοντος δακτυλίου μέσω του Bootstrap Node ή να ξεκινήσει τον δικό του.

- **PastryNodeFactory** - Κατασκευάζει και αρχικοποιεί της παραμέτρους του Pastry Node. Αυτό περιλαμβάνει το στρώμα μεταφοράς, το πρωτόκολλο συντήρησης του Leafset και του πίνακα IP (IP Table). Αυτά τα πρωτοκολλά είναι απαραίτητα στην FreePastry για την συντήρηση της δομής του δικτύου επικάλυψης της.

- **Daemon thread –** Δεν επιτρέπει στην JVM (Java Virtual Machine) να κλείσει όταν η main μέθοδος ολοκληρωθεί.

Ό κώδικας για να δημιουργήσουμε και να συνδεθούμε σε έναν δακτύλιο με επεξηγήσεις του κάθε βήματος είναι ο παρακάτω :

```
   public class OurFirstRing {
 public OurFirstRing (int bindport, InetSocketAddress bootaddress, Environment env) throws
Exception {

   // Τυχαία ανάθεση αναγνωριστικού
   NodeIdFactory nidFactory = new RandomNodeIdFactory(env);

   // κατασκευή του PastryNodeFactory – αρχικοποίηση πρωτοκόλλων
   PastryNodeFactory factory = new SocketPastryNodeFactory(nidFactory, bindport, env);

   // Κατασκευή του κόμβου
   PastryNode node = factory.newNode();

   /*Δηλώνουμε την διεύθυνση του Bootstrap Node από τον οποίο θα εισαχθεί ο δικός μας στο
σύστημα και εκκινούμε την διαδικασία εισαγωγής*/
   node.boot(bootaddress);
```

```
   // Ο κόμβος ίσως χρειαστεί να ανταλλάξει πολλά μηνύματα μέχρι να εισαχθεί πλήρως
synchronized(node) {
     while(!node.isReady() && !node.joinFailed()) {
       // Καθυστερούμε λίγο για να μην περιμένουμε ανταλλάσσοντας  μηνύματα συνεχώς
       node.wait(500);

       // Ακύρωση αν δεν μπορούμε να μπούμε
       if (node.joinFailed()) {
         throw new IOException("Could not join the FreePastry ring. Reason: " +
node.joinFailedReason());
       }
     }
   }

   System.out.println("Finished creating new node "+node);
  }


  public static void main(String[] args) throws Exception {
   // Δημιουργούμε το περιβάλλον για τον κόμβο μας
   Environment env = new Environment();

   // Η παρακάτω γραμμή είναι χρήσιμη όταν το LAN μας έχει ενεργοποιημένο το UPnP
   env.getParameters().setString("nat_search_policy","never");

   try {
     // Η θύρα που χρησιμοποιούμε τοπικά στον δικό μας κόμβο
     int bindport = 9000

     // Η  bootaddress είναι η διεύθυνση του Bootstrap Κόμβου
     InetAddress bootaddress= InetAddress.getByName("192.168.1.10");
     // Η  bootport είναι θύρα που επικοινωνεί ο Bootstrap Κόμβος
     int bootport = 9001;
    // Η  InetSocketAddress περιέχει το IP και την θύρα του Bootstrap Κόμβου
     InetSocketAddress bootAddress = new InetSocketAddress(bootaddr,bootport);


     OurFirstRing ofr = new OurFirstRing(bindport, bootAddress, env);
   } catch (Exception ex) {
     System.out.println(ex);
   }
}
}
```

Τώρα το μόνο που χρειαζόμαστε είναι τουλάχιστον δύο υπολογιστές σε ένα LAN όπου ο ένας με την IP "192.168.1.10" θα είναι ο Bootstrap Node και το σημείο εισαγωγής των άλλων στο δίκτυο.

Αυτό βέβαια δεν είναι δεσμευτικό καθώς ένας κόμβος μπορεί να μπει στο δίκτυο από κάποιον που έχει μπει πριν από αυτόν περνώντας την IP του Bootstrap Node σαν όρισμα στην γραμμή εντολών και τροποποιώντας λίγο τον κώδικα ώστε να το δέχεται και να αρχικοποιεί την InetAddress αλλά για να είναι πιο ευανάγνωστο το παράδειγμα χρησιμοποιούμε μόνο έναν με μία τυχαία IP και για να είναι κατανοητό το τι κάνουμε. Ακόμα καλύτερα, στην περίπτωση που έχουμε υπολογιστές με σταθερές IP στο δίκτυο, θα ήταν να περνάγαμε ένα αρχείο με την λίστα αυτών των IP και των αντίστοιχων θυρών τους έτσι ένας νέος κόμβος να είχε περισσότερες πιθανότητες εισαγωγής σε περίπτωση που για κάποιος Bootstrap Node παρουσιαζόταν προβληματικός ή είναι εκτός λειτουργίας τροποποιώντας τον κώδικα έτσι ώστε σε περίπτωση προβλήματος να δοκίμαζε τον επόμενο στην λίστα.

3.4 Δημιουργία απλής εφαρμογής ανταλλαγής μηνυμάτων ανάμεσα σε κόμβους

Στην προηγούμενη παράγραφο είδαμε πώς μπορούμε να στήσουμε έναν δακτύλιο της FreePastry. Τώρα θα επεκτείνουμε λίγο το προηγούμενο παράδειγμα ώστε να μπορούν οι κόμβοι να ανταλλάσσουν μηνύματα μεταξύ τους δημιουργώντας μία απλή FreePastry εφαρμογή (Application) που να μπορεί να το κάνει αυτό και παράλληλα να δούμε πώς γίνεται αυτή η ανταλλαγή μηνυμάτων καθώς και την δομή του της κλάσης Message.

Οι έννοιες που θα μας χρειαστούν εδώ είναι οι εξής :

- **CommonAPI** - Είναι ένα ενιαίο API (Application Programming Interface) για δομημένα δίκτυα επικάλυψης. Το API είναι σχεδιασμένο ώστε να επιτρέπει σε εφαρμογές να γράφονται με ένα τρόπο που να είναι ανεξάρτητος πρωτοκόλλου, επιτρέποντας την εύκολη μετατροπή τους από την Pastry σε Chord, σε CAN, σε Tapestry, κ.ο.κ. Οι Εφαρμογές χρειάζεται να αλληλεπιδρούν με τα Interfaces του πακέτου CommonAPI (*rice.p2p.commonapi*) και δεν χρειάζεται να να ξέρουν το οτιδήποτε από το πακέτο της Pastry (*rice.pastry*) καθώς φροντίζει για αυτό το ίδιο το API.

- **Application –** Τυπικά λέμε κάθε πρόγραμμα που τρέχει πάνω από ένα κόμβο. Ποιό συγκεκριμένα είναι ένα Interface που πρέπει κάθε εφαρμογή να υλοποιεί. Αυτό το Interface επιτρέπει στον κόμβο κάτω από την εφαρμογή να στέλνει μηνύματα, να την ειδοποιεί για μηνύματα που περνάνε από τον κόμβο αλλά και αλλαγές που συμβαίνουν στους γειτονικούς κόμβους. Μπορούμε να έχουμε πολλές εφαρμογές πάνω από ένα κόμβο αλλά και πολλά στιγμιότυπα από μία εφαρμογή και αυτό γιατί πολλές είναι ενδιάμεσες άλλων εφαρμογών υψηλότερου επιπέδου.

- **Endpoint –** Αυτό το Interface αναπαριστά το σημείο από το οποίο μία εφαρμογή μπορεί να στέλνει μηνύματα και είναι το "παράθυρο" μίας εφαρμογής στο δίκτυο. Το Endpoint παρέχεται από την μέθοδο registerApplication() του αντικειμένου Node.

- **Id –** Το Interface αυτό είναι μία αφαίρεση του αναγνωριστικού ενός κόμβου σε ένα δίκτυο

επικάλυψης. Το Interface υλοποιείται και από το NodeId.

- **Message –** Είναι η αφαίρεση ενός μηνύματος στο common API. Είναι ένα Interface και για αυτό τα μηνύματα μεταξύ των κόμβων θα πρέπει να υλοποιήσουν τις μεθόδους του.

Θα αρχίσουμε εξετάζοντας πρώτα την κλάση του μηνύματος μας και πώς υλοποιείται το Interface Message.

```
  public class MyMessage implements Message {

// Από που στάλθηκε το μήνυμα
 Id from;

// Ποιος είναι ο παραλήπτης του
 Id to;

 public MyMessage(Id from, Id to) {
  this.from = from;
  this.to = to;
 }

 public String toString() {
  return "MyMessage"+from+" to "+to;
 }

/*Του θέτουμε χαμηλή προτεραιότητα ώστε να μην επιλέγεται πρώτο σε σχέση με τα μηνύματα
του δικτύου επικάλυψης */

 public int getPriority() {
  return Message.LOW_PRIORITY;
 }
}
```

Η μέθοδος που μας υποχρεώνει το Interface Message να υλοποιήσουμε είναι ή getPriority() στην οποία μπορούμε να θέσουμε την προτεραιότητα του μηνύματος. Εδώ έχουμε επιλέξει χαμηλή ώστε να μην επιλέγεται πρώτο σε σχέση με τα μηνύματα συντήρησης του δικτύου επικάλυψης. Να τονίσουμε επίσης το γεγονός ότι μπορούμε να χρησιμοποιήσουμε μία κλάση που υλοποιεί το Interface Message και σαν υποδοχέα και αντί απλά να εκτυπώνουμε την ειδοποίηση του μηνύματος με την μέθοδο toString() το μήνυμα να εμπεριέχεται σε μία μεταβλητή ή οποία θα εξάγεται με την κατάλληλη μέθοδο που θα παρέχουμε στην κλάση. Το παραπάνω κομμάτι κώδικα αποτελεί την βασική και απλή υλοποίηση ενός μηνύματος.

Ας δούμε τώρα την "εφαρμογή" που θα στέλνει αυτό το μήνυμα.

```
   public class MyApplication implements Application {

  /* Το EndPoint αναπαριστά τον κόμβο κάτω από την εφαρμογή μας .  Κάνοντας κλήσεις σε αυτό
στέλνουμε μηνύματα σε άλλους κόμβους */


 protected Endpoint endpoint;


//Το Node είναι ο κόμβος πάνω στον οποίο υπάρχει η εφαρμογή

 protected Node node;

 public MyApplication(Node node) {
  // Δημιουργούμε ένα στιγμιότυπο του Endpoint
  this.endpoint = node.buildEndpoint(this, "myEndpointInstance");

  this.node = node;

  // Κάνουμε register το endpoint και τώρα μπορούμε να δεχόμαστε μηνύματα
  this.endpoint.register();
 }


 public Node getNode() {
  return node;
 }


 // Στέλνουμε ένα μήνυμα στον κόμβο με Id = id με δρομολόγηση

 public void routeMyMsg(Id id) {
  System.out.println(this+" sending to "+id);
  Message msg = new MyMessage(endpoint.getId(), id);
  endpoint.route(id, msg, null);
 }


//Προσπερνάμε το overlay και στέλνουμε κατευθείαν το μήνυμα στον κόμβο

 public void routeMyMsgDirect(NodeHandle nh) {
  System.out.println(this+" sending direct to "+nh);
  Message msg = new MyMessage(endpoint.getId(), nh.getId());
  endpoint.route(null, msg, nh);
 }


 //Καλείται όταν λάβουμε ένα μήνυμα
```

```
  public void deliver(Id id, Message message) {
    System.out.println("Received message from "+id+" message: "+message);
  }

  // Καλείται όταν λάβουμε εντοπίσουμε ένα νέο γείτονα.

  public void update(NodeHandle handle, boolean joined) {
  }

   /*Καλείται όταν το κάποιο μήνυμα περάσει από τον κόμβο μας. Επιστρέφουμε true ώστε ο
κόμβος μας να το  να το επαναπροωθήσει.*/

  public boolean forward(RouteMessage message) {
    return true;
  }

  public String toString() {
    return "MyApplication "+endpoint.getId();
  }

}
```

Παρατηρούμε ότι για θεωρήσουμε μία κλάση σαν εφαρμογή στην FreePastry θα πρέπει αυτή να υλοποιεί το Interface Application. Οι μέθοδοι του που θα πρέπει να υλοποιηθούν είναι οι:

- *boolean forward(RouteMessage message)* : Η μέθοδός αυτή καλείται όταν περάσει ένα μήνυμα από τον κόμβο μας. Αν έχει άλλο παραλήπτη επιστρέφουμε true ώστε να επαναπροωθήσουμε το μήνυμα. Σε περίπτωση που δεν θέλουμε ο κόμβος μας να συμμετέχει την δρομολόγηση επιστρέφουμε false και το δρομολόγιο του μηνύματος σταματάει εκεί.
- *void update(NodeHandle handle, boolean joined)*: Η μέθοδός αυτή καλείται όταν "ακούσουμε" ένα νέο γείτονα. Δεν μας χρειάζεται εδώ και την αφήνουμε κενή.
- *void deliver(Id id, Message message)*: Είναι η μέθοδος που καλείται όταν φτάσει ένα μήνυμα στον κόμβο και εντός της γίνεται η όποια επεξεργασία του επιθυμούμε.

Εκτός από τις παραπάνω μεθόδους θα διακρίνουμε και άλλες δύο οι οποίες είναι υπεύθυνες για την αποστολή μηνυμάτων. Δεν είναι υποχρεωτικές από το Application Interface και τίθεται σε εμάς η υλοποίηση τους. Ας τις δούμε λίγο αναλυτικότερα:

- *void routeMyMsg(Id id)* : Είναι η τυπική μέθοδος που για να στείλουμε μηνύματα στον κόμβο με το πλησιέστερο κλειδί που δίνουμε σαν όρισμα

```
  public void routeMyMsg(Id id) {
 System.out.println(this+" sending to "+id);
 Message msg = new MyMessage(endpoint.getId(), id);
 endpoint.route(id, msg, null);
}
```

Εσωτερικά της δημιουργούμε το μήνυμα και όπως έχουμε πει με το Endpoint , το οποίο είναι η αναφορά στον κόμβο κάτω από την εφαρμογή και το σημείο επικοινωνίας της με το δίκτυο , στέλνουμε μηνύματα. Να αναφέρουμε ότι το τρίτο όρισμα στην route() είναι ένα NodeHandle του κόμβου που, αν το επιθυμούμε μπορούμε να στείλουμε εκεί το μήνυμα σαν πρώτο βήμα. Εδώ βάλαμε null γιατί δεν επιθυμούμε κάτι τέτοιο. Η δρομολόγηση που γίνεται είναι αυτήν που περιγράψαμε στο προηγούμενο κεφάλαιο της Pastry.

- *void routeMyMsgDirect(NodeHandle nh)*: Η συγκεκριμένη μέθοδος μπορούμε να πούμε ότι είναι μία παραποίηση της προηγούμενης που με τα κατάλληλα ορίσματα παρακάμπτει το δίκτυο επικάλυψης και την κλασσική δρομολόγηση της Pastry και στέλνει μηνύματα κατευθείαν σε ένα κόμβο. Σε πολλές περιπτώσεις αυτό είναι πολύ χρήσιμο καθώς η δρομολόγηση της Pastry στέλνει τα μηνύματα στο πλησιέστερο Id που της δίνουμε περνώντας μέσω άλλων κόμβων μέχρι να φτάσει στον προορισμό του και μπορεί να μην επιθυμούμε να εκθέτουμε το μήνυμα σε άλλους.

```
  public void routeMyMsgDirect(NodeHandle nh) {
 System.out.println(this+" sending direct to "+nh);
 Message msg = new MyMessage(endpoint.getId(), nh.getId());
 endpoint.route(null, msg, nh);
}
```

Παραθέτουμε πιο πάνω την routeMyMsgDirect , που στέλνει μηνύματα κατευθείαν σε ένα κόμβο και αμέσως γίνετε αντιληπτό ότι ή route() μέθοδος δέχεται διαφορετικό συνδυασμό ορισμάτων. Συγκεκριμένα δίνοντας κενή τιμή (null) ως Id και αυτή την φορά περνώντας τον NodeHandle του απομακρυσμένου κόμβου επιτυγχάνουμε την άμεση (direct) αποστολή μηνυμάτων. Σαν υπενθύμιση να πούμε ότι το αντικείμενο NodeHandle είναι ο τρόπος που αναφερόμαστε σε έναν συγκεκριμένο κόμβο στο δίκτυο και παρέχει πληροφορίες όπως είναι η διεύθυνση IP και η θύρα του.

Έχουμε ήδη έτοιμες τις κλάσεις MyMessage και MyApplication που υλοποιούν ένα μήνυμα της FreePastry και μια εφαρμογή που στέλνει αυτά τα μηνύματα αντίστοιχα. Αλλά πως μπορούμε να τοποθετήσουμε την δημιουργία της εφαρμογής πάνω στον κόμβο ;

Το μόνο πού έχουμε να κάνουμε είναι να την τοποθετήσουμε στο εξής σημείο της κλάσης OurFirstRing της προηγούμενης παραγράφου.

```
// Κατασκευή του κόμβου
PastryNode node = factory.newNode();
```

```
// Δημιουργούμε την εφαρμογή
MyApplication app = new MyApplication(node);

   /* Δηλώνουμε την διεύθυνση του Bootstrap Node από τον οποίο θα
εισαχθεί ο δικός μας στο  σύστημα και εκκινούμε την διαδικασία
εισαγωγής*/

   node.boot(bootaddress);
```

3.5 Χρησιμοποιώντας το Scribe

Το Scribe είναι μία εφαρμογή φτιαγμένη πάνω στην FreePastry το οποίο δίνει την δυνατότητα σε κάποιον κόμβο να δημιουργεί κανάλια στα οποία μπορεί να δημοσιεύει περιεχόμενο αλλά και να εγγράφεται ο ίδιος σε κανάλια άλλων κόμβων και να λαμβάνει το περιεχόμενο αυτών. Αυτό το είδος επικοινωνίας ονομάζεται publish/subscribe (Δημοσίευσης/Συνδρομής) και οι δύο βασικοί τρόποι αποστολής μηνυμάτων είναι οι Multicast και Anycast.

Σε αυτήν την παράγραφο οι ορισμοί που μας είναι χρήσιμοι είναι οι εξής:

- **Topic:** Ομάδα κόμβων γύρω από ένα θέμα στην οποία στέλνουμε τα μηνύματα μας. Οι κόμβοι πρέπει να εγγράφονται σε αυτήν για να τα λαμβάνουν

- **IdFactory:** Μια συνάρτηση αναγνωριστικών που παράγει αναγνωριστικά συμβατά με την Pastry.

- **PastryIdFactory:** Μια συνάρτηση αναγνωριστικών που χρησιμοποιεί τον αλγόριθμο SHA-1 για την παραγωγή αναγνωριστικών.

- **ScribeContent:** Ένα μήνυμα του Scribe.

- **Multicast:** Εκπομπή μηνυμάτων σε όλους όσους έχουν γραφτεί στην ομάδα που γίνεται η εκπομπή αυτή.

- **Anycast:** Με αυτήν μέθοδο ένα μήνυμα θα γίνει αποδεκτό από τους κόμβους που είναι εγγεγραμμένοι σε μία ομάδα. Συνήθως την χρησιμοποιούμε όταν θέλουμε να βρούμε ένα συγκεκριμένο κόμβο ο οποίος μπορεί να μας παρέχει μία υπηρεσία. Όλα τα υπόλοιπα μηνύματα θα απορριφθούν μέχρι να βρει έναν διαθέσιμο να μας την προσφέρει.

- **ScribePolicy:** Πολιτική επιλογής της anycast, της μορφολογίας του Scribe δέντρου και της συμπεριφοράς του.

Κάνοντας μία σύντομη αναφορά στο επόμενο κεφάλαιο θα ήταν χρήσιμο για να δούμε σε τι μας χρησίμευσε τελικά η εφαρμογή SCRIBE. Την χρησιμοποιούμε για να ελέγξουμε αν κάποιος χρήστης στην λίστα των επαφών μας είναι online. Αυτό γίνετε με το να μας στέλνει ένα μήνυμα μέσω του Topic του στο οποίο είμαστε εγγεγραμμένοι. Ας δούμε όμως πως αυτό υλοποιείται στην FreePastry μεσώ παραδείγματος και επεξήγησης του.

Αρχίζουμε με την κλάση MyScribeMessage:

```
 public class MyScribeMessage implements ScribeContent {

// Ένας χειριστής του κόμβου του αποστολέα του μηνύματος
NodeHandle from;


public MyScribeContent(NodeHandle from) {
  this.from = from;

}

public String toString() {
  return "Message from "+from;
}
}
```

Το Interface ScribeContent δεν μας υποχρεώνει να υλοποιήσουμε κάποια μέθοδο και προς το παρών είναι κενό γι'αυτό υλοποιήσαμε τις δίκες μας για τον χειρισμό του. Χρειάζεται όμως να το δηλώσουμε για ένα μήνυμα μας ότι το υλοποιεί για να το αναγνωρίζει ο Scribe Client. Όπως στην παράγραφο 3.4 και την κλάση Message το MyScribeMessage (τύπου ScribeContent) μπορεί να χρησιμοποιηθεί σαν υποδοχέας και να επιλέξουμε εμείς ποιο θα είναι το "ωφέλιμο φορτίο" του μηνύματος. Εδώ είναι ο χειριστής του κόμβου μας (NodeHandle) ο οποίος με τον τρόπο που υλοποιεί την δική του μέθοδο toString() το "Message from "+from του βλέπουμε ποιο πάνω το μήνυμα στον παραλήπτη θα είναι της μορφής "Message from <0x####...>//IP:port" όπου:

- **<0x####...>:** το αναγνωριστικό του κόμβου μας
- **IP**: η IP διεύθυνση μας
- **port:** η θύρα που χρησιμοποιεί για να επικοινωνεί με το δίκτυο επικάλυψης

Τώρα δούμε τώρα τις μεθόδους που χρειάζεται να υλοποιήσουμε για έναν Scribe Client. Αυτές είναι οι εξής:

- *public boolean anycast(Topic topic, ScribeContent sc) :* Καλείται όποτε μας έρχεται ένα μήνυμα με την μέθοδο anycast. Αν επιστρέψουμε true το μήνυμα σταματάει εδώ και είμαστε ο μοναδικός παραλήπτης που το αποδέχεται. Διαφορετικά ψάχνει για

κάποιον άλλο παραλήπτη που να είναι πρόθυμος το αποδεχτεί.

- *public void deliver(Topic topic, ScribeContent content)* : Είναι παρόμοια στην λειτουργία της με την deliver του Message μόνο που στα ορίσματα της έχουμε το μήνυμα μου μας ήρθε και το Topic στο οποίο δημοσιεύτηκε.

- *public void childAdded(Topic topic, NodeHandle child):* Καλείται όταν γράφεται κάποιος κόμβος σε ένα Topic το οποίο μας ανήκει. Εκτός από το Topic σαν όρισμα δέχεται και τον χειριστή του κόμβου αυτού. Αν θέλουμε να στείλουμε κάποιο μήνυμα 1 προς 1 με αυτόν ή να κάνουμε οτιδήποτε άλλο από εδώ μπορούμε να πάρουμε τον χειριστή του και να τον αποθηκεύσουμε για αργότερα. Αν δεν μας ενδιαφέρει κάτι τέτοιο την αφήνουμε κενή.

- *public void childRemoved(Topic topic, NodeHandle child)*: όπως η μέθοδος childAdded() μόνο που εδώ παίρνουμε τον χειριστή ενός κόμβου που αποχωρεί από το Topic.

- *public void subscribeFailed(Topic topic):* Καλείται όταν αποτύχει η εγγραφή του κόμβου μας μας σε ένα Topic. Σαν όρισμα δέχεται το Topic στο οποίο αποτύχαμε να γραφτούμε. Εδώ θα μπορούσαμε να υλοποιήσουμε μία ρουτίνα επανεγγραφής αν θέλουμε να έχουμε μία τέτοια συμπεριφορά στο πρόγραμμα μας.

Στο σημείο αυτό θα κατασκευάσουμε τον Scribe Client μας που θα υλοποιεί το Interface ScribeClient και τις μεθόδους που μόλις περιγράψαμε. Θα γραφτούμε σε ένα Topic και θα στείλουμε ένα μήνυμα με τις μεθόδους multicast και anycast.

```
public class MyScribeClient implements ScribeClient{


// Ο χειριστής του Scribe.
  Scribe myScribe;

// Το Topic μας.
  Topic myTopic;

// Ο χειριστής του κόμβου κάτω από την εφαρμογή μας.
  protected Endpoint endpoint;


  public MyScribeClient(Node node) {
    // Δημιουργούμε το Endpoint
    this.endpoint = node.buildEndpoint(this, "myEndPointiInstance");

    // Κατασκευάζουμε τον χειριστής του Scribe.
```

```
    myScribe = new ScribeImpl(node,"myScribeInstance");

    // Κατασκευάζουμε το Topic με όνομα “Our Topic”.
    myTopic = new Topic(new PastryIdFactory(node.getEnvironment()), "Our Topic");
    System.out.println("myTopic = "+myTopic);

    // Κάνουμε Register ώστε να μπορούμε να λαμβάνουμε μηνύματα για την εφαρμογή
    endpoint.register();
  }

//Εγγραφόμαστε στο Topic
  public void subscribe() {
    myScribe.subscribe(myTopic, this);
  }

// Στέλνει ένα μήνυμα multicast
  public void sendMulticast() {
    System.out.println("Node "+endpoint.getLocalNodeHandle()+" broadcasting a message");
    MyScribeContent myMessage = new MyScribeContent(endpoint.getLocalNodeHandle());
    myScribe.publish(myTopic, myMessage);

  }

 // Καλείται όταν λάβουμε ένα μήνυμα από το Topic πού έχουμε γραφτεί .
  public void deliver(Topic topic, ScribeContent content) {
    System.out.println("MyScribeClient deliver("+topic+","+content+")");
  }

   // Στέλνει ένα μήνυμα anycast
  public void sendAnycast() {
    System.out.println("Node "+endpoint.getLocalNodeHandle()+" anycasting a message");
    MyScribeContent myMessage = new MyScribeContent(endpoint.getLocalNodeHandle());
    myScribe.anycast(myTopic, myMessage);
  }

 //Καλείται όταν μας έρθει ένα μήνυμα με την anycast.
  public boolean anycast(Topic topic, ScribeContent content) {
    System.out.println("MyScribeClient anycast Message("+topic+","+content+"):");
    return true;
    }

    public void childAdded(Topic topic, NodeHandle child) {
//    System.out.println("MyScribeClient child added("+topic+","+child+")");
  }

  public void childRemoved(Topic topic, NodeHandle child) {
//    System.out.println("MyScribeClient child removed("+topic+","+child+")");
  }
```

```
  public void subscribeFailed(Topic topic) {
//    System.out.println("MyScribeClient Failed subscribing to topic ("+topic+")");
  }

}
```

Πέρα των μεθόδων που περιγράψαμε δημιουργήσαμε και δύο δικές μας για να στέλνουν τα μηνύματα με τις μεθόδους του Scribe anycast και publish η οποία κάνει multicasting το μήνυμα μας κάτω από ένα Topic. Στην μέθοδο boolean anycast(Topic topic, ScribeContent content) που μας υποχρεώνει το ScribeClient Interface επιστρέφουμε true ώστε να σταματήσει η δημοσίευση του μηνύματος καθώς η anycast βρήκε τον κόμβο μας διαθέσιμο.

Ας δούμε τώρα πώς μπορούμε να ενσωματώσουμε τον MyScribeClient στην κλάση μας OurFirstRing. Ο καινούργιος κώδικας που θα προσθέσουμε είναι ελαφρά τονισμένος.

```
   public class OurFirstRing {
 public OurFirstRing (int bindport, InetSocketAddress bootaddress, Environment env) throws
Exception {

   // Τυχαία ανάθεση αναγνωριστικού
   NodeIdFactory nidFactory = new RandomNodeIdFactory(env);

   // κατασκευή του PastryNodeFactory – αρχικοποίηση πρωτοκόλλων
   PastryNodeFactory factory = new SocketPastryNodeFactory(nidFactory, bindport, env);

   // Κατασκευή του κόμβου
   PastryNode node = factory.newNode();

// Δημιουργία του ScribeClient
   MyScribeClient myScribeClient = new MyScribeClient(node);


   //*Δηλώνουμε την διεύθυνση του Bootstrap Node από τον οποίο θα εισαχθεί ο δικός μας στο
σύστημα και εκκινούμε την διαδικασία εισαγωγής*/
   node.boot(bootaddress);

   // Ο κόμβος ίσως χρειαστεί να ανταλλάξει πολλά μηνύματα μέχρι να εισαχθεί πλήρως
synchronized(node) {
     while(!node.isReady() && !node.joinFailed()) {
       // Καθυστερούμε λίγο για να μην περιμένουμε ανταλλάσσοντας μηνύματα συνεχώς
       node.wait(500);

       // Ακύρωση αν δεν μπορούμε να μπούμε
       if (node.joinFailed()) {
```

```
        throw new IOException("Could not join the FreePastry ring. Reason: " +
node.joinFailedReason());
       }
     }
   }

   System.out.println("Finished creating new node "+node);
}

//Κάνουμε εγγραφή στο Topic

myScribeClient.subscribe();

//Στέλνουμε με την μέθοδο Multicast

myScribeClient.sendMulticast();

//Στέλνουμε με την μέθοδο Anycast

MyScribeClient.sendAnycast();

public static void main(String[] args) throws Exception {
   // Δημιουργούμε το περιβάλλον για τον κόμβο μας
   Environment env = new Environment();

   // Η παρακάτω γραμμή είναι χρήσιμη όταν το LAN μας έχει ενεργοποιημένο το UPnP
   env.getParameters().setString("nat_search_policy","never");

   try {
     // Η θύρα που χρησιμοποιούμε τοπικά στον δικό μας κόμβο
     int bindport = 9000

     // Η  bootaddress είναι η διεύθυνση του Bootstrap Κόμβου
     InetAddress bootaddress= InetAddress.getByName("192.168.1.10");
     // Η  bootport είναι θύρα που επικοινωνεί ο Bootstrap Κόμβος
     int bootport = 9001;
    // Η  InetSocketAddress περιέχει το IP και την θύρα του Bootstrap Κόμβου
     InetSocketAddress bootAddress = new InetSocketAddress(bootaddr,bootport);


     OurFirstRing ofr = new OurFirstRing(bindport, bootAddress, env);
   } catch (Exception ex) {
     System.out.println(ex);
   }
}
}
```

Αυτό ήταν ότι χρειαζόμασταν για την δημιουργία ενός ScribeClient. Να τονίσουμε ότι τα μηνύματα τα στέλνουμε στον εαυτό μας γιατί είμαστε μόνο εμείς γραμμένοι στο Topic μας. Στην εφαρμογή που υλοποιήσαμε δεν γραφόμαστε εμείς στο δικό μας αλλά σε αυτά των επαφών της λίστας των φίλων μας και αυτοί στο δικό μας.

3.6 Χρησιμοποιώντας τον κατανεμημένο πίνακα κατακερματισμού PAST

Ο κατανεμημένος πίνακας κατακερματισμού (DHT) της Pastry είναι ο PAST. Σε αυτή την παράγραφο θα δούμε πως υλοποιείται από την FreePastry. Επίσης ο PAST παρέχεται και σε μία εναλλακτική μορφή με λειτουργία συλλογής σκουπιδιών (Garbage Collecting) με όνομα GCPast. Αυτό του επιτρέπει την διατήρηση αντικειμένων με προκαθορισμένο χρόνο ζωής τον οποίο τον ορίζουμε κατά την εισαγωγή του αντικειμένου GCPastContent στον GCPast.

Τώρα Θα δημιουργήσουμε ένα στιγμιότυπο του για τον κόμβο μας με συνεχή αποθήκευση (Persistent Storage) και θα υλοποιήσουμε μεθόδους εισαγωγής (insert) αλλά και αναζήτησης (lookup) δεδομένων σε αυτόν. Για να δείξουμε τους δύο τρόπους που μπορούμε να χρησιμοποιήσουμε τις Continuations για εισαγωγή (insert) και μια αναζήτηση (lookup) στον PAST. Θα πραγματοποιήσουμε μία εισαγωγή με μία εσωτερική Continuation ως εσωτερική ανώνυμη κλάση (inner anonymous class) και μία αναζήτηση χρησιμοποιώντας την Continuation της παραγράφου 3.1.

Οι βασικές έννοιες που θα μας χρειαστούν είναι:

- **PastContent:** Ένα αντικείμενο υποδοχέας που περιέχει δεδομένα και το κλειδί στο οποία συσχετίζονται.
- **IdFactory:** Μια συνάρτηση αναγνωριστικών που παράγει αναγνωριστικά συμβατά με την Pastry.
- **PastryIdFactory:** Μια συνάρτηση αναγνωριστικών που χρησιμοποιεί τον αλγόριθμο SHA-1 για την παραγωγή αναγνωριστικών.
- **Storage:** Προσφέρει τοπική αποθήκευση. Αυτή μπορεί να είναι συνεχής στον δίσκο ή προσωρινή στην μνήμη για όσο είναι ενεργός ένας κόμβος.
- **MemoryStorage**: Η υλοποίηση της Storage η οποία προσφέρει προσωρινή αποθήκευση στην μνήμη για όσο είναι ενεργός ένας κόμβος.
- **PersistentStorage:** Η υλοποίηση της Storage η οποία προσφέρει συνεχή αποθήκευση στον δίσκο.
- **Cache:** Κρυφή μνήμη για γρήγορη προσπέλαση δεδομένων.
- **LRUCache:** Κρυφή μνήμη για γρήγορη προσπέλαση δεδομένων η οποία διαγράφει δεδομένα τα οποία δεν χρησιμοποιούνται συχνά (Least Recenty Used).

- **StorageManager:** Συνδέει την Cache με την Storage και αυτόματα τοποθετεί τα δεδομένα που εισάγονται ή ζητούνται από την Storage στην Cache για να επιταχύνει μελλοντικές αναζητήσεις.

Αρχικά θα δημιουργήσουμε ένα αντικείμενο PastContent. Τα δεδομένα πού αποθηκεύουμε είναι τύπου Object πού επεκτείνουν όλες οι κλάσεις της Java οπότε μετά την αναζήτηση θα χρειαστεί να κάνουμε casting σε αυτό πού περιμένουμε να μας επιστρέψει.

```
  public class myPastContent implements PastContent{

   public myPastContent(Id id, Object content,  long version){
   this.id = id;
   this.content = content;
   this.version = version;
}
   public PastContent checkInsert(Id id, PastContent existing){


      if(existing!=null && (this.getVersion()<((myPastContent)existing).getVersion())){

            return existing;
      }
      return this;

}

   public Id getId() {
      return this.id;
   }

   public boolean isMutable() {
      return true;
   }
   public Object getContent(){
   return content;
   }

   @Override
  public String toString() {
   return "ID"+id+"MyPastContent ["+content+"]";
 }

   public long getVersion() {
      return version;
   }
```

```
    public PastContentHandle getHandle(Past past) {
       return new myPastContentHandle(id,past.getLocalNodeHandle());
    }

private class myPastContentHandle implements PastContentHandle{

    public myPastContentHandle(Id id,NodeHandle nodeHandle){
    this.id = id;
    this.nodeHandle = nodeHandle;
    }

    public Id getId() {
       return this.id;
    }

    public NodeHandle getNodeHandle() {
       return this.nodeHandle;
    }

    private Id id;
    private NodeHandle nodeHandle;

}

Id id;
Object content;
long version;
```

Οι μεταβλητές που περιέχει η κλάση PastContent είναι οι:

- **id (Id)**: Το κλειδί με το οποίο αποθηκεύουμε το αντικείμενο αυτό στον PAST.
- **Content (Object)**: Τα δεδομένα που περιέχει και για στα οποία αντιστοιχείται το κλειδί για την αναζήτηση.
- **Version (long)**: Η έκδοση των δεδομένων. Θα δούμε αργότερα σε τι χρησιμεύει όταν τα περιγράψουμε την checkInsert μέθοδο.

Πριν περιγράψουμε τις μεθόδους που πρέπει να υλοποιήσουμε για το Interface PastContent θα πρέπει να αναφέρουμε ότι έχουμε δημιουργήσει μία εσωτερική κλάση την myPastContentHandle η οποία περιέχει τον χειριστή του κόμβου μας και την χρειάζεται η μέθοδος PastContentHandle getHandle(Past past).Αυτό είναι χρήσιμο στο να μπορεί ένας κόμβος, ο οποίος διατηρεί ένα αντίγραφο από τα δεδομένα τα οποία έχουμε τοποθετήσει στον PAST, να επικοινωνεί πίσω με τον δικό μας. Την έχουμε με έντονο background για να είναι εύκολα εντοπίσιμη στον κώδικα.

Ας δούμε τώρα και τις μεθόδους τις οποίες πρέπει να υλοποιηθούν για το Interface PastContent:

- **PastContent checkInsert(Id id, PastContent pc)** : Αυτή η μέθοδος χρησιμοποιείται κατά την αντιγραφή ενός PastContent. Επειδή ο PAST στην FreePastry δεν υποστηρίζει την διαγραφή (delete) περιεχομένων από τον χρήστη γιατί θεωρείται μη ασφαλής πρακτική καθώς κάποιος προβληματικός κόμβος που δεν πάρει το μήνυμα μπορεί να συνεχίσει να εκθέτει τα δεδομένα μας. Θέτοντας την isMutable() να επιστρέφει true η ckeckInsert μπορεί να επανεγγράψει τα παλιά περιεχόμενα. Εδώ μπορεί να παραχθεί μία τεχνική η οποία διαγράφει δεδομένα απλά βάζοντας τιμή null στην μεταβλητή content. Ένας άλλος τρόπος θα για να διαγράψουμε δεδομένα θα ήταν αν χρησιμοποιήσαμε τον GCPast και επανεγγράφαμε με ληγμένη χρονική διάρκεια τα GCPastContent αντικείμενα κάτω από ένα κλειδί οπότε αυτά θα διαγράφονταν καθώς δεν θα ίσχυαν πια. Τα ληγμένα αντικείμενα αυτά ονομάζονται και “Tombstones”.

  Όπως μπορούμε να παρατηρήσουμε εμείς ελέγχουμε αν τα περιεχόμενα υπάρχουν και αν η έκδοση τους είναι μεγαλύτερη από τα προϋπάρχοντα για να τα αποθηκεύσουμε. Διαφορετικά η checkInsert δεν επικαλύπτει τα δεδομένα καθώς θεωρούνται παλιά και διατηρεί αυτά πού έχει ήδη.

- **PastContentHandle getHandle(Past past)** : Επιστρέφει τον χειριστή κόμβου (NodeHandle)στον οποίο ανήκουν τα δεδομένα.

- **Id getId()** : Επιστρέφει το κλειδί στο οποίο αντιστοιχίζονται τα δεδομένα που αποθηκεύουμε.

- **boolean isMutable()** : Ανάλογα με το αν επιτρέψουμε true ή false τα δεδομένα μας είναι επανεγγράψιμα ή όχι.

Η εξωτερική Continuation είναι ίδια με της παραγράφου 3.1 οπότε δεν θα την παραθέσουμε εδώ και θα πάμε κατευθείαν στο πρόγραμμα μας.

```
   public class OurFirstRing {
 public OurFirstRing (int bindport, InetSocketAddress bootaddress, Environment env) throws
Exception {

   // Τυχαία ανάθεση αναγνωριστικού
   NodeIdFactory nidFactory = new RandomNodeIdFactory(env);

   // κατασκευή του PastryNodeFactory – αρχικοποίηση πρωτοκόλλων
   PastryNodeFactory factory = new SocketPastryNodeFactory(nidFactory, bindport, env);

   // Κατασκευή του κόμβου
   PastryNode node = factory.newNode();

//Κατασκευή του Storage με  5 MB χωρητικότητα
```

```
Storage storage = new PersistentStorage(idf, "./storage", 5 * 1024 * 1024, node.getEnvironment());

Κατασκευάζουμε μία LRU Cache me 0.5 MB χωρητικότητα
LRUCache lruCachre =  new LRUCache(new MemoryStorage(idf), 2 * 1024 * 1024

/*  ReplicationLevel+1 ο μέγιστος αριθμός αντιγράφων που θα διατηρούμε δίκτυο*/
int ReplicationLevel = 5

// Δημιουργούμε τον PAST
Past past = new PastImpl(node, new StorageManagerImpl(idf, storage,lruCache,
node.getEnvironment())), ReplicationLevel, "myPastInstance");

   /*Δηλώνουμε την διεύθυνση του Bootstrap Node από τον οποίο θα εισαχθεί ο δικός μας στο
σύστημα και εκκινούμε την διαδικασία εισαγωγής*/
   node.boot(bootaddress);

   // Ο κόμβος ίσως χρειαστεί να ανταλλάξει πολλά μηνύματα μέχρι να εισαχθεί πλήρως
synchronized(node) {
     while(!node.isReady() && !node.joinFailed()) {
       // Καθυστερούμε λίγο για να μην περιμένουμε ανταλλάσσοντας  μηνύματα συνεχώς
       node.wait(500);

       // Ακύρωση αν δεν μπορούμε να μπούμε
       if (node.joinFailed()) {
         throw new IOException("Could not join the FreePastry ring. Reason: " +
node.joinFailedReason());
       }
     }
   }

   System.out.println("Finished creating new node "+node);


// Η συμβολοσειρά που θα αποθηκεύσουμε
 final String str = "test";

// Δημιουργούμε το αντικείμενο περιεχομένου του past με ορίσματα (κλειδί , περιεχόμενο)
 final PastContent myContent = new MyPastContent(idf.buildId(str), str,
env.getTimeSource().currentTimeMillis();

/*Εισαγωγή των δεδομένων στον PAST με την μέθοδο insert.
Η μέθοδος μας επιστρέφει των αριθμό των επιτυχών αποθηκεύσεων */
 past.insert(myContent, new Continuation<Boolean[], Exception>() {
 // Παίρνουμε ένα πίνακα από επιτυχείς αποθηκεύσεις
 public void receiveResult(Boolean[] results) {
 int numSuccessfulStores = 0;
 for (int i = 0; int < results.length; int++) {
 if (results[int].booleanValue())
```

```
 numSuccessfulStores++;
          }
          System.out.println(myContent + " successfully stored at " +  numSuccessfulStores + "
locations.");
        }

        public void receiveException(Exception result) {
          System.out.println("Error storing "+myContent);
          result.printStackTrace();
        }
      });

    // περιμένουμε 5 δευτερόλεπτα μέχρι να ολοκληρωθεί η αντιγραφή
    env.getTimeSource().sleep(5000);

// Κάνουμε την αναζήτηση με την Continuation της παραγράφου 3.1
Id mylookupKey = idf.buildId("test"); // δημιουργία του κλειδιού αναζήτησης
myContinuation mylookupContinuation = new myContinuation();
past.lookup(mylookupKey,mylookupContinuation);

    // περιμένουμε 5 δευτερόλεπτα μέχρι να ολοκληρωθεί η αναζήτηση.
 env.getTimeSource().sleep(5000);

if ((myPastContent) mylookupContinuation.getResult() != null) {
                String lookupStr = (String) ((myPastContent)
mylookupContinuation.getResult()).getContent();
System.out.println("I got data with id: "+mylookupkey+ " and Content: "+lookupStr );
            }else{
System.out.println("Search unsuccessful for id: "+mylookupkey );
}

  }


  public static void main(String[] args) throws Exception {
    // Δημιουργούμε το περιβάλλον για τον κόμβο μας
    Environment env = new Environment();

    // Η παρακάτω γραμμή είναι χρήσιμη όταν το LAN μας έχει ενεργοποιημένο το UPnP
    env.getParameters().setString("nat_search_policy","never");

    try {
      // Η θύρα που χρησιμοποιούμε τοπικά στον δικό μας κόμβο
      int bindport = 9000

      // Η  bootaddress είναι η διεύθυνση του Bootstrap Κόμβου
      InetAddress bootaddress= InetAddress.getByName("192.168.1.10");
```

```
    // H  bootport είναι θύρα που επικοινωνεί ο Bootstrap Κόμβος
    int bootport = 9001;
   // H  InetSocketAddress περιέχει το IP και την θύρα του Bootstrap Κόμβου
    InetSocketAddress bootAddress = new InetSocketAddress(bootaddr,bootport);


    OurFirstRing ofr = new OurFirstRing(bindport, bootAddress, env);
   } catch (Exception ex) {
    System.out.println(ex);
   }
}
}
```

Παραθέτουμε τις υλοποιήσεις του κάθε στοιχείου που έχουμε αναφέρει ως τώρα και που βλέπουμε στον κώδικα.

Η PastImpl (δημιουργεί τον PAST) χρειάζεται τα εξής ορίσματα:

- Τον κόμβο (node).
- Τον StorageManager.
- Τον αριθμό των αντιγράφων που θα διατηρεί
- Ένα όνομα για το στιγμιότυπο

Η StorageManagerImpl κατασκευάζει τον StorageManager και χρειάζεται τα :

- Ένα IdFactory
- Μία storage – Εδώ χρησιμοποιούμε συνεχή (Persistent)
- Μια cache - χρησιμοποιούμε την LRUCache.

Η PersistentStorage χρειάζεται τα:

- Ένα IdFactory
- Τον φάκελο που θα αποθηκεύει δεδομένα.
- Το μέγιστο μέγεθος (σε bytes) – έχουμε ορίσει 5 MB
- Το environment (περιβάλλον)

Η LRUCache χρησιμοποιεί τα:

- Μια storage – Χρησιμοποιούμε την MemoryStorage (αποθήκευση στην μνήμη)
- Το μέγιστο μέγεθος (σε bytes) – έχουμε ορίσει 0.5 MB
- Το environment (περιβάλλον)

3.7 Application Level Socket Interface

Η FreePastry μας παρέχει ένα δικό της μηχανισμό για την δημιουργία sockets (υποδοχών) για την επικοινωνία μας με τους άλλους κόμβους τα οποία αποτελούν εξομοίωση τους και δεν απαιτούν κλήσεις συστήματος. Αυτός ο μηχανισμός μας παρέχει τα εξής πλεονεκτήματα:

- Δεν χρειάζεται να δηλώνουμε θύρες (ports) για να δημιουργούμε sockets και έτσι αποφεύγουμε διάφορα προβλήματα συνδεσιμότητα όπως δύστροπα NAT, διάφορα τείχη προστασίας και ανωμαλίες κατά την δρομολόγηση.

- Δεν επεμβαίνουμε στην κίνηση του δικτύου επικάλυψης και έτσι μπορούμε να στέλνουμε μεγάλου μεγέθους δεδομένα χωρίς το επηρεάζουμε.

Η διαδικασία είναι σχετικά απλή για να κάνουμε μία εφαρμογή μας να υποστηρίζει το Application Level Socket Interface. Θα παραθέσουμε πρώτα την τροποποιημένη κλάση MyApplication της παραγράφου 3.4 και μετά θα προχωρήσουμε στην απαραίτητη επεξήγηση που χρειάζεται.

```
public class MyApplication implements Application {

/* Το EndPoint αναπαριστά τον κόμβο κάτω από την εφαρμογή μας .  Κάνοντας κλήσεις σε αυτό στέλνουμε μηνύματα σε άλλους κόμβους */

protected Endpoint endpoint;


//Το Node είναι ο κόμβος πάνω στον οποίο υπάρχει η εφαρμογή

protected Node node;

public MyApplication(Node node) {

  // Δημιουργούμε ένα στιγμιότυπο του Endpoint
  this.endpoint = node.buildEndpoint(this, "myEndpointInstance");

  this.node = node;
  //Μέγεθος του μηνύματος σε Byte
  Message_Length= 600;
  // Buffer για εξερχόμενα μηνύματα
 ByteBuffer out = ByteBuffer.wrap(Message_Length);
```

```
  // Buffer για εισερχόμενα μηνύματα
   ByteBuffer  in = ByteBuffer.allocate(Message_Length);
   endpoint.accept(new AppSocketReceiver() {

     //Καλείτε όταν δεχόμαστε ένα νέο Socket
     public void receiveSocket(AppSocket socket) {
       //Κάνουμε Register το Socket
       socket.register(true, false, 30000, this);

       // Ξανακαλούμε την accept ώστε να μπορούμε να δεχόμαστε πάνω από ένα socket
       endpoint.accept(this);
     }

     //Καλείτε όταν ένα socket είναι έτοιμο να διαβάσει ή να γράψει
     public void receiveSelectResult(AppSocket socket, boolean canRead, boolean canWrite) {
       in.clear();
       try {
         // Διαβάζουμε από το socket και αποθηκεύουμε στον ByteBuffer
       socket.read(in);

    //Εκτυπώνουμε το μήνυμα στην οθόνη σε Unicode μορφή
          System.out.println("Recieved Message: "+new String(in.array(),"UTF-8"));

       } catch (IOException ioe) {
         ioe.printStackTrace();
       }
       //Ξανακαλούμε την register αν περιμένουμε περισσότερα από ένα μηνύματα από το socket
       socket.register(true, false, 3000, this);
     }


      /* Σε περίπτωση που υπάρξει κάποιο πρόβλημα εκτυπώνουμε την εξαίρεση που
δημιουργήθηκε*/

     public void receiveException(AppSocket socket, Exception e) {
       e.printStackTrace();
     }
   });

   // Κάνουμε register το endpoint και τώρα μπορούμε να δεχόμαστε μηνύματα
   this.endpoint.register();
 }

 public Node getNode() {
   return node;
 }
```

```
  public void sendMyMsgDirect(NodeHandle nh, String message) {
  System.out.println(this+" opening Socket to "+nh);
  endpoint.connect(nh, new AppSocketReceiver() {

   //Καλείται όταν ένα Socket γίνεται διαθέσιμο

    public void receiveSocket(AppSocket socket) {
     // register for writing
     socket.register(false, true, 30000, this);
    }

      /* Σε περίπτωση που υπάρξει κάποιο πρόβλημα εκτυπώνουμε την εξαίρεση που
δημιουργήθηκε*/
    public void receiveException(AppSocket socket, Exception e) {
     e.printStackTrace();
    }

   //Γράφουμε στο Socket
    public void receiveSelectResult(AppSocket socket, boolean canRead, boolean canWrite) {
     try {
        out.put(message.getBytes(),0,message.getBytes().lenght);
      socket.write(out);
      // see if we are done
      if (!out.hasRemaining()) {
       socket.close();
       out.clear();
      } else {
       // keep writing
       socket.register(false, true, 30000, this);
      }
     } catch (IOException ioe) {
      ioe.printStackTrace();
     }
    }
  }, 30000);
 }

 // Στέλνουμε ένα μήνυμα στον κόμβο με Id = id με δρομολόγηση
 public void routeMyMsg(Id id) {
  System.out.println(this+" sending to "+id);
  Message msg = new MyMessage(endpoint.getId(), id);
  endpoint.route(id, msg, null);
 }


//Προσπερνάμε το overlay και στέλνουμε κατευθείαν το μήνυμα στον κόμβο
```

```
  public void routeMyMsgDirect(NodeHandle nh) {
    System.out.println(this+" sending direct to "+nh);
    Message msg = new MyMessage(endpoint.getId(), nh.getId());
    endpoint.route(null, msg, nh);
  }


 //Καλείται όταν λάβουμε ένα μήνυμα

  public void deliver(Id id, Message message) {
    System.out.println("Received message from "+id+" message: "+message);
  }

  // Καλείται όταν λάβουμε εντοπίσουμε ένα νέο γείτονα.

  public void update(NodeHandle handle, boolean joined) {
  }

   /*Καλείται όταν το κάποιο μήνυμα περάσει από τον κόμβο μας. Επιστρέφουμε true ώστε ο
κόμβος μας να το  να το επαναπροωθήσει.*/

  public boolean forward(RouteMessage message) {
    return true;
  }

  public String toString() {
    return "MyApplication "+endpoint.getId();
  }

}
```

Αυτό που παρατηρούμε αμέσως είναι η endpoint.accept() στην οποία δημιουργούμε εσωτερικά ένα αντικείμενο AppSocketReceiver() το οποίο είναι υπεύθυνο για να μας επιστρέφει sockets όταν κάποιος προσπαθεί να επικοινωνήσει μαζί μας και να μας στείλει κάποιο μήνυμα. Όταν πάρουμε το socket του περνάμε με την σειρά τις παραμέτρους false, true, 30000, this όπου είναι η ικανότητα να γράφουμε σε αυτό, η ικανότητα να διαβάζουμε, ο χρόνος timeout του socket σε milliseconds και το αντικείμενο AppSocketReceiver στο οποίο αντιστοιχεί. Στην περίπτωση που θέλουμε να στείλουμε εμείς κάποιο μήνυμα αλλάζουμε τις ιδιότητες εγγραφής και ανάγνωσης από το socket που ζητάμε από την sendMyMsgDirect που δημιουργήσαμε και η οποία καλεί την endpoint.connect() με όρισμα ένα χειριστή κόμβου (Nodehandle) με τον οποίο θέλουμε να επικοινωνήσουμε και ένα AppSocketReceiver() ακριβώς όπως η endpoint.accept(). Μία κοινή μέθοδος που βλέπουμε και στις δύο περιπτώσεις είναι η receiveSelectResult() η οποία Καλείται όταν ένα socket είναι έτοιμο να διαβάσει ή να γράψει. Στο socket γράφουμε με την μέθοδο write(out) και διαβάζουμε με την read(in) όπου out και in είναι τα ByteBuffers εξόδου και εισόδου αντίστοιχα.

3.8 SSL Layer

Σε περίπτωση που θα θέλαμε να επιλέξουμε ένα σταθερό Id για κάθε κόμβο του δικτύου επικάλυψης, να ελέγχουμε την πιστοποίηση του κάθε κόμβου για την δημιουργία ενός κλειστού δακτυλίου και να κρυπτογραφήσουμε την κίνηση των μηνυμάτων μεταξύ τους θα πρέπει να χρησιμοποιήσουμε ένα στρώμα επικοινωνίας παραπάνω. Το SLL Layer είναι το κατάλληλο για αυτό και στην παράγραφο αυτήν θα δούμε πώς το χρησιμοποιούμε.

3.8.1 Τι είναι το SSL;

Το SSL [22] είναι ένα στρώμα επικοινωνίας που επιτρέπει την πιστοποίηση μεταξύ δύο υπολογιστών οι οποίοι θέλουν να επικοινωνήσουν μεταξύ τους και κρυπτογραφεί την μεταξύ τους επικοινωνία. Χρησιμοποιεί ένα συνδυασμό δημοσίου και ιδιωτικού κλειδιού. Τα δύο μέρη που θέλουν να επικοινωνήσουν στέλνουν μεταξύ τους τα δημοσιά κλειδιά τους και συμφωνούν στην παραγωγή ενός συμμετρικού κλειδιού για την γρηγορότερη κρυπτογράφηση / αποκρυπτογράφηση των μηνυμάτων που ανταλλάσσουν.

Συνοπτικά τα βήματα που χρειάζεται για να εγκατασταθεί μια SSL σύνοδος (session) είναι :

- Ο κόμβος Α στέλνει ένα “hello” μήνυμα στον κόμβο Β το οποίο περιλαμβάνει της κρυπτογραφικές δυνατότητες του.
- Ο Β στέλνει τις ίδιες πληροφορίες συμπεριλαμβανομένου και του ψηφιακού πιστοποιητικού του.
- Ο Α τον πιστοποιεί και αν έχει ζητηθεί στέλνει και αυτός το δικό τού στον Β όπου ακολουθεί η πιστοποίηση ου από τον Β.
- Ο Α και ο Β συνεργάζονται μεταξύ τους για να επιλέξουν τον επιθυμητό αλγόριθμο κρυπτογράφησης. Ο Α δημιουργεί ένα συμμετρικό κλειδί που το υπογράφει με το δημόσιο κλειδί του Β και του το στέλνει. Ο Β με το ιδιωτικό του κλειδί αποκρυπτογραφεί το μήνυμα αυτό και εξαγάγει το συμμετρικό κλειδί.
- Ανταλλάσσουν μεταξύ τους μηνύματα ότι και οι δύο είναι έτοιμοι για να αρχίσει η επικοινωνία και ότι η “χειραψία” (handshake) , όπως λέγεται η διαδικασία αυτή, έλαβε τέλος.

3.8.2 Ενσωμάτωση του SSL Layer πάνω από το Transport Layer

Εδώ θα κατασκευάσουμε το SSL Layer και θα εγκαταστήσουμε μία μία Bind Strategy η οποία θα φροντίζει ο κόμβος μας να έχει το ίδιο αναγνωριστικό με το όνομα του πιστοποιητικού που δημιουργήσαμε το οποίο βρίσκετε σε ένα εξωτερικό αρχείο. Το παρακάτω κομμάτι κώδικα περιέχει μόνο το σημείου πού δημιουργούμε τον κόμβο καθώς τα υπόλοιπα δεν είναι απαραίτητα και έχουν αναλυθεί σε πολλές από της προηγούμενες παραγράφους. Επίσης να πούμε ότι συμπεριλάβαμε την βιβλιοθήκη bouncycastle (http://www.bouncycastle.org/) που αποτελεί συλλογή κρυπτογραφικών αλγορίθμων

```
// Δημιουργούμε τον προμηθευτή ασφάλειας της BouncyCastle
        Security.addProvider(new BouncyCastleProvider());


        // Δημιουργούμε το Keystore
        final KeyStore store = KeyStore.getInstance("UBER", "BC");
        store.load(new FileInputStream(KeyStoreFile), "".toCharArray());


        /*Δημιουργούμε το αναγνωριστικό από το όνομα του αρχείου που περιέχει το πιστοποιητικό*/
        factory = new SocketPastryNodeFactory(nidFactory, bindport, env) {
        protected TransportLayer<SourceRoute<MultiInetSocketAddress>, ByteBuffer> getSourceRouteTransportLayer(TransportLayer<MultiInetSocketAddress, ByteBuffer> etl, PastryNode pn, MultiAddressSourceRouteFactory esrFactory) {


            // Παίρνουμε το Default Transport Layer
            TransportLayer<SourceRoute<MultiInetSocketAddress>, ByteBuffer> sourceRoutingTransportLayer = super.getSourceRouteTransportLayer(etl, pn, esrFactory);


            try {
                    /*Επιστρέφουμε το Transport Layer με την προσθήκη του SSL στρώματος πάνω από αυτό  αυτό*/
                return new SSLTransportLayerImpl<SourceRoute<MultiInetSocketAddress>, ByteBuffer>(sourceRoutingTransportLayer, store, store, pn.getEnvironment());
            } catch (IOException ioe) {
                throw new RuntimeException(ioe);
            }
        }
```

```
        @Override
        protected BindStrategy<TransportLayerNodeHandle<MultiInetSocketAddress>,
SourceRoute<MultiInetSocketAddress>> getBindStrategy() {
            return new BindStrategy<TransportLayerNodeHandle<MultiInetSocketAddress>,
SourceRoute<MultiInetSocketAddress>>() {


                public boolean accept(TransportLayerNodeHandle<MultiInetSocketAddress> u,
SourceRoute<MultiInetSocketAddress> l, Map<String, Object> options) {
                    // Παίρνουμε το Id από το πιστοποιητικό
                    String idName = (String)
options.get(SSLTransportLayer.OPTION_CERT_SUBJECT);


                    if (idName != null) {
                       //Συγκρίνουμε το id με το όνομα του αρχείου το οποίο το περιέχει
                       if (u.getId().toStringFull().equals(idName)) {
                          // accept
                          return true;
                       } else {
                          // reject
                          System.out.println("Rejecting id:" + u + " which does not match the
certificate entry:" + idName);
                          return false;
                       }
                    }
                    return true;
                 }
              };
           }
        };
   //Δημιουργούμε τον κόμβο με το Id
   node = factory.newNode(id);
```

Η FreePastry περιλαμβάνει υπηρεσία πιστοποίησης (Certificate Authority) και το εργαλείο CAToolImpl για να παράγουμε πιστοποιητικά. Η παρακάτω εντολή μας παράγει ένα πιστοποιητικό.

Το “foo” είναι ο κωδικός του Keystore που περιέχει το KeyMaterial του πιστοποιητικού.

java -classpath FreePastry-2.1.jar;bouncycastle.jar; org.mpisws.p2p.pki.x509.CAToolImpl -p foo -store

Την πρώτη φορά που θα τρέξουμε το CAToolImpl δεν θα έχουμε κάποιο Keystore και θα παραχθεί μία έξοδος παρόμοια με αυτό :

```
[0]  Version: 1
     SerialNumber: 1
     IssuerDN: CN=MyCA
     Start Date: Mon Nov 01 15:16:27 EET 2010
     Final Date: Sat Nov 01 15:16:27 EET 2020
     SubjectDN: CN=MyCA
      Public Key: RSA Public Key
     modulus: 9be7b2f4f1f2d308357dcb8cba3cf65648e99c1fdd72aacae8fddce46b3
25c7ed3eaa96ed4efbba1b65f3b0509c128f1547e4b4baf7e2303e732bf835e2a0ad1ff2bf339e9d
9f7ca68f8f9fe37e3bb5394fa2465b78e26463cdb8d3bd753021f
     public exponent: 10001

     Signature Algorithm: SHA1WithRSAEncryption
       Signature: 38fcc2c1bd1bd0da11d530b0980a9abe55fec05c
            c8e868f361de9106a6d6c7644d9508a0b4e715b9
            a76fe442c0a33e80f0f4ab222f57587fe318ba94
            5c55296a64166aec162c74c191a52d5b9eb3bf27
            9e8a87f3241512bcafa80f7377edfe07

Stored to EBD1E8789423177AFA0D31519B8F56080BDB17D4.store
```

Κεφάλαιο 4 – Η Εφαρμογή FreePastry DHT Messenger

Σε αυτό το κεφάλαιο της πτυχιακή εργασίας θα κάνουμε την παρουσίασης του προγράμματος FreePastry DHT Messenger, ενός πρότυπου εφαρμογής ανταλλαγής άμεσων μηνυμάτων, που δημιουργήθηκε με βάση την ανάλυση της θεωρίας της Pastry και την υλοποίηση της με την βοήθεια της βιβλιοθήκης FreePastry.

Συνοπτικά οι υπηρεσίες που προσφέρουμε με την εφαρμογή αυτή είναι:

- Συνομιλία μεταξύ χρηστών.
- Διατήρηση λίστας επαφών (Διαγραφή, εισαγωγή και μπλοκάρισμα χρήστη).
- Αποστολή και λήψη e-mails.
- Επιλογή κατάστασης (Online, Busy Away και Offline)
- Αλλαγή και δημοσίευση στοιχείων όπως το μήνυμα κατάστασης του χρήστη, το ψευδώνυμο και η εικόνα του προφίλ του.

4.1 Δημιουργία ενός δικτύου

Για να φτιάξουμε ένα μικρό δίκτυο κόμβων που να τρέχουν την εφαρμογή θα πρέπει τουλάχιστον ένας να έχει ενεργοποιηθεί. Από το σημείο και αυτό και μετά οποιοσδήποτε άλλος κόμβος θέλει να εισαχθεί στο δίκτυο δηλώνει ως bootaddress την IP διεύθυνση ενός κόμβου πού είναι ήδη στο δίκτυο. Επίσης μπορεί να επιλέξει και άλλες επιλογές σύνδεσης θέτοντας τις παραμέτρους στο configuration αρχείο της εφαρμογής.

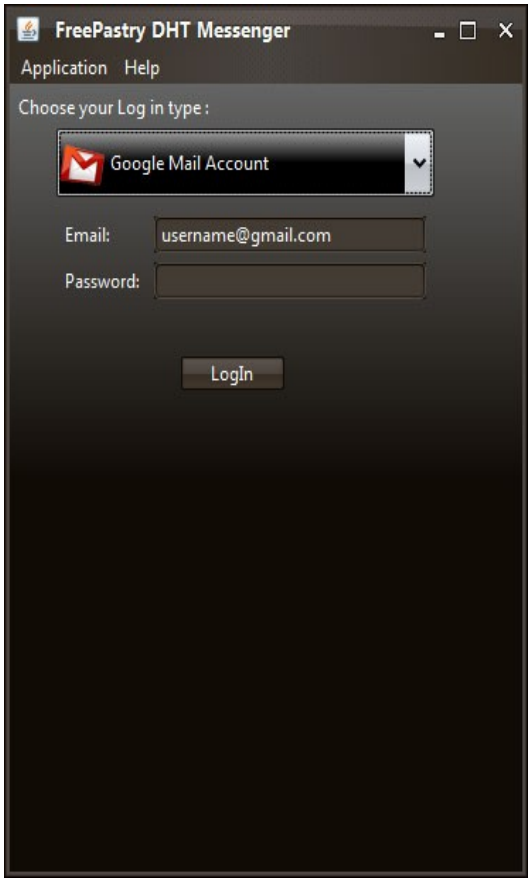
Ακολουθεί μία τυπική ρύθμιση αυτών των παραμέτρων και η επεξήγηση τους.

```
bindport 9001
bootport 9000
bootAddress 192.168.1.10
UPnpSettingsOn true
ReplicationLevel 5
SSLmode true
KeyStoreFileName 69328E581184B05A4E339C5F70993E710E316B73.store
```

Οι παράμετροι που μπορούμε να ορίσουμε είναι:

- bindport : Η θύρα επικοινωνίας του κόμβου μας.
- bootport: Η θύρα επικοινωνίας του κόμβου στον οποίο συνδεόμαστε για την εισαγωγή μας στο δίκτυο.
- bootaddress: Η IP διεύθυνση του κόμβου σύνδεσης.
- UpnpSettingsOn: Το θέτουμε ως true αν στο LAN μας είναι ενεργοποιημένες οι ρυθμίσεις Upnp (Universal Plug and Play).
- ReplicationLevel: Ο αριθμός αντιγράφων που θα κρατάει ο PAST.
- SSLmode: Το θέτουμε ως true αν θέλουμε να χρησιμοποιήσουμε SSL σύνδεση στο δίκτυο.
- KeyStoreFileName: Το όνομα του αρχείου πού περιέχει το πιστοποιητικό για την SSL σύνδεση.

4.2 Επιλογή λογαριασμού εισαγωγής στο σύστημα



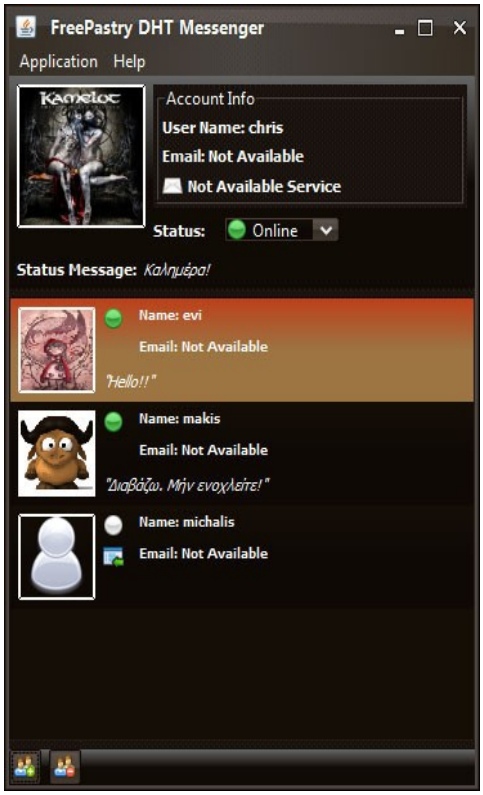
Σχήμα 10: Επιλογή τύπου Account για εισαγωγή στην εφαρμογή.

Στην εφαρμογή προσφέρονται δύο τρόποι με τους οποίους μπορεί κάποιος να εισαχθεί στην υπηρεσία. Ο πρώτος είναι χρησιμοποιώντας τον λογαριασμό email του Google που διατηρεί και ο δεύτερος είναι η εισαγωγή του ως δοκιμαστικός χρήστης που έχει όμως περιορισμένες δυνατότητες. Στην δεύτερη περίπτωση η επιλογή αποστολής email και αλλαγής του ψευδωνύμου έχουν απενεργοποιηθεί καθώς το όνομα αποτελεί το αναγνωριστικό του χρήστη και δεν θέλουμε να αλλάζει. Αντίθετα στην περίπτωση του Gmail λογαριασμού το αναγνωριστικό το αποτελεί ή email διεύθυνση του χρήστη.

Μετά την εισαγωγή του χρήστη στο σύστημα τοποθετείται στον PAST ο χειριστής κόμβου του (NodeHandle) με κλειδί αναζήτησης το αναγνωριστικό του. Έτσι λύνεται και το Rendezvous πρόβλημα καθώς κάποιος μπορεί να αναζητήσει κάποιον άλλο χρήστη σύμφωνα με το αναγνωριστικό του, να αποκτήσει τον χειριστή κόμβου του, και έτσι να αρχίσουν να ανταλλάσσουν μηνύματα. Επίσης μία λύση ακόμα αποτελεί η αποστολή του χειριστή κόμβου μέσω ενός Topic του Scribe όπως θα δούμε στην επόμενη παράγραφο και που χρησιμοποιείται στην εφαρμογή στην παρούσα φάση του project. Διατηρείται και η πρώτη μέθοδος καθώς θα γίνει προσπάθεια προσθήκης και άλλων λειτουργιών, που πιθανός να την απαιτούν, και μετά την παρουσίαση των αποτελεσμάτων της πτυχιακής.

4.3 Η λίστα επαφών και οι προσωπικές πληροφορίες

Η λίστα επαφών ενός χρήστη και οι προσωπικές του πληροφορίες που περιέχουν τα μεταβλητά στοιχεία ψευδώνυμο, εικόνα προφίλ και μήνυμα κατάστασης αποθηκεύονται και ανακτούνται από τον PAST με τα κλειδιά "αναγνωριστικό-list" και "αναγνωριστικό-info" αντίστοιχα. Αυτό δίνει την δυναμική σε ένα χρήστη να εισάγεται και να ανακτά την λίστα και τα προσωπικά του στοιχεία από οποιονδήποτε κόμβο στο δίκτυο.



Σχήμα 11: Το κυρίως Interface της εφαρμογής

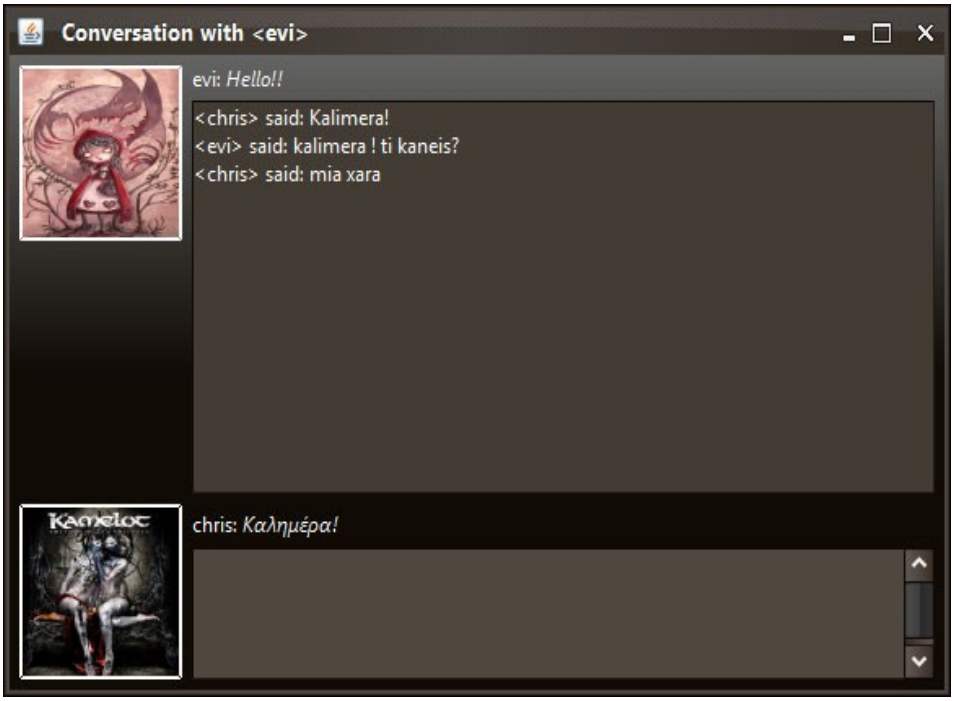
Στην παραπάνω φωτογραφία χρήσης της εφαρμογής βλέπουμε μία άποψη της λίστας επαφών του χρήστη chris ο οποίος έχει 2 ενεργούς φίλους, τους evi και makis, όπου παρέχονται όλα τα μεταβλητά προσωπικά στοιχεία τους συμπεριλαμβανομένου και της κατάστασης τους. Ο χρήστης michalis έχει μία παραπάνω ένδειξη που σημαίνει ότι δεν υπάρχει ή δεν μας έχει στην λίστα του ως φίλο ακόμα. Για να φύγει αυτή η ένδειξη από την επαφή θα πρέπει ο michalis να έχει στην λίστα του τον χρήστη chris και ταυτόχρονα να βρεθεί online μαζί του. Από την στιγμή πού θα γίνει αυτό θα ενεργοποιηθούν οι επαφές και στους δύο χρήστες και θα μπορεί ο καθένας να παίρνει ενημερώσεις για την κατάσταση και τα προσωπικά στοιχεία του άλλου όταν αυτά αλλάζουν. Σε περίπτωση διαγραφής του ενός από τον άλλο οι ενημερώσεις σταματούν και ο χρήστης παρουσιάζεται μόνιμα offline.

Η ενημέρωση των στοιχείων και της κατάστασης χρήστη γίνεται μέσω του Scribe και του Topic που διατηρεί ο κάθε χρήστης και οι επαφές του. Αυτό που έχει να κάνει είναι κατά την εισαγωγή του στο σύστημα και αφού ζητήσει και πάρει την λίστα επαφών του από τον PAST να γραφτεί στα Topic της κάθε επαφής στην λίστα του με κλειδί "αναγνωριστικό επαφής-liveness". Συγκεκριμένα τα στοιχεία που δημοσιεύει ο κάθε χρήστης είναι : η κατάσταση του , ο χειριστής κόμβου του (κάνει ενημέρωση τον προηγούμενου), η λίστα των ενεργών και μη μπλοκαρισμένων χρηστών και την έκδοση των προσωπικών του στοιχείων. Όταν πάρει το μήνυμα του κάποιος θα ελέγξει αν βρίσκετε στους ενεργούς χρήστες του χρήστη που έστειλε το μήνυμα. Αν όχι , δεν θα μπορεί να του στείλει μηνύματα και θα τον βλέπει σε κατάσταση offline στην λίστα του. Αν δεν είναι μπλοκαρισμένος θα ενημερώσει τον χειριστή κόμβου της επαφής και θα ελέγξει αν η έκδοση των προσωπικών στοιχείων της επαφής είναι παλαιότερα από την έκδοση πού στάλθηκε. Αν διαπιστώσει ότι υπάρχει νεότερη θα κάνει μία αναζήτηση στον PAST για να πάρει τα ανανεωμένα. Αυτό γίνεται γιατί τα προσωπικά στοιχεία περιέχουν και την φωτογραφία προφίλ που δεν είναι καλή τακτική να την στέλνουμε συνεχώς μέσω του ίδιου Topic καθώς είναι υπεύθυνο για την "ζωτικότητα" ενός χρήστη.

Η "ζωτικότητα" (liveness) μίας επαφής στην λίστα γίνεται μειώνοντας έναν μετρητή ο οποίος όταν φτάσει στην τιμή "μηδέν" (0) θεωρεί τον χρήστη offline. Όταν όμως μας έρχεται ένα μήνυμα από την επαφή αυτή,μέσω του Scribe και με κατάσταση διαφορετική του offline ο μετρητής παίρνει ξανά την αρχική του τιμή. Η ακρίβεια για το αν ένας χρήστης είναι online εξαρτάται σε μεγάλο βαθμό από τον συγχρονισμό της αποστολής μηνυμάτων στο Topic του και το πόσο συχνά θα μειώνεται ο μετρητής που του αντιστοιχεί στο άλλο άκρο.

4.4 Έναρξη διαλόγου μεταξύ δύο χρηστών

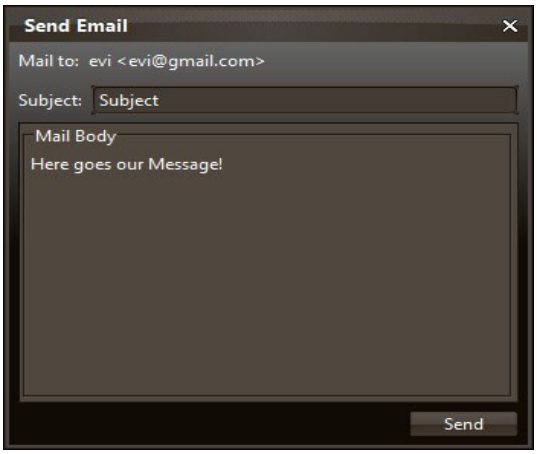
Για να ανοίξουμε μία συζήτηση με μία επαφή στην λίστα μας αυτή θα πρέπει να είναι online και να μην μας έχει μπλοκάρει. Για να γίνει αυτό θα πρέπει να επιλέξουμε την επαφή και με πάτημα του δεξιού κουμπιού τού ποντικιού να επιλέξουμε από το αναδυόμενο μενού πού θα εμφανιστεί την επιλογή "Open Chat". Στην συνέχεια δημιουργείται το απαραίτητο AppSocket για να γίνει η αποστολή μηνυμάτων εφικτή. Μόλις στείλουμε ένα μήνυμα στο παραλήπτη θα δεσμευτεί επίσης ένα AppSocket και αυτομάτως θα ανοίξει το παράθυρο διαλόγου



Σχήμα 12: Στιγμιότυπο διαλόγου μεταξύ δύο χρηστών.

4.4 Αποστολή E-mail

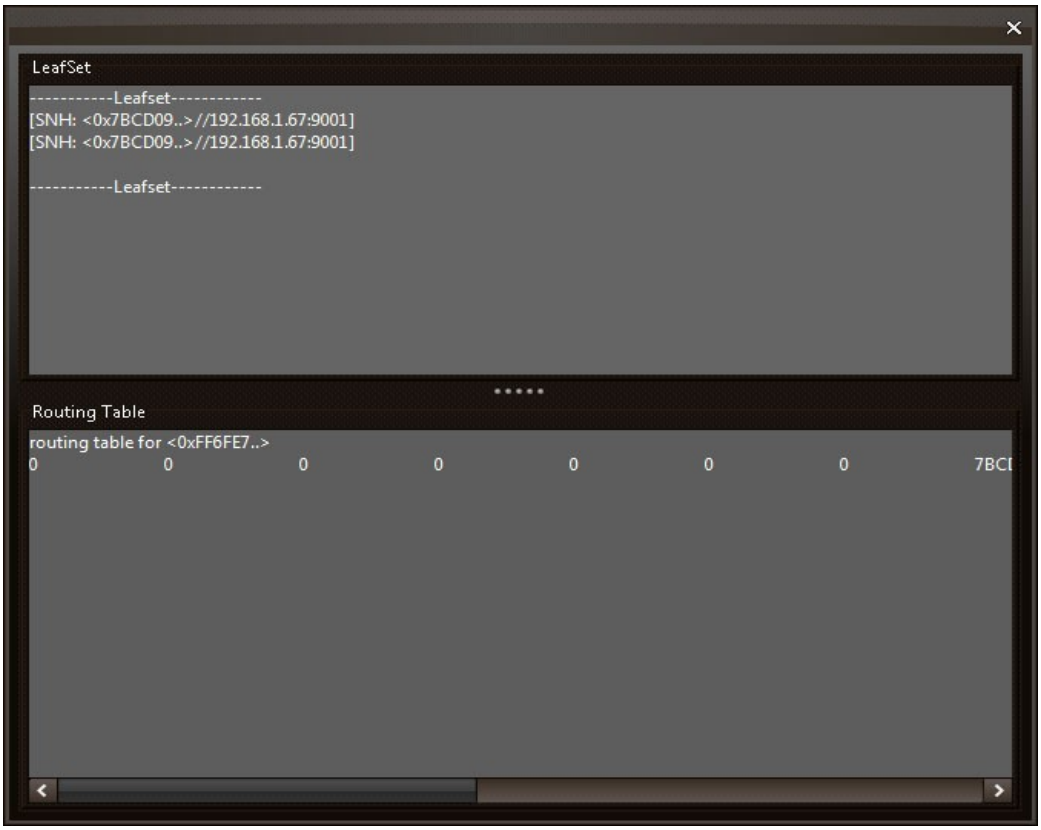
Η εφαρμογή μας επιτρέπει να στέλνουμε και email μηνύματα (π.χ. Σε περίπτωση που κάποιος χρήστης είναι offline). Αυτό γίνεται χρησιμοποιώντας την βιβλιοθήκη Javamail. Η επιλογή βρίσκεται στο αναδυόμενο μενού αν επιλέξουμε την επαφή που θέλουμε να το στείλουμε. Σε κάθε email που τοποθετείται σαν πρόθεμα του θέματος η ετικέτα “FreePastryDHTMessenger” και μετά το θέμα που επιλέγει ο χρήστης.



Σχήμα 13: Η φόρμα συμπλήρωσης και αποστολής e-mail.

4.4 Οι πίνακες που διατηρεί ένας κόμβος

Συμπληρωματικά στην εφαρμογή μας έχουμε συμπεριλάβει και την εμφάνιση των πινάκων πού διατηρεί ένας κόμβος. Αυτοί είναι το LeafSet (σύνολο φύλλων) και το Routing Table (πίνακας δρομολόγησης). Ο πίνακας NeighborhoodSet (σύνολο γειτονίας) δεν συμπεριλαμβάνεται στην FreePastry από την έκδοση 1.4 και μετά καθώς δεν απαιτείται για την δρομολόγηση ενός μηνύματος. Οι πίνακες αυτοί είναι προσβάσιμοι από το μενού Application-> FreePastryTables.



Σχήμα 14: Οι πίνακες που διατηρεί ένας κομβος στην FreePastry.

Επίλογος

Κάνοντας μία κριτική των μεθόδων που παρέχονται από βιβλιοθήκη Freepastry σε συνδυασμό με την τελική υλοποίηση της εφαρμογής μπορούμε να πούμε πώς το *Rendezvous* δύο χρηστών, πού ήταν και το πρόβλημα της επικοινωνίας μεταξύ τους σε ένα ομότιμο σύστημα με κατανεμημένο πίνακα κατακερματισμού, ήταν αρκετά εύκολο και αποδοτικό είτε αυτό γίνει μέσω αναζήτησης στον πίνακα είτε μέσω αποστολής του χειριστή κόμβου του ενός στον άλλο μέσω του Scribe. Επίσης η διαθεσιμότητα των δεδομένων που αποθηκεύουμε στον πινάκα μας "απελευθερώνει" από το να τα αποθηκεύουμε σε ένα συγκεκριμένο και κεντρικό σημείο αποφεύγοντας ένα μοναδικό σημείο αποτυχίας (single point of failure). Το αρνητικό είναι ότι για την ενημέρωση της "ζωτικότητας" των χρηστών στέλνονται συνεχώς μηνύματα μέσω του δικτύου επικάλυψης που προσδίδει στην ήδη μεγάλη κίνηση πακέτων μέσω αυτού. Η βέλτιστη περίπτωση για κάτι τέτοιο από άποψη αριθμού των πακέτων αλλά και ταχύτητας ήταν μεσώ πολυδιανομής. Η αποστολή απλών μηνυμάτων ή το διάβασμα από μία εγγραφή στον πινάκα κατακερματισμού που να δηλώνει την κατάσταση του χρήστη είναι μη αποδοτικές και αργές μέθοδοι. Επίσης λόγω της χρησιμοποίησης του Gmail σαν πιστοποίηση του χρήστη υπάρχει πιθανότητα σε περίπτωση που η υπηρεσία δεν είναι διαθέσιμη να παρουσιαστεί ένα μοναδικό σημείο αποτυχίας.

Σαν κριτική τώρα της FreePastry ως API μπορούμε να πούμε ότι το σύνολο των εντυπώσεων είναι θετικές παρότι είναι πειραματική τεχνολογία. Ιδιαίτερη προσοχή αξίζει η προσπάθεια εξομοίωσης των υποδοχών (Sockets) ενός συστήματος με το Application Socket Interface ώστε να καταφέρνει να προσπερνά διάφορα φράγματα χαμηλής συνδεσιμότητας. Το γεγονός όμως ότι είναι πειραματική τεχνολογία και με την υποστήριξη μικρής ομάδας ατόμων οδήγησε στην εμφάνιση κάποιων bugs τα οποία παρατηρήθηκαν, δύο από τα οποία είναι :

- Ένα bug του συστήματος καταγραφής της FreePastry πού μας εμφανίζει πολλές φόρες, κατά την εισαγωγή του κόμβου στο δίκτυο, ότι μας έρχονται μηνύματα από άγνωστες διευθύνσεις τα οποία και απορρίπτει . Δεν θα έπρεπε να εμφανίζονται. Σαν προσωρινή λύση η ομάδα του project προτείνει την ανακατεύθυνση των γεγονότων καταγραφής σε αρχείο και αυτό μόνο για να μην φαίνονται.

- Ένα bug όταν γίνεται χρήση στρώματος SSL που εμφανίζεται κατά την αντιγραφή (Replication) των αντικειμένων που βάζουμε στον κατανεμημένο πίνακα κατακερματισμού από ένα συγκεκριμένο μέγεθος και πάνω (από δοκιμές που γίνανε είναι περίπου 15-20 KB). Η αλλαγή του buffer μεταφοράς κρυπτογραφημένων δεδομένων και μη δεν απέδωσε και κάποια λύση στο θέμα δεν έχει δοθεί στην mailing list του project. Προς το παρόν η εφαρμογή μπορεί να λειτουργήσει σωστά χωρίς το στρώμα SSL. Υποστηρίζεται όμως κατ' επιλογή του χρήστη και πιθανός με την επόμενη έκδοση της FreePastry να είναι πλήρως αξιοποιήσιμο ή αν δοθεί κάποια μέθοδος σαν απάντηση στις online συζητήσεις για να το αποφύγουμε.

Τις online συζητήσεις μπορεί κάποιος να τις παρακολουθήσει στο: https://mailman.rice.edu/pipermail/freepastry-discussion-l/ όπως επίσης να επιβεβαιώσει την ύπαρξη αυτών των σφαλμάτων.

Βιβλιογραφία

Δημοσιεύσεις

- [1] An Overview on Peer-to-Peer Information Systems
  Karl Aberer, Manfred Hauswirth

- [2] Storage management and caching in PAST, a large-scale,persistent peer-to-peer storage utility
  Antony Rowstron , Peter Druschel

- [3] Scribe: A large-scale and decentralized application-level multicast infrastructure
  Miguel Castro, Peter Druschel, Anne-Marie Kermarrec and Antony Rowstron

- [4] Exploiting network proximity in peertopeer overlay networks
  Miguel Castro, Peter Druschel, Y. Charlie Hu, Antony Rowstron.

- [5]Looking Up Data In P2P Systems
  Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica
  MIT Laboratory for Computer Science

- [6] Pastry Overview
  http://www.freepastry.org/PAST/overview.pdf

Βιβλία

- [7] Δίκτυα υπολογιστών
  Tanenbaum, Andrew S.

- [8] Java Προγραμματισμος
  Deitel&Deitel

Σύνδεσμοι

- [9] http://en.wikipedia.org/wiki/Instant_messaging
- [10] http://en.wikipedia.org/wiki/Overlay_network
- [11] http://en.wikipedia.org/wiki/Distributed_hash_table
- [12] http://en.wikipedia.org/wiki/Napster
- [13] http://en.wikipedia.org/wiki/File_sharing
- [14] http://en.wikipedia.org/wiki/Usenet
- [15] http://en.wikipedia.org/wiki/Peer-to-peer
- [16] http://ezinearticles.com/?How-Skype-Works&id=496462
- [17] http://www.milkaddict.com/how-windows-live-messenger-works/

- [18] http://www.slideshare.net/rvenkatesh25/introduction-to-peertopeer-networks
- [19] http://www.p2p-security.com/about-p2p/the-four-generations
- [20] https://trac.freepastry.org/wiki/FreePastryTutorial
- [21] http://www.oracle.com/technetwork/java/index-jsp-139225.html
- [22] http://el.wikipedia.org/wiki/SSL