ΥΛΟΠΟΙΗΣΗ PEER-TO-PEER ΠΡΩΤΟΚΟΛΛΩΝ ΣΕ JXTA

Πέτρος Καραγκουνίδης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πολυτεχνείο Κρήτης
Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού και Δικτυακών Εφαρμογών

Επιβλέπων Καθηγητής
Βασίλης Σαμολαδάς

ΧΑΝΙΑ, ΙΟΥΛΙΟΣ 2006

ΠΕΡΙΕΧΟΜΕΝΑ

ΣΧΗΜΑΤΑ

ΠΙΝΑΚΕΣ

ΕΥΧΑΡΙΣΤΙΕΣ

Ποιον να πρωτοευχαριστήσω σε μια κόλα Α4 και ακόμα περισσότερο με ποια σειρά? Ξεκινώντας από τα άμεσα συνδεδεμένα πρόσωπα με αυτήν την διπλωματική, θέλω να ευχαριστήσω τον επιβλέποντα μου καθηγητή Κο Σαμολαδά που με ανέχτηκε για ένα χρόνο και κάτι παραπάνω, και για την Χρυσσάνη Ξηρουχάκη που δέχτηκε να την πρήζω για κάτι που ασχολιόταν τρία χρόνια στο παρελθόν. Οι συνθήκες εργασίας ήταν πραγματικά ευχάριστες μαζί τους.

Ευχαριστώ όλη μου την κομπανία στα Χανιά για τις τρελές καταστάσεις στην αντιφατική αυτή πόλη και δεν θέλω να υπονοήσω ότι φταίνε αυτοί για την καθυστέρηση της ολοκλήρωσης της διπλωματικής μου. Με σειρά βάρους ευχαριστώ τον κοινωνικό αλλά καταπιεστικό Βάρδο, τον "υποστηρικτή" αλλά αγχωτικό Ντούμπα, τον εξυπηρετικό αλλά φονταμενταλιστή σε κάθε τομέα Μύγα, τον φιλότιμο αλλά αργόοοοοο Ποντίκη, τον φιλόσοφο αλλά καμία φορά σνομπ Πανξ, τον πολυμαθή αλλά βαρεμένο Λύκο, τον ανεκτικό αλλά κακόγουστο στα άτομα που διαλέγει για συγκατοίκηση Βοσκό, τον αυθόρμητο αλλά μηδενιστή δυτικού τύπου Τροπολιτσά, τον σταθερό αλλά μπεκρή Στρατηγό, τον ευαίσθητο αλά Τούρκο, το πιο κατάλληλο άτομο για να επικοινωνήσω αλλά διπλωμάτη Κομμάτια, τον διακριτικό αλλά αναποφάσιστο Ντίνο, την καλοσυνάτη αλλά "ξύλινη" Δώρα, την "επιθετική" αλλά χαμένη Μαρία, τον αυθεντικό αλλα δυσνόητο Ελλασώνα και τον ιδιοφυέστατο αλλά παρτάκια Καλαμπόκα. Επιπλέον, ευχαριστώ για την φιλία του τον αξέχαστο αλλά απρόσεκτο φίλο Πιι που δυστυχώς δεν θα μπορέσει να διαβάσει ποτέ αυτή την γραμμή...

Επίσης, δεν ξέρω τι να πω για την "μικρή" Σου, που τόσο με βοήθησε τους δυο τελευταίους αγχωτικούς μήνες και τόσα άκουσε από τα mini ξεσπάσματα μου. Υπόσχομαι ότι το άλλο καλοκαίρι θα έχω περισσότερο ελεύθερο χρόνο.

Φυσικά, δεν ξεχνώ τους υπερβολικά προχωρημένους και ανοιχτόμυαλους γονείς μου , ούτε τις "ορέξεις" τους αποτέλεσμα των οποίων ήταν η ύπαρξη μου.

Πάνω από όλα ευχαριστώ την Τυχαιότητα που μέχρι τώρα μου τα κανονίζει όλα ωραία....

Κεφάλαιο 1

ΕΙΣΑΓΩΓΗ

Τα υπολογιστικά συστήματα διαχωρίζονται σε κεντρικοποιημένα και σε κατανεμημένα συστήματα. Τα κατανεμημένα συστήματα βασίστηκαν κυρίως στο client-server μοντέλο, κατά το οποίο κάποιοι computer nodes εκτελούν τα χρέη ενός εξυπηρετητή (server) και κάποιοι άλλοι που συνδέονται με τους servers εκτελούν τα χρέη ενός πελάτη (client). Έτσι ο ρόλος κάθε computer node σε ένα δίκτυο που είναι υλοποιημένο με βάση τις αρχές που διέπει ένα μοντέλο client-server είναι ξεκάθαρος. Ή θα είναι server και θα εξυπηρετεί αιτήσεις που θα δέχεται από απομακρυσμένους υπολογιστές ή θα είναι client και θα στέλνει αιτήσεις σε κάποιον απομακρυσμένο server περιμένοντας κάποια ανταπόκριση. Ακόμα και το Internet είναι κατά κόρον πλασμένο στο καλούπι του client-server μοντέλου.

Παρόλα αυτά, η γέννηση νέων αναγκών και η συνεχώς αυξανόμενη εισροή χρηστών στο Internet (που σημαίνει περισσότεροι computer nodes συνδεδεμένοι και περισσότερο content διαθέσιμο), άρχισαν να καθιστούν το μοντέλο client-server αναποτελεσματικό σε κάποιες περιπτώσεις. Ένα νέο μοντέλο έπρεπε να υιοθετηθεί, με το οποίο οι χρηστές θα μπορούσαν να εκμεταλλευτούν τους διαθέσιμους πόρους του Internet με τον πιο αποδοτικό τρόπο. Το νέο μοντέλο που προέκυψε ήταν το peer-to-peer, κατά το οποίο οι clients συνδέονται απευθείας μεταξύ τους χωρίς κανένα παρεμβαλλόμενο. Η φύση, δηλαδή, του computer node σε ένα peer-to-peer δίκτυο είναι διττή, εφόσον αυτός είναι και server και client ταυτόχρονα, μιας και μπορεί να προσφέρει και να λάβει πόρους την ίδια στιγμή. Η αποτελεσματικότητα του peer-to-peer μοντέλου φαίνεται αν συλλογιστούμε ένα δίκτυο στο οποίο η ζήτηση που προσδιορίζεται από τους clients είναι πολύ ψηλότερη από την πρόσφορα των servers. Το μοντέλο client-server μόνο σε δικτυακό κραχ θα μπορούσε να οδηγήσει σε αυτή την περίπτωση, ενώ η απευθείας σύνδεση και ανταλλαγή δεδομένων μεταξύ των συμμετεχόντων χωρίς την παρεμβολή κεντρικοποιημένων servers θα μπορούσε να ξεπεράσει

πολύ ευκολότερα αυτά τα προβλήματα οικονομικής φύσης.

Έτσι, κατανοώντας την χρησιμότητα της αρχιτεκτονικής P2P, έπρεπε πολλές εφαρμογές να προγραμματιστούν με βάση αυτή την αρχιτεκτονική και όχι με το συγκεντρωτικό client-server μοντέλο. Όμως, ενώ η αρχιτεκτονική ενός peer-to-peer συστήματος φαίνεται αρκετά απλή, η υλοποίηση του είναι σύνθετη λόγω της δυναμικής και απρόβλεπτης συμπεριφοράς των χρηστών. Η υψηλή πολυπλοκότητα που διέπει την υλοποίηση ενός τέτοιου συστήματος καθιστά ως πρόβλημα την ανάπτυξη σύνθετων P2P μηχανισμών. Μηχανισμοί, που θα αποτελέσουν την βάση και θα διευκολύνουν την σύνθεση ολοκληρωμένων P2P εφαρμογών. Ευτυχώς, όμως, για τους προγραμματιστές υπάρχουν πλατφόρμες ανάπτυξης λογισμικού οι οποίες παρέχουν κλάσεις και μεθόδους που ευνοούν την υλοποίηση P2P συστημάτων και υπηρεσιών.

1.1 - Ορισμός Προβλήματος

1.1.1 - Ο Στόχος μας

Ο βασικός στόχος της συγκεκριμένης διπλωματικής είναι η υλοποίηση κάποιων πολύπλοκων peer-to-peer μηχανισμών που θα περιγραφτούν λεπτομερέστερα αργότερα. Σκοπός είναι να μπορεί κάποιος προγραμματιστής να αναπτύξει high-level εφαρμογές, οι οποίες θα έχουν σαν βάση αυτούς τους μηχανισμούς. Τέτοιου τύπου εφαρμογές αναπτύχθηκαν και στα πλαίσια της διπλωματικής για την επαλήθευση και την αποτίμηση της λειτουργίας των μηχανισμών. Οι εφαρμογές αυτές συνθέτουν στο σύνολο τους ένα P2P Sharing System, το Guerilla. Στο δίκτυο του Guerilla ένας χρηστής έχει ουσιαστικά δυο δυνατότητες: 1) Να εκδώσει μια εικονική ταινία[1] 2) Να στείλει ένα query ζητώντας να μάθει για το ποιοι χρήστες, που είναι συνδεδεμένοι στο δίκτυο του Guerilla, φιλοξενούν έναν αριθμό, που θέτει αυτός, από κάποιες ταινίες. Οι high-level μηχανισμοί, για λόγους συντομίας, θα λέγονται Guerilla μηχανισμοί και οι εφαρμογές πάνω σε αυτούς Guerilla applications.

[1] Ο κάθε χρήστης έχει αποθηκευμένο στην τοπική του cache ένα συγκεκριμένο dataset που περιέχει τίτλους ταινιών, τις κατηγόριες στις οποίες ανήκουν και τα αντίστοιχα popularities τους (δηλαδή την ζήτηση που έχει η εκάστοτε ταινία). Ο κάθε χρήστης μπορεί να εκδώσει οποιαδήποτε από τις ταινίες που περιέχονται στο dataset σαν να ήταν αποθηκευμένες στην τοπική cache του.

1.1.2 - Προκλήσεις

Κύριο χαρακτηριστικό ενός P2P δίκτυο, στο οποίο μπορεί να βρίσκονται χιλιάδες peers συνδεδεμένοι, που ο κάθε ένας από αυτούς μπορεί να παρέχει δυναμικά ποικιλία πόρων και μπορεί να δρα ανεξάρτητα, είναι η πολύπλοκη υλοποίηση του. Φανταστείτε πόσο πολυπλοκότητα κρύβει η υλοποίηση ενός δικτύου στο οποίο ο κάθε συμμετέχων μπορεί να συνδεθεί και να αποσυνδεθεί, να προσφέρει και να αποσύρει τους πόρους του ή και να αλλάξει την διεύθυνση του, η οποία μπορεί και να μην είναι ποτέ stable, οποία στιγμή θέλει, κάτι για το οποίο φυσικά δεν μπορεί να κάνει ένας κεντρικοποιημένος server αν θέλει να φημίζεται για την αξιοπιστία του. Σε ένα τέτοιο χαοτικό σύστημα, με αρκετά υψηλή εντροπία, οι μηχανισμοί που θα αναπτυχθούν θα πρέπει να προσαρμόζονται σε ένα άκρως δυναμικό περιβάλλον, κάτι που σαφώς αυξάνει την πολυπλοκότητα της υλοποίησης τους.

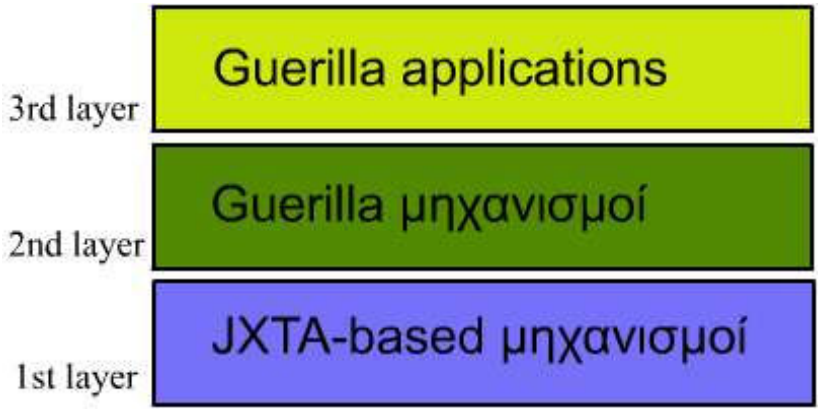
1.1.3 - Η Λύση

Η ανάπτυξη πολύπλοκων peer-to-peer μηχανισμών είναι, γενικά, ένα δύσκολο εγχείρημα εξαιτίας των λόγων που περιγράφτηκαν προηγουμένως (δυναμική και απρόβλεπτη συμπεριφορά των peers). Η υλοποίηση, όμως, απλοποιείται μέχρι κάποιο βαθμό αν υπάρχει ένα framework το οποίο παρέχει ένα σύνολο από standards που θα υποστηρίζουν την ανάπτυξη peer-to-peer μηχανισμών.

Ένα τέτοιο framework είναι και το JXTA, το οποίο παρέχει βασικές

συναρτήσεις απαραίτητες για ένα peer-to-peer δίκτυο. Όλες οι εφαρμογές ή οι υπηρεσίες που λειτουργούν βασισμένα στις αρχές του peer-to-peer μοντέλου μοιράζονται μεταξύ τους κοινές ιδιότητες, ανεξάρτητα από την λειτουργικότητα τους. Μπορεί η P2P εφαρμογή ICQ να χρησιμοποιείται για instant messaging και η P2P εφαρμογή eMule για file transferring, αλλά και οι δυο εφαρμογές προϋποθέτουν για παράδειγμα την εύρεση κάποιων peers ή την ανταλλαγή μηνυμάτων μεταξύ των peers. Η πλατφόρμα JXTA διευκολύνει την υλοποίηση τέτοιων βασικών peer-to-peer συμπεριφορών. Παρέχοντας διάφορους μηχανισμούς και υπηρεσίες για να καθίσταται απλούστερο το peer-to-peer programming, δίνει την δυνατότητα στον προγραμματιστή να αναπτύξει P2P εφαρμογές ή high-level P2P μηχανισμούς. Αυτοί οι JXTA-based μηχανισμοί είναι που χρησιμοποιήθηκαν για να αναπτυχθούν πάνω σε αυτούς higher-level P2P μηχανισμοί (οι Guerilla μηχανισμοί).

Όποτε, όπως φαίνεται και από το σχήμα 1, οι JXTA-based μηχανισμοί είναι στο πιο low-level και χρησιμοποιούνται για την ανάπτυξη των high-level Guerilla μηχανισμών, οι οποίοι με την σειρά τους αποτελούν την βάση για να «χτιστούν» οι Guerilla applications.



Σχήμα 1: Αρχιτεκτονική του Guerilla

Κεφάλαιο 2

ΣΧΕΤΙΚΗ ΕΡΓΑΣΙΑ

Τα κατανεμημένα συστήματα κατηγοριοποιούνται με βάση την αρχιτεκτονική τους. Οι βασικές αρχιτεκτονικές κατανεμημένων συστημάτων είναι η client-server και η peer-to-peer αρχιτεκτονική. Κατά το client-server μοντέλο το περιεχόμενο για κοινή χρήση είναι αποθηκευμένο σε κεντρικούς υπολογιστές με γνωστή διεύθυνση που λέγονται servers, ενώ στο peer-to-peer μοντέλο το περιεχόμενο είναι αποθηκευμένο σε όσους υπολογιστές είναι συνδεδεμένοι στο δίκτυο.

Τα συστήματα που βασίζονται στην P2P αρχιτεκτονική έχουν αρκετά πλεονεκτήματα σε σχέση με αυτά που βασίζονται στο client-server μοντέλο, αλλά υστερούν ως προς την πολυπλοκότητα της υλοποίησης τους. Τα P2P συστήματα είναι μεγάλος πονοκέφαλος για τους προγραμματιστές, εφόσον αυτοί πρέπει να αναπτύξουν μηχανισμούς προσαρμοστικούς στο άκρως δυναμικό και απρόβλεπτο περιβάλλον που τα χαρακτηρίζει.

Τα περισσότερα peer-to-peer συστήματα διαχωρίζονται, κυρίως, αναλόγως με το πόσο κεντρικοποιημένα είναι. Υπάρχουν τα pure P2P συστήματα που είναι τα πιο αποκεντρωμένα και ο ρόλος του server είναι σχεδόν εκμηδενισμένος, τα hybrid P2P συστήματα που υπάρχει μεν server, αλλά δεν φυλάει τα πραγματικά δεδομένα δε, και τα mixed P2P συστήματα που συγκεντρώνουν χαρακτηριστικά από τα δύο προηγούμενα μοντέλα. Επιπλέον, υπάρχουν και τα anonymous P2P συστήματα τα οποία εμμένουν στην ανωνυμία του χρήστη.

Τα P2P συστήματα αναλόγως με τη λειτουργικότητα τους μπορούν να διαχωριστούν στις εξής κατηγορίες: Distributed computing, file sharing και collaboration.

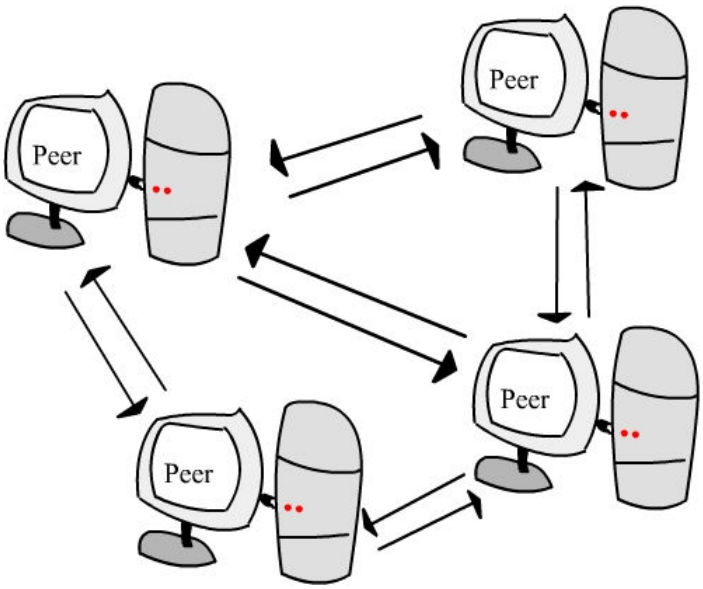
Σκοπός αυτής της διπλωματικής είναι η υλοποίηση πολύπλοκων high-level P2P μηχανισμών, οι οποίοι θα αποτελούν την βάση για την ανάπτυξη P2P εφαρμογών. Τέτοιες εφαρμογές αναπτύχθηκαν και στα πλαίσια της διπλωματικής για την επαλήθευση και την αποτίμηση της λειτουργίας των μηχανισμών. Οι εφαρμογές αυτές συνθέτουν στο σύνολο τους ένα P2P Sharing System, το Guerilla, του οποίου η αρχιτεκτονική, οι αλγόριθμοι και οι μηχανισμοί του διατυπώθηκαν από την Χρυσάννη Ξηρουχάκη στα πλαίσια του μεταπτυχιακού της.

Η υλοποίηση του Guerilla Sharing System, το οποίο περιλαμβάνει τους high-level μηχανισμούς, αλλά και τις εφαρμογές που βασίζονται σε αυτούς, υλοποιήθηκε με την χρήση της πλατφόρμας JXTA, η οποία παρέχει την κατάλληλη τεχνολογία για την ανάπτυξη P2P υπηρεσιών και εφαρμογών.

2.1 - Peer-to-Peer

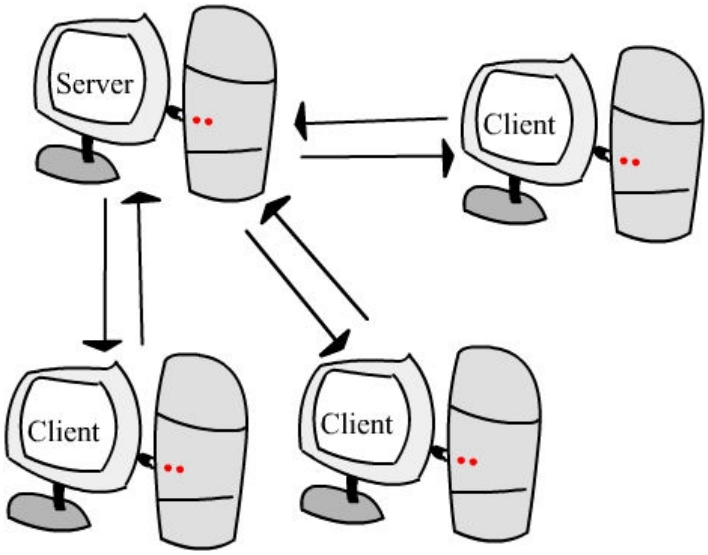
2.1.1 - Ορισμός P2P Δικτύου

Ένα P2P computer network θεωρείται [XKTC03, Wiki, KR03] ουσιαστικά το δίκτυο που βασίζεται στους πόρους όλων των υπολογιστών που συμμετέχουν ενεργά σε αυτό. Είναι δηλαδή ένα δίκτυο που τις υπηρεσίες του δεν τις προσφέρει ένας κεντρικός υπολογιστής, ένας server δηλαδή, αλλά οι ίδιοι οι nodes που συνδέονται σε αυτό το δίκτυο. Έτσι κάθε συνδεδεμένος computer node που συνδέεται στο P2P network μπορεί να είναι ταυτόχρονα και server, εφόσον παρέχει υπηρεσίες, αλλά και client εφόσον ζητά και λαμβάνει υπηρεσίες από τους άλλους nodes του δικτύου. Επομένως, ένα P2P computer network είναι αποκεντρωτικό, εφόσον το περιεχόμενο μπορεί να μοιραστεί για κοινή χρήση από όσους είναι συνδεδεμένοι στο δίκτυο (σχήμα 2).



Σχήμα 2: Peer-to- Peer Μοντέλο

Αντιθέτως, το client-server μοντέλο (σχήμα 3) είναι συγκεντρωτικό εφόσον μόνο ο server μπορεί να αποθηκεύσει και να μοιράσει περιεχόμενο ή να προσφέρει υπηρεσίες, αχρηστεύοντας έτσι τις δυνατότητες όλων αυτών που συμμετέχουν στο δίκτυο. Σε ένα P2P network μπορεί να γίνει εκμετάλλευση όλων των πόρων των συνδεδεμένων υπολογιστών.



Σχήμα 3: Client-Server Μοντέλο

Το P2P δίκτυο (peer-to-peer δηλαδή, ή ομότιμος μεταξύ ομότιμου) ονομάζεται έτσι γιατί οι peers θεωρούνται ομότιμοι μεταξύ τους, εφόσον όλοι έχουν τον ίδιο ρόλο στο σύστημα. Μπορεί είτε να παρέχουν, είτε να ζητήσουν υπηρεσίες, περιεχόμενο ή πόρους αντίθετα με το client-server μοντέλο όπου οι ρόλοι είναι διαχωρισμένοι σε αυτό του εξυπηρετητή και σε αυτό του πελάτη.

2.1.2 - Πλεονεκτήματα P2P Δικτύων

Στο μοντέλο του client-server δημιουργούνται βασικά προβλήματα εξαιτίας του γεγονότος ότι υπάρχει ένας μόνο περιορισμένος αριθμός κεντρικών εξυπηρετητών που παρέχουν της υπηρεσίες τους στους συνδεδεμένους σε αυτούς clients. Τα προβλήματα αυτά ένα P2P δίκτυο, στο οποίο όλοι είναι clients και servers ταυτόχρονα, ξεπερνιούνται. Σύμφωνα με το [XKTC03, Wiki] τα πλεονεκτήματα ενός P2P δικτύου είναι τα εξής:

Ανοχή σε σφάλματα:
Καταρχήν, στο μοντέλο client-server αν καταρρεύσει ο server καταρρέει όλη η εφαρμογή, ενώ στο P2P δίκτυο αν καταρρεύσει ένας node τότε η ζημία είναι πολύ μικρότερη έως και μηδενική.

Κλιμάκωση:
Ένα peer-to-peer δίκτυο χαρακτηρίζεται από scalability. Η αυξημένη εισροή clients στο μοντέλο client-server σημαίνει πιο αργό data transfer επειδή το bandwidth θα πρέπει να μοιραστεί σε περισσότερους υπολογιστές, ενώ στα P2P συστήματα περισσότεροι computer nodes σημαίνει ακόμη μεγαλύτερη χορήγηση πόρων, εφόσον ο κάθε peer ατομικά παρέχει πόρους στην υπηρεσία του κοινού καλού. Με λίγα λόγια γίνεται καλύτερη αξιοποίηση πόρων εφόσον όλοι προσφέρουν, αντιθέτως με το client-server μοντέλο που όλοι παίρνουν (εκτός από τους ελάχιστους σε αριθμό "γενναιόδωρους" servers). Η λογική του P2P λέει ουσιαστικά ότι εφόσον η κοινωνία του Internet αποτελείται από εκατομμυρίου υπολογιστές είναι κρίμα να χαραμίζονται οι τεράστιες σε μέγεθος ικανότητες τους.

Ανωνυμία:
Διατηρείται κατά ένα πολύ μεγάλο βαθμό η ανωνυμία των συμμετεχόντων στο δίκτυο. Υπάρχουν μάλιστα αρκετές P2P αρχιτεκτονικές που δίνουν μεγάλο βάρος στην διατήρηση της ανωνυμίας των χρηστών θυσιάζοντας αρκετές φορές την απόδοση του συστήματος. Δεν είναι λίγες οι φορές που μεσοί οικονομικά

χρηστές του Internet κυνηγήθηκαν νομικά από εταιρείες επειδή χρησιμοποιούσαν συστήματα ανταλλαγής -μουσικών κυρίως- αρχείων αναγκάζοντας τους να πληρώσουν τεράστια χρηματικά ποσά. Μάλλον, αν το φαινόμενο διογκωθεί ακόμα παραπάνω, πιο σημαντική θα είναι η ανωνυμία των χρηστών παρά το πόσο γρήγορα μπορεί κάποιος να κατεβάσει μουσική και video movies. Χαρακτηριστικό είναι το μότο του ES5 "Resistance is futile, only the Anonymous will Survive" όπου τονίζεται η σημαντικότητα της ανωνυμίας σε ένα P2P δίκτυο.

Διαχείριση:
Σε ένα P2P σύστημα το administrative cost είναι μικρό. Για τους users είναι πολύ απλούστερο τεχνικά και πιο οικονομικό να γίνουν εξυπηρετητές σε ένα P2P σύστημα, παρά σε ένα σύστημα που βασίζεται στο μοντέλο client-server (τα τραβάνε οι καημένοι οι προγραμματιστές για αυτούς). Το στήσιμο ενός server ο οποίος θα παρέχει υπηρεσίες για ένα client-server σύστημα είναι σχετικά πολύπλοκη και ακριβή διαδικασία

2.1.3 - Μειονεκτήματα P2P Δικτύων

Το περιβάλλον ενός P2P δίκτυο είναι άκρως δυναμικό και απρόβλεπτο. Σε ένα P2P δίκτυο μπορεί να βρίσκονται χιλιάδες peers συνδεδεμένοι, και ο κάθε ένας από αυτούς μπορεί να παρέχει δυναμικά έναν μεγάλο αριθμό πόρων και μπορεί να δρα αυτόνομα. Ένα δίκτυο στο οποίο ο κάθε συμμετέχων μπορεί να συνδεθεί και να αποσυνδεθεί, να προσφέρει και να αποσύρει τους πόρους του ή και να αλλάξει την διεύθυνση του, η οποία μπορεί και να μην είναι ποτέ stable, οποία στιγμή θέλει, σίγουρα χρειάζεται πολύπλοκους μηχανισμούς και αλγόριθμους που θα προσαρμόζονται στις αλλαγές αυτές. Επομένως, η υλοποίηση P2P συστημάτων είναι περισσότερο πολύπλοκη από συστήματα βασισμένα στο client-server μοντέλο, όπου εκεί τα πράματα είναι περισσότερο στατικά και προβλέψιμα.

2.1.4 - Τύποι P2P Συστημάτων

Τα peer-to-peer συστήματα μπορούν να διαχωριστούν στις εξής κατηγόριες:

Pure P2P:
Οι peers δρουν σαν client και servers ταυτόχρονα χωρίς να υπάρχει κάποιο είδος

κεντρικού έλεγχου. Το πιο χαρακτηριστικό από τα pure P2P συστήματα είναι το Gnutella [Gnu, KR03]. Στο συγκεκριμένο sharing system υπάρχει απλά ένας κόμβος εκκίνησης ο οποίος παρέχει στον συνδεόμενο peer την διεύθυνση ενός γνωστού peer που είναι ήδη συνδεδεμένος στο δίκτυο. Η τοπολογία του δικτύου Gnutella είναι αδόμητη και δεν υπάρχει ιεραρχική δομή, ούτε super peers (θα αναφερθούν παρακάτω).

Hybrid P2P:
Υπάρχει κεντρικός server ο οποίος δεν κρατά τα πραγματικά δεδομένα, αλλά πληροφορίες που αφορούν τους peers, και ανταποκρίνονται σε κάθε αίτηση που σχετίζονται με αυτές τις πληροφορίες. Τα πραγματικά δεδομένα τα κρατούν οι peers και τα ανταλλάσσουν απευθείας μεταξύ τους και απλά ο server γνωρίζει τι αντικείμενα κάνει διαθέσιμα για κοινή χρήση ο κάθε ένας από αυτούς. Το Napster [KR03, XKTC03] ήταν ένα από τα πιο δημοφιλή hybrid P2P. Ο server του Napster γνώριζε ανά πάσα στιγμή τι αρχεία αποθήκευε ο κάθε peer του δικτύου. Έτσι, όταν ένας peer ρωτούσε για ένα συγκεκριμένο αρχείο στον server, αυτός έψαχνε την τοπική του cache για να βρει ποιος peer το φιλοξενεί. Ο server επέστρεφε την διεύθυνση του peer που αποθήκευε το επιθυμητό αρχείο στον peer που έκανε την αίτηση. Οι δυο peers συνδεόταν μεταξύ τους για να ανακτήσει ο ένας από τον άλλον το αρχείο.

Mixed P2P:
Συγκεντρώνουν χαρακτηριστικά pure P2P και hybrid P2P. Στα mixed P2P συστήματα εμφανίζονται οι λεγόμενοι super peers (ή ultra peers), δηλαδή peers που εξαιτίας κάποιων έξτρα δυνατοτήτων (πχ, ταχεία σύνδεση στο Internet ή υψηλή υπολογιστική ισχύ) κρατούν κάποια metadata που οι άλλοι peers δεν έχουν (όπως λίστες με τα διαθέσιμα αρχεία). Οι super peers συγκεντρώνουν τους χρήστες που βρίσκονται κοντά τους, (χρήστες ίδιου ISP ή ίδιας περιοχής) και σχηματίζουν ομάδες από peers. Ο super peer της κάθε ομάδας, στην οποία θεωρείται αρχηγός, συγκεντρώνει πληροφορίες για το τι δεδομένα φιλοξενεί ο κάθε peer της ομάδας του. Όταν ένας peer ψάχνει για ένα αρχείο απευθύνεται στον αρχηγό του και αυτός του επιστρέφει την διεύθυνση του peer που αποθηκεύει το επιθυμητό αρχείο για να συνδεθεί απευθείας μαζί του. Επιπλέον, επειδή όλοι οι super peers συνδέονται μεταξύ τους, ο ερωτηθείς αρχηγός μπορεί να ζητήσει από άλλους super peers να επιστρέψουν λίστες με διευθύνσεις δικών τους peers που φιλοξενούν το συγκεκριμένο αρχείο στον peer που έκανε την αίτηση. Ένα χαρακτηριστικό παράδειγμα mixed P2P είναι και το δίκτυο FastTrack της εφαρμογής Kazaa [Kaz] που λειτουργεί με τον τρόπο που περιγράφτηκε παραπάνω.

Anonymous P2P:
Τα anonymous P2P είναι η τελευταία γενιά των P2P δικτύων και δίνουν βάση στην ανωνυμία. Οι συνδέσεις δεν γίνονται ποτέ απευθείας μεταξύ δυο peers, αλλά τα δεδομένα διασχίζουν ένα σύνολο ενδιάμεσων κόμβων έτσι ώστε να μην καθίσταται εύκολη η αποκάλυψη του χρηστή που ζήτησε τα δεδομένα. Ο κάθε χρηστής που ζητά δεδομένα μπορεί να τα ζητά είτε για τον εαυτό του, είτε εκ μέρους κάποιου αλλού χρηστή. Παραδείγματα anonymous P2P συστημάτων είναι το ANts και ο client Azureus του γνωστού BitTorrent [Tor, Wiki].

2.1.5 - P2P Εφαρμογές

Τα P2P συστήματα αναλόγως με τη λειτουργικότητα τους μπορούν να διαχωριστούν στις εξής κατηγόριες [XKTC03]:

Distributed Computing:
Μια εργασία η οποία χρειάζεται πολλούς πόρους για να διεκπεραιωθεί μπορεί να διαιρεθεί σε πολλά "κομμάτια", δηλαδή σε πολλές μικρότερες εργασίες, και το κάθε "κομμάτι" να εκτελείται σε έναν peer ενός P2P δικτύου. Έτσι ένα δίκτυο πολλών peer που ο καθένας συνεισφέρει ένα ποσοστό των πόρων του για την εκτέλεση της εργασίας μπορεί να είναι πολύ πιο ισχυρό και αποδοτικό από ένα συνδυασμό κάποιων ισχυρών υπολογιστών που θα εκτελούσαν αυτήν την εργασία. Χαρακτηριστικό distributed computing σύστημα είναι το επιστημονικό project SETI@home [SETI], ένα πρόγραμμα ανίχνευσης εξωγήινης νοημοσύνης το οποίο χρειάζεται τεραστία υπολογιστική ισχύ και την οποία την δανείζεται από αδρανείς υπολογιστές που είναι συνδεδεμένοι στο Internet.

File Sharing:
Σε ένα P2P δίκτυο ο κάθε συνδεδεμένος σε αυτό peer μπορεί να κάνει διαθέσιμα για κοινή χρήση κάποια από τα δεδομένα που φιλοξενεί. Έτσι κάθε peer μπορεί ταυτόχρονα να κάνει downloading κάποια αρχεία από κάποιον άλλον peer ή uploading κάποια από τα δικά του αρχεία. Παραδείγματα file sharing εφαρμογών είναι το Napster, ο προγονός όλων των file sharing συστημάτων, το Mule, το Bit Torrent και πολλά αλλά.

Collaboration:
Πολλές εφαρμογές που σχετίζονται με άμεση επικοινωνία ή επαφή μεταξύ χρηστών μπορούν να υλοποιηθούν με το μοντέλο P2P. Τέτοιες εφαρμογές μπορεί να είναι instant messaging, chat, online games και άλλες που μπορούν να

χρησιμοποιηθούν σε επαγγελματικό η οικιακό περιβάλλον. Μια P2P εφαρμογή για instant messaging είναι και το open-source Jabber.

2.2 - Ξ.Κ.Τ.Χ Project

2.2.1 - Σκοπός

Το Guerilla Sharing System αναπτύχθηκε σε θεωρητικό επίπεδο από την Χρυσσάνη Ξηρουχάκη στα πλαίσια του μεταπτυχιακού της με την έγκριση των καθηγητών Μανώλη Κουμπαράκη (επιβλέπων), Πέτρο Τριανταφύλλου (επιβλέπων) και Σταύρο Χριστοδουλάκη. Σκοπός του project αυτού ήταν η προσέγγιση ενός P2P sharing system το οποίο θα διασφάλιζε υψηλή απόδοση μέσω της πλήρους εκμετάλλευσης όλων των διαθέσιμων πόρων των συμμετεχόντων στο δίκτυο, καθώς και της αποδοτικής διαχείρισης του content . Προτάθηκε μια διαφορετική αρχιτεκτονική P2P, και ένα σύνολο πρωτοκόλλων και μηχανισμών για την επίτευξη αυτού του σκοπού [XKTC03] . Το project θα αναφέρεται από εδώ και στο εξής ως Ξ.Κ.Τ.Χ, όνομα προερχόμενο από τα αρχικά των συντελεστών του.

2.2.2 - Εναλλακτική Αρχιτεκτονική για P2P Sharing System

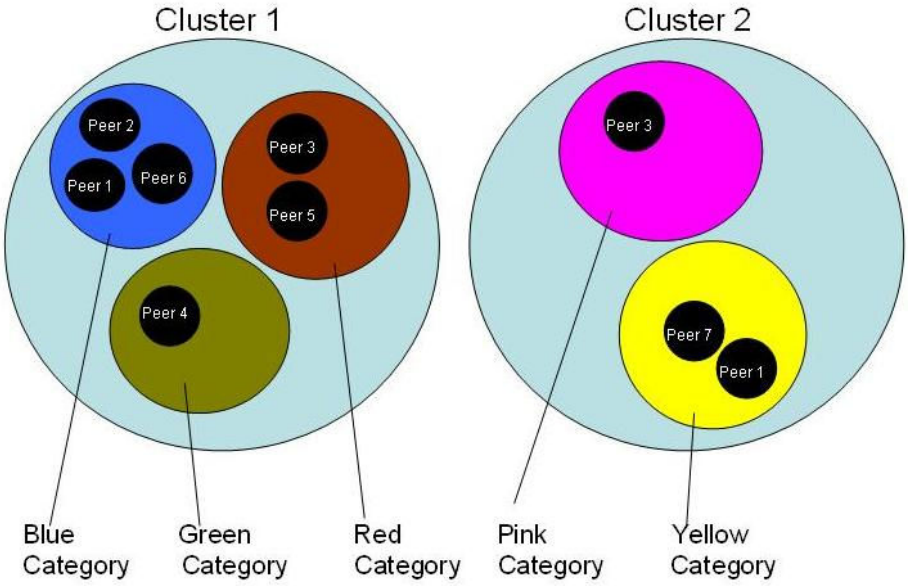
Η εναλλακτική P2P αρχιτεκτονική του Ξ.Κ.Τ.Χ, στην οποία και βασίστηκε η υλοποίηση του Guerilla Sharing System είναι η εξής [XKTC03] :

Οι peers/nodes του συστήματος οργανώνονται λογικά σε ένα σύνολο *C* αποτελούμενο από clusters. Κάθε peer που ανήκει στον ίδιο cluster μπορεί να εξυπηρετεί αιτήσεις που σχετίζονται με documents που συνεισφέρθηκαν από peers του ίδιου cluster, ή να βρει έναν άλλον peer που να μπορεί να το κάνει. Το τελευταίο μπορεί να γίνει αν οι peers αποθηκεύουν metadata τα οποία αντιστοιχίζουν documents με cluster nodes.

Υποθέτουμε ότι όλα τα documents που συνεισφέρονται από τους peers του συστήματος διαχωρίζονται σε ένα σύνολο *S* σημασιολογικών κατηγόριων (document semantic categories) . Έτσι οι clusters των peers αποτελούν αποθηκευτικό χώρο (storage repositories) και ο κάθε cluster μπορεί να αποθηκεύει και να εξυπηρετεί αιτήσεις για documents που ανήκουν σε μια ή

παραπάνω κατηγόριες. Κάθε κατηγόρια μπορεί να ανήκει μόνο σε έναν cluster. Οι peers μπορούν να συνδέονται με clusters σύμφωνα με την κατηγόρια στην οποία ανήκουν τα documents που εκδίδουν. Έτσι, ανάλογα με το πως συνδέονται οι κατηγόριες με τους clusters ένας peer μπορεί να ανήκει σε πάνω από έναν cluster αν συνεισφέρει documents που σχετίζονται με πάνω από μια κατηγόρια.

Το παρακάτω σχήμα αναπαριστά ένα στιγμιότυπο του sharing system για την καλύτερη κατανόηση της αρχιτεκτονικής του.



Σχήμα 4: Οργανωτική Άποψη του Συστήματος

Παρατηρούμε στο σχήμα 4 ότι οι δυο clusters του συστήματος περιέχουν κάποιες σημασιολογικές κατηγόριες. Ο cluster 1 περιέχει την *blue*, την *green* και την *red* category, όπου καμία από αυτές δεν μπορεί να βρίσκεται και στον cluster 2. Αντίστοιχα, ο cluster 2 φιλοξενεί την *pink* και την *yellow* category, οι οποίες δεν μπορούν να υπάρχουν και στον cluster 1 την ίδια στιγμή. Ο κάθε cluster περιέχει peers ανάλογα με το τι είδους αρχεία έχει εκδώσει ο καθένας. Ο peer 3 για παράδειγμα έχει εκδώσει αρχεία που ανήκουν στην *pink* και στην *red* category, οπότε αυτός εμπεριέχεται και στους δυο clusters, ενώ ο peer 7 που έχει εκδώσει αρχεία που ανήκουν μόνο στην *yellow* category φιλοξενείται από τον cluster 2.

2.2.3 - Εξασφάλιση Υψηλής Απόδοσης

Ειπώθηκε, ότι σκοπός του Ξ.Κ.Τ.Χ ήταν η προσέγγιση ενός P2P sharing system το οποίο θα διασφάλιζε υψηλή απόδοση μέσω της πλήρους εκμετάλλευσης όλων

των διαθέσιμων πόρων των συμμετεχόντων στο δίκτυο, καθώς και της αποδοτικής διαχείρισης του content. Για να εξηγηθεί ακριβώς ο τρόπος με τον οποίο εξασφαλίζεται υψηλή απόδοση στο σύστημα πρέπει να δοθεί αρχικά η επεξήγηση του όρου "load" (φορτίο). Ως “load” νοείται ο αριθμός των αιτήσεων που εξυπηρετεί ένας peer του συστήματος .

Για να εξασφαλιστεί στο σύστημα υψηλή απόδοση πρέπει να αποφεύγονται τα bottlenecks και να ισορροπιστεί το φορτίο σε όλους τους peers. Η εξισορρόπηση στο φορτίο επιτυγχάνεται μόνο αν υπάρχει global-load balancing, το οποίο για να κατανοηθεί πρέπει πρώτα από όλα να αναλυθούν δυο άλλες έννοιες, το inter-cluster load balancing και το intra-cluster load balancing που ουσιαστικά αποτελούν συνιστάμενες του.

Λέμε ότι στο σύστημα μας επιτυγχάνεται inter-cluster load balancing αν καταχωρήσουμε document categories σε clusters με τέτοιο τρόπο ώστε να εξασφαλιστεί μια δίκαιη κατανομή των document-category popularities στους clusters των peers. Popularity ενός αρχείου εννοούμε ουσιαστικά την ζήτηση που έχει αυτό το αρχείο, οπότε ως document-category popularity μιας κατηγορίας *S* εννοούμε το συνολικό άθροισμα των popularities των documents που ανήκουν στην *S*.

Ως intra-cluster load balancing εννοούμε ότι για κάθε cluster, όλοι οι peers που ανήκουν σε αυτόν πρέπει να λαμβάνουν αναλογικά περίπου τον ίδιο αριθμό αιτήσεων από τον συνολικό αριθμό αιτήσεων που στοχεύουν αυτόν τον cluster. Λέγοντας "αναλογικά" εννοείται ότι ο αριθμός των αιτήσεων που λαμβάνει ο κάθε peer σε ένα cluster δεν πρέπει να είναι ίσος σε απόλυτους αριθμούς με τις αιτήσεις που λαμβάνει ο κάθε άλλος peer του cluster, αλλά εξαρτάται από κάποιους παράγοντες. Οι παράγοντες αυτοί για κάθε peer είναι η υπολογιστική του ισχύ, το αν συνεισφέρει documents σε πάνω από έναν cluster και η αποθηκευτική του χωρητικότητα. Το να αναγκαστεί ένας αδύναμος υπολογιστής να εξυπηρετήσει ακριβώς τον ίδιο αριθμό αιτήσεων με έναν πιο δυνατό τότε δεν έχουμε δίκαιη κατανομή του φόρτου εργασίας, αν και φαινομενικά αυτό φαίνεται το δικαιότερο εφόσον όλοι θα προσφέρουν ακριβώς τους ίδιους πόρους. Όμως, έτσι το δίκαιο αποτελεί εφαρμογή του ίδιου μέτρου για διαφορετικούς υπολογιστές, για υπολογιστές που δεν είναι όμοιοι μεταξύ τους, και επομένως παύει να είναι δίκαιο.

Αν επιτευχθεί ανεξάρτητα το inter-cluster load balancing και το intra-cluster load balancing τότε επιτυγχάνεται και το global-load balancing, και στην περίπτωση αυτή το μέσο φορτίο όλων των peers που ανήκουν στο σύστημα είναι

ομοιόμορφα κατανεμημένο. Το Ξ.Κ.Τ.Χ έχει πετύχει την δίκαιη κατανομή του φόρτου εργασίας, πάντα αναφερόμενοι στον κόσμο των υπολογιστών, γιατί αυτό το εγχείρημα στον κόσμο των ανθρώπων θα ήταν το λιγότερο ουτοπία.

2.2.4 - Εξασφάλιση inter-cluster load balancing

Για την εξασφάλιση του inter-cluster load balancing προτάθηκε ένας Dynamic Adaptation μηχανισμός. Ο μηχανισμός αυτός αποτελείται, γενικά, από μια cluster αρχιτεκτονική και από αλγόριθμους και πρωτόκολλα απαραίτητα για την δυναμική του προσαρμογή στο μεταβλητό περιβάλλον ενός P2P συστήματος. Ο σκοπός ύπαρξης αυτού του μηχανισμού είναι η διατήρηση του fairness index value κοντά στα επιθυμητά όρια. Το fairness index είναι ένας δείκτης του load balancing. Όσο πιο κοντά είναι στην τιμή "1" τόσο πιο δίκαιη είναι η κατανομή του φόρτου εργασίας, ενώ αν γίνει "0" τότε η κατανομή δεν είναι δίκαιη για κανένα peer του συστήματος.

2.2.5 - Εξασφάλιση intra-cluster load balancing

Για να επιτευχθεί το intra-cluster load balancing θα πρέπει να αποθηκεύονται replicas από documents με μεγάλο popularity (documents, δηλαδή, που έχουν υψηλή ζήτηση) στους peers με τέτοιο τρόπο ώστε το συνολικό popularity των δεδομένων που θα έχουν αποθηκευμένα όλοι οι peers του ίδιου cluster να είναι σχεδόν ίσο.

2.3 - JXTA

2.3.1 - Τι είναι το JXTA

Το JXTA (juxtapose=side by side) είναι μια open source πλατφόρμα, ορισμένη από ένα σύνολο πρωτοκόλλων βασισμένων σε XML που επιτρέπουν σε κάθε συνδεδεμένη συσκευή στο δίκτυο (από κινητά τηλέφωνα και wireless PDAs σε PCs και servers) να επικοινωνεί και να συνεργάζεται σαν κόμβος ενός P2P δικτύου. Τα πρωτόκολλα αυτά, που αποτελούν τον πυρήνα του JXTA, περιγράφουν κάποιες γενικές συμπεριφορές ενός P2P δικτύου. Οι JXTA peers δημιουργούν ένα virtual δίκτυο στο οποίο ο κάθε peer μπορεί να επικοινωνεί με άλλους peers και πόρους ακόμα και όταν αυτοί βρίσκονται πίσω από firewalls

και NATs. Το JXTA μπορεί να γίνει συμβατό με διάφορες γλώσσες προγραμματισμού και αυτές οι υλοποιήσεις λέγονται “bindings”. Το java binding, JXTA2SE, με την οποία υλοποιήθηκε και το Guerilla Sharing System, είναι η πιο ώριμη υλοποίηση έως τώρα, αλλά υπάρχει και η αναπτυσσόμενη Jxta-c, που είναι υλοποίηση του JXTA σε C [JXTA, JXTA05].

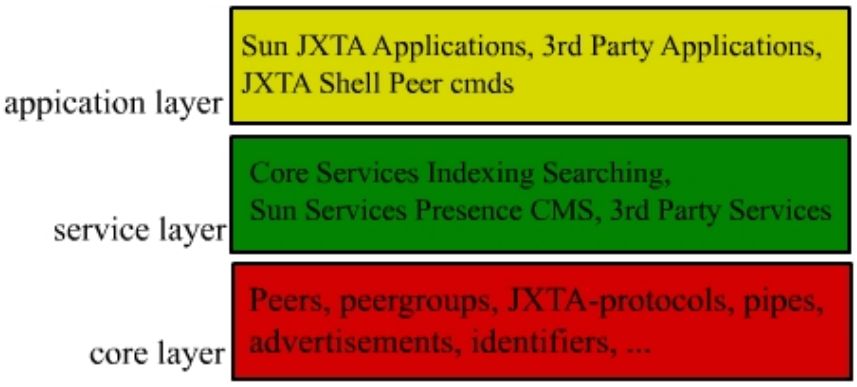
2.3.2 - Αρχιτεκτονική του JXTA

Η αρχιτεκτονική του JXTA είναι χωρισμένη σε τρία layers (σχήμα 5) [JXTA05]. Το χαμηλότερο layer είναι το JXTA platform, πάνω του είναι υλοποιημένο το layer με τα JXTA-services και στο πιο υψηλό επίπεδο είναι το application layer που παρέχει διάφορες P2P εφαρμογές:

Layer 1: Platform (JXTA Core)
Ο πυρήνας του JXTA αποτελείται από στοιχεία που είναι απαραίτητα για κάθε P2P υπηρεσία ή εφαρμογή. Παρέχει το abstract για ένα P2P network, ανεξάρτητα από το τι υπηρεσίες αυτό προσφέρει. Τα στοιχεία αυτά- που θα περιγραφτούν αναλυτικά αργότερα- είναι οι peers, τα peergroups, τα advertisements, οι identifiers, τα JXTA protocols και κάποια στοιχεία σχετικά με την μετάδοση πληροφοριών ή με security και authentication.

Layer 2: Services
Network services που μπορεί να μην είναι απαραίτητες για την λειτουργία ενός P2P δικτύου αλλά είναι συνήθεις ή επιθυμητές σε ένα P2P περιβάλλον. αυτές οι υπηρεσίες μπορούν να υιοθετηθούν σε διάφορες P2P εφαρμογές και είναι χτισμένες πάνω στην JXTA-platform παρέχοντας συγκεκριμένες λειτουργικότητες.

Layer 3: Applications
Έχει υλοποιηθεί πάνω στις λειτουργικότητες που παρέχει το service layer και περιλαμβάνει ολοκληρωμένες εφαρμογές, όπως P2P instant messaging, document και resource sharing, entertainment content management και delivery, P2P e-mail systems, distributed auction systems και πολλά αλλά.



Σχήμα 5: Αρχιτεκτονική του JXTA

2.3.3 - JXTA Components

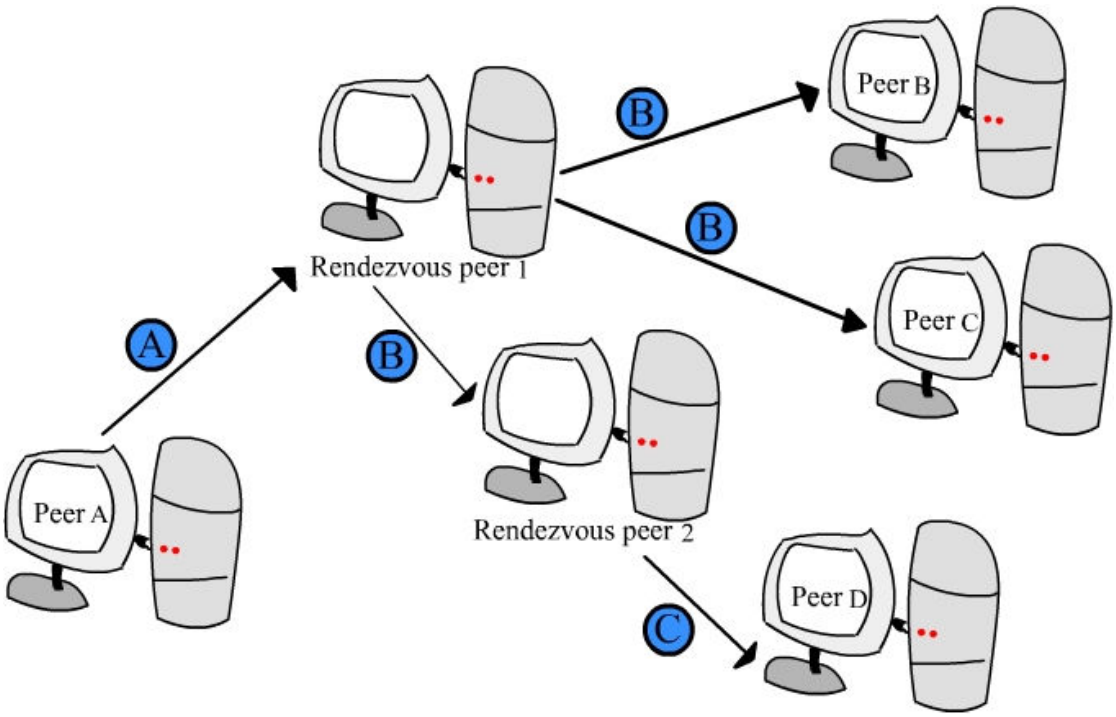
Τα JXTA Components είναι κάποια στοιχεία απαραίτητα για κάθε P2P υπηρεσία ή εφαρμογή. Τα components αυτά είναι τα εξής [BGKS02, GG02, JXTA05, W02]:

Peers:
Είναι η βασική υπολογιστική μονάδα που υλοποιεί κάποια από τα JXTA protocols. Οι peers έχουν ένα high-level χαρακτήρα στο JXTA, νοούνται σαν μια αφηρημένη οντότητα που μπορεί να υλοποιήσει κάποιες P2P υπηρεσίες και εφαρμογές. Έτσι, ο προγραμματιστής δεν εμμένει στην υλοποίηση του peer, αλλά στην υλοποίηση των υπηρεσιών αυτών. Κάθε peer αναγνωρίζεται από ένα μοναδικό ID και εκδίδει ένα ή περισσότερα network interfaces για να χρησιμοποιηθούν με τα πρωτόκολλα του JXTA. Τα interfaces αυτά διαφημίζονται ως peer endpoints και χρησιμοποιούνται για point-to-point επικοινωνία μεταξύ των χρηστών. Οι JXTA peers διαχωρίζονται στις εξής κατηγόριες :

-Minimal Edge Peer:
Απλά στέλνουν και λαμβάνουν messages.

-Full Featured Edge Peer:
Στέλνουν και λαμβάνουν messages, και μπορούν να αποθηκεύσουν advertisements.

-Rendezvous Peer:
Εχουν τα χαρακτηριστικά των Full Featured Edge Peers, αλλά επιπλέον μπορούν να ανακαλύπτουν peers ή peer resources εκ μέρους άλλων peers. Για να γίνει αυτό ο rendezvous peer κρατάει σε μια λίστα τους peers που είναι συνδεδεμένοι με αυτόν.

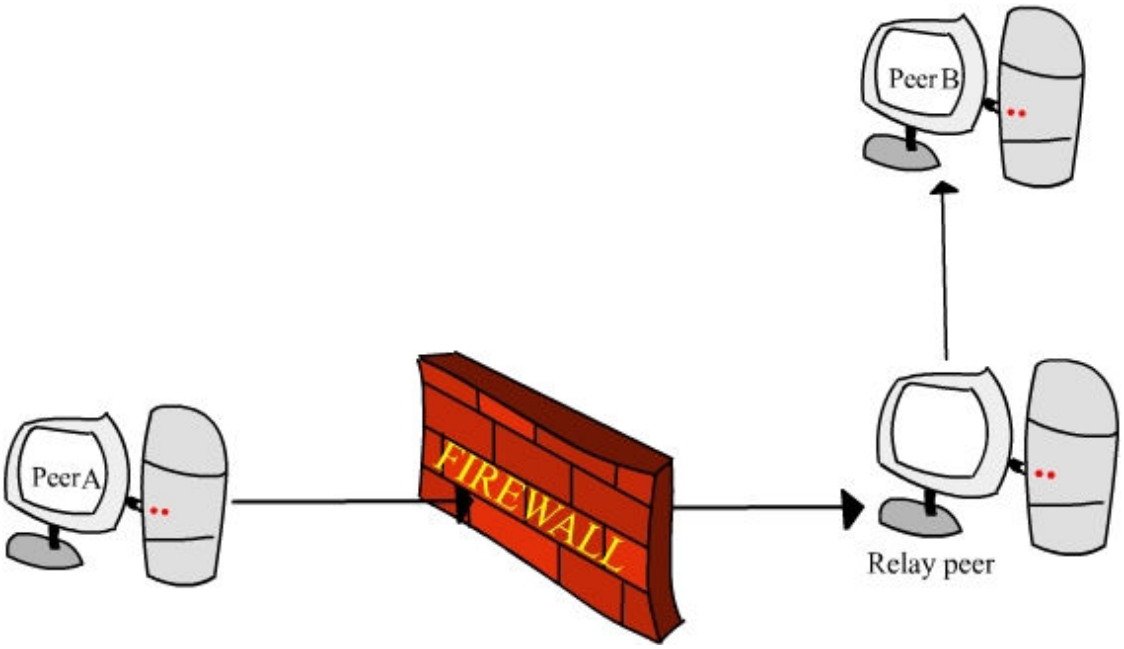


Σχήμα 6: Λειτουργικότητα των Rendezvous Peers

Στο σχήμα 6 βλέπουμε πως ένας peer κάνει propagation ένα message χρησιμοποιώντας έναν rendezvous peer. Στο βήμα Α, ο peer A ζητάει να συνδεθεί με τον rendezvous peer 1. Αφού γίνει με επιτυχία η σύνδεση ο peer A στέλνει στον rendezvous peer 1 το προς μετάδοση message. Ο rendezvous peer 1 το λαμβάνει και βλέπει ποιοι peers (απλοί ή rendezvous) είναι συνδεδεμένοι με αυτόν (peer B, peer C και rendezvous peer 2) και στέλνει σε αυτούς το message (βήμα Β). Στο βήμα C, ο rendezvous peer 2 που έχει λάβει το message από τον rendezvous peer 1 στέλνει εκ μέρους του το message σε όλους τους γνωστούς του peers, δηλαδή στον peer D.

-Relay Peer:
Εχουν τα χαρακτηριστικά των Full Featured Edge Peers, αλλά επιπλέον κρατούν πληροφορίες για το που ακριβώς βρίσκονται απομακρυσμένοι peers, δρομολογούν messages εκ μέρους άλλων peers και έχουν την δυνατότητα να

προωθούν messages διάφορων peers που δεν μπορούν να επικοινωνήσουν άμεσα με άλλους peers (λόγω NAT,firewall κτλ). ένας peer έξω από ένα firewall μπορεί να χρησιμοποιήσει relay peers για να επικοινωνήσει με έναν peer πίσω από firewall και αντίστροφα.



Σχήμα 7: Λειτουργικότητα των Relay Peers

Στο σχήμα 7, οι peer A και B θέλουν να επικοινωνήσουν μεταξύ τους. Όμως ο peer A βρίσκεται πίσω από firewall. Αρχικά συνδέεται με τον relay peer χρησιμοποιώντας ένα πρωτόκολλο που μπορεί να διαπεράσει το firewall (πχ HTTP). Ο relay peer συνδέεται με τον peer B και έτσι ο peer A έχει αποκτήσει μια εικονική σύνδεση με τον B μέσω του relay peer.

Peer Groups:
Σύνολο peers που έχουν προσυμφωνήσει σε ένα κοινό set από services. Το JXTA ορίζει ένα core set από services, αλλά μπορούν να αναπτυχθούν και καινούρια. Το κάθε peergroup, που αναγνωρίζεται από ένα μοναδικό ID μπορεί να έχει δικιά του membership policy. Τα πρωτόκολλα του JXTA περιγράφουν πως οι peers εκδίδουν, ανακαλύπτουν, προσχωρούν ή παρακολουθούν τα peergroups. Τα peergroups σχηματίζουν μια ιεραρχική σχέση τύπου parent-child με πρώτο peergroup το NetPeerGroup το οποίο παρέχει όλα τα JXTA-core services.

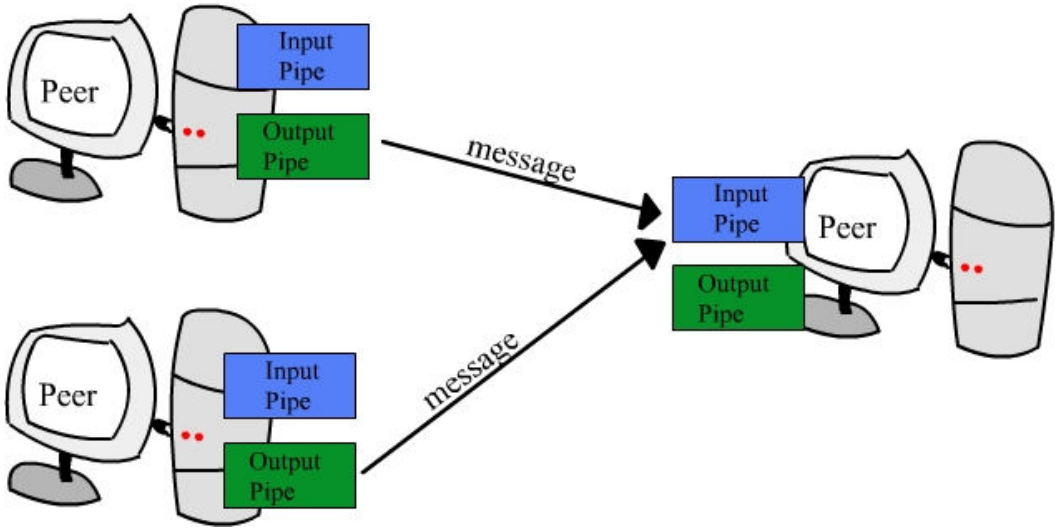
Network Services:
Τα πρωτόκολλα του JXTA διαχωρίζονται σε Peers Services και σε Peer Group

Services:

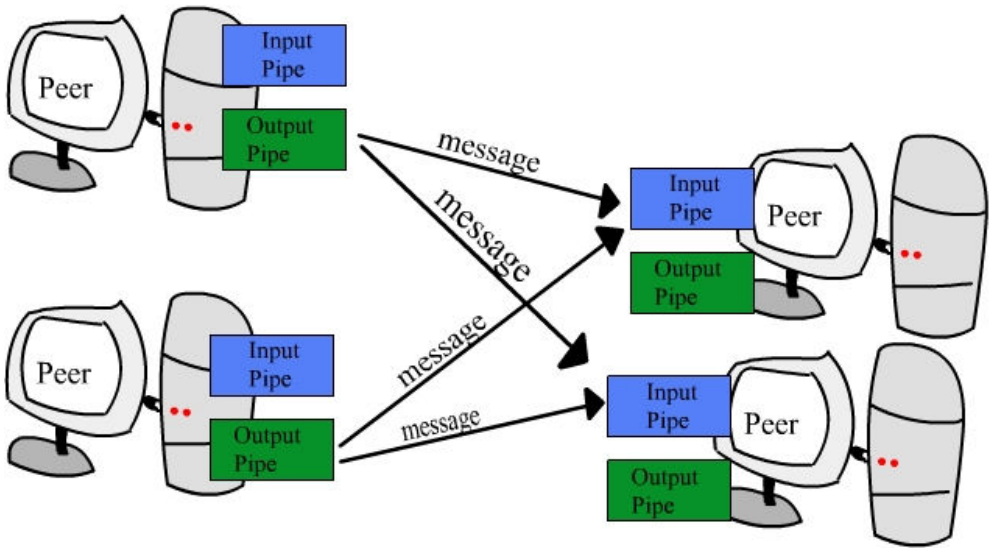
-Peer Services:
Υπηρεσίες που προσφέρονται από ένα peer και είναι διαθέσιμες μόνο όσο ο peer είναι συνδεδεμένος στο δίκτυο. Αν καταρρεύσει ο peer καταρρέει και το service του.

-PeerGroup Services:
Υπηρεσίες που προσφέρονται από ένα peergroup στα μέλη του. Ουσιαστικά παρέχεται από διάφορα μέλη του εκάστοτε peergroup και όσο υπάρχει έστω και ένας peer συνδεδεμένος στο peergroup το service είναι διαθέσιμο σε αυτό. Το JXTA ορίζει ένα core set από peergroup services, αλλά επιπλέον services μπορούν να αναπτυχθούν.

Pipes:
Εικονικά κανάλια επικοινωνίας που συνδέσουν τους peers χωρίς απευθείας φυσική ζεύξη. Τις χρησιμοποιούν οι peers για να στέλνουν μεταξύ τους messages και υποστηρίζουν την μεταφορά οποιουδήποτε object. Τα pipe endpoints αναφέρονται ως input pipe, το endpoint που θα λάβει δεδομένα, και output pipe, το endpoint που θα στείλει δεδομένα. Τα pipe endpoints «δένονται» σε peer endpoints για να μπορούν οι peers να στείλουν ή να λάβουν μηνύματα. Τα pipes διαχωρίζονται σε point-to-point pipes (σχήμα 8), που χρησιμοποιούνται για την μεταφορά δεδομένων από έναν peer σε έναν άλλο, και σε propagate pipes (σχήμα 9) που χρησιμοποιούνται για την μεταφορά δεδομένων από έναν peer σε πολλούς άλλους peers.



Σχήμα 8: Point-to-point pipes



Σχήμα 9: Propagate pipes

Messages:
Objects που στέλνουν μεταξύ τους οι peers. Είναι η βασική μονάδα πληροφορίας

που μεταδίδεται μέσω μιας pipes από ένα endpoint σε ένα άλλο. Τα JXTA messages μπορούν να αναπαρασταθούν ως XML και ως binary έγγραφα.

JXTA Modules:
Abstraction που περιγράφει μια JXTA λειτουργικότητα.

Advertisements:
XML έγγραφα που αναπαριστούν όλους τους πόρους του JXTA network (όπως peers, peergroups, pipes, services). Τα πρωτόκολλα του JXTA χρησιμοποιούν τα advertisements για να περιγράψουν και να εκδώσουν την ύπαρξη των πόρων ενός peer. Οι peers ανακαλύπτουν τους διάφορους πόρους του δικτύου ψάχνοντας για τα αντίστοιχα advertisements που τους περιγράφουν έχοντας την δυνατότητα όταν τα ανακαλύπτουν να τα αποθηκεύουν τοπικά. Έχουν συγκεκριμένο lifetime το οποίο ορίζει την διαθεσιμότητα των resources που περιγράφουν και μπορούν να επανεκδοθούν πριν λήξουν.

IDs:
Όλοι οι πόροι του JXTA (peers, peer groups, pipes και αλλά) αναγνωρίζονται μοναδικά από ένα JXTA ID. Κάθε αναφορά σε κάποιο πόρο του JXTA γίνεται μέσω αυτού του ID.

2.3.4 - JXTA Protocols

Τα JXTA protocols μαζί με τα υπόλοιπα JXTA components αποτελούν τον πυρήνα του JXTA και περιγράφουν κάποιες συμπεριφορές που μπορεί να έχει κάθε P2P δίκτυο ανεξάρτητα από το τι είδους ανάγκες αυτό καλύπτει. Τα πρωτόκολλα αυτά περιγράφουν, γενικά, τον τρόπο με τον οποίο οι peers ανακαλύπτουν ο ένας τον άλλον, αυτό-οργανώνονται σε peergroups, διαφημίζουν και ανακαλύπτουν πόρους του δικτύου, επικοινωνούν μεταξύ τους και παρακολουθούν ο ένας τον άλλον, και όλα αυτά χρησιμοποιώντας τα JXTA messages. Τα πρωτόκολλα αυτά είναι τα εξής [BGKS02, GG02, JXTA05, W02]:

Peer Discovery Protocol – PDP:
Χρησιμοποιείται από τους peers για να διαφημίσουν τα resources τους και για να ανακαλύψουν resources άλλων peers.

Peer Information Protocol – PIP:
Χρησιμοποιείται από τους peers για να αποκομίσουν πληροφορίες για την κατάσταση άλλων peers.

Peer Resolver Protocol – PRP:
Επιτρέπει στους peers να στέλνουν ένα γενικού περιεχομένου query σε έναν ή περισσότερους peers και να λαμβάνουν μια ανταπόκριση (αντίθετα με το PDP και το PIP που στέλνουν μηνύματα συγκεκριμένου περιεχομένου)

Pipe Binding Protocol – PBP:
Χρησιμοποιείται από τους peers για να συνδέσουν ένα pipe μεταξύ ενός ή παραπάνω peer ώστε να μπορούν να ανταλλάξουν μεταξύ τους μηνύματα.

Endpoint Routing Protocol – ERP:
Χρησιμοποιείται από τους peers για να βρουν διαδρομές σε απομακρυσμένους peers. Όταν ένας peer θέλει να στείλει ένα μήνυμα σε έναν άλλο peer, μπορεί να χρησιμοποιήσει το ERP για να βρει μια ακολουθία από relay peers οι οποίοι θα αναλάβουν να στείλουν το μήνυμα στον τελικό προορισμό.

Rendezvous Protocol – RVP:
Είναι υπεύθυνο για την μετάδοση μηνυμάτων μέσα σε ένα peergroup και για τον έλεγχο της ορθής μετάδοσης τους (TTL, loop back detection, κτλ).
Χρησιμοποιείται από το Peer Resolver και το Pipe Binding Protocol για την μετάδοση μηνυμάτων.

Κεφάλαιο 3

ΣΧΕΔΙΑΣΜΟΣ

Ειπώθηκε ότι ο βασικός στόχος της συγκεκριμένης διπλωματικής είναι η υλοποίηση κάποιων πολύπλοκων peer-to-peer μηχανισμών οι οποίοι θα αποτελούν την βάση πάνω στην οποία κάποιος προγραμματιστής θα μπορεί να χτίσει high-level εφαρμογές. Το Guerilla Sharing System, ένα P2P πρόγραμμα ανταλλαγής αρχείων, συντίθεται και αυτό από τέτοιες εφαρμογές, οι οποίες αναπτύχθηκαν στα πλαίσια της διπλωματικής για την επαλήθευση και την αποτίμηση της λειτουργίας των μηχανισμών που αναφέρθηκαν.

Επιπλέον, αναφέρθηκε ότι το Guerilla Sharing System βασίστηκε στο Ξ.Κ.Τ.Χ. Το Ξ.Κ.Τ.Χ προτείνει μια λογική δομή για ένα P2P σύστημα ανταλλαγής αρχείων, η οποία θα διασφάλιζε υψηλή απόδοση στο σύστημα μέσω της πλήρους εκμετάλλευσης όλων των διαθέσιμων πόρων των συμμετεχόντων στο δίκτυο, καθώς και της αποδοτικής διαχείρισης του content.

Το παρών κεφάλαιο ουσιαστικά αναλύει το πως σχεδιάστηκε το Guerilla Sharing System βάσει του Ξ.Κ.Τ.Χ, το τι επιπλέον παραδοχές έπρεπε να παρθούν και τι τροποποιήσεις έπρεπε να γίνουν σε αυτό για να υλοποιηθεί το σύστημα στην πλατφόρμα JXTA.

3.1 - Guerilla Components

Τα Guerilla Components είναι κάποια στοιχεία απαραίτητα για την ορθή λειτουργία του Guerilla Sharing System και σχεδιάστηκαν έτσι ώστε να υλοποιούν κάθε συμπεριφορά του συστήματος όπως αυτή περιγράφεται – είτε επακριβώς, είτε παραλλαγμένα- στο Ξ.Κ.Τ.Χ. Τα components αυτά

είναι τα εξής:

3.1.1 - Guerilla Server

Το P2P σύστημα που προτείνεται στο Ξ.Κ.Τ.Χ αποτελείται από αυτόνομους peers οι οποίοι αυτό-οργανώνονται σε clusters αναλόγως με το σημασιολογικό περιεχόμενο των αρχείων που εκδίδουν για κοινή χρήση. Οι ίδιοι οι peers είναι αυτοί που αποθηκεύουν τα αρχεία προς μοίρασμα και όταν ένας peer αναζητά ένα αρχείο δεν απευθύνεται σε κάποιο κεντρικοποιημένο κατάλογο, σε κάποιο server δηλαδή που φυλάει μια δυναμική βάση δεδομένων για τα αρχεία που έχει ο κάθε ομότιμος, αλλά απευθύνεται σε έναν τυχαίο peer του cluster που φιλοξενεί την σημασιολογική κατηγόρια που ανήκει το ζητούμενο αρχείο. Αν αυτός ο peer δεν έχει αποθηκευμένο το επιθυμητό αρχείο προωθεί το query σε άλλους peers του ίδιου cluster μέχρι αυτό να βρεθεί και να επιστραφεί τελικά η απάντηση στον αρχικό peer. Επομένως, θα ήταν άχρηστο να δημιουργηθεί ένας server με κεντρικοποιημένο κατάλογο, εφόσον η ίδια η οργάνωση των peers σε clusters είναι που βοηθάει τον κάθε peer να βρει αυτό που θέλει.

Όμως, σύμφωνα με το Ξ.Κ.Τ.Χ, ένας peer όταν πρωτοσυνδέεται στο σύστημα πρέπει να βρει έναν ήδη συνδεδεμένο peer ο οποίος θα του δώσει κάποια metadata[2], που έχει αποθηκευμένα στην τοπική του cache, απαραίτητα για την εύρυθμη λειτουργία του νέου peer στο σύστημα.

Πρακτικά, σε ένα P2P δίκτυο ο κάθε peer μπορεί να συνδέεται και να αποσυνδέεται ανά πάσα στιγμή και να μην έχει καν σταθερό IP. Άρα ο peer που θα ήθελε να ενταχθεί στο σύστημα δεν θα είχε στην διάθεση του κάποια σταθερή διεύθυνση και επομένως δεν θα μπορούσε να συνδεθεί στο Guerilla Sharing System. Όποτε, η δημιουργία ενός server με σταθερό και γνωστό IP που απλά αποθηκεύει τις διευθύνσεις των ήδη συνδεδεμένων peer είναι υποχρεωτική. Ο peer που θέλει να κάνει join στο σύστημα συνδέεται με τον server, ο οποίος του δίνει την διεύθυνση ενός τυχαίου peer για να συνδεθεί με αυτόν και να ανακτήσει τα επιθυμητά metadata.

Ο server του Guerilla Sharing System δεν είναι απαραίτητος μόνο στο να φυλάει διευθύνσεις peers, αλλά και στο να δημιουργεί και να αποθηκεύει JXTA advertisements που αναπαριστούν κάποιους πόρους του δικτύου. Τα

[2] Τα metadata αυτά είναι tables που περιέχουν αντιστοιχίσεις semantic categories με clusters και clusters με peers. Δηλαδή, υποδεικνύουν στον νέο peer τι semantic categories περιέχει ο κάθε cluster και ποιους peers περιέχει ο κάθε cluster εκείνη την στιγμή.

advertisements αυτά είναι peer group advertisements και pipe advertisements (αναπαριστούν όλα τα peer groups και όλα τα pipes του Guerilla). Τα peer group advertisements ανακτούνται από την cache του server από κάποιον απομακρυσμένο peer όταν αυτός τα χρειαστεί και τα pipe advertisements αντιγράφονται στην cache του κάθε peer (δηλαδή βρίσκονται εξαρχής στο software του peer του Guerilla Sharing System) για να ανακτηθούν τοπικά από έναν peer όταν αυτός τα χρειαστεί Για παράδειγμα, το advertisement που αναπαριστά το peer group Guerilla, στο οποίο βρίσκονται όλοι οι peers που έχουν συνδεθεί στο Guerilla Sharing System, γίνεται διαθέσιμο από τον server σε έναν απομακρυσμένο peer όταν αυτός πρωτοσυνδέεται στο δίκτυο. Χωρίς αυτό κανένας peer δεν μπορεί να συνδεθεί στο σύστημα. Το ότι τα advertisements φυλάσσονται στον server δεν σημαίνει ότι ένας peer τα ανακτά μόνο από αυτόν. Από την στιγμή που ένας peer θα βρει κάποια advertisements και θα τα αποθηκεύσει στην τοπική του cache, τότε ένας άλλος ομότιμος μπορεί να τα ανακτήσει από αυτόν και όχι από τον server.

Συνοπτικά: *Η χρησιμότητα του server έγκειται στους εξής λόγους:*

1) *Αποθηκεύει τα ID όλων των peers του δικτύου. Κάποιος peer που θέλει να συνδεθεί αρχικά στο δίκτυο, ζητάει το ID ενός άγνωστου για αυτόν peer από τον server, ώστε να ανακτήσει από αυτόν τα απαραίτητα metadata.*
2) *Δημιουργεί και αποθηκεύει όλα τα peer group και pipe advertisements. Τα peer group advertisements παρέχονται από τον server στους απομακρυσμένους peers όταν αυτοί τα ζητήσουν και τα pipe advertisements βρίσκονται εξαρχής στην local cache των peers για να τα ανακτήσουν τοπικά. Τα peer group advertisements ανακτώνται και από τις τοπικές caches των peers που τα έχουν ήδη ανακαλύψει.*

Εναλλακτικές ιδέες: *Όσον αφορά τα peer group advertisements:*

-Θα μπορούσαν να δημιουργούνται στους peers και όχι στον server ώστε η ανάκτηση τους να γίνεται τοπικά και όχι απομακρυσμένα, εφόσον θα ήταν αποθηκευμένα εξαρχής στους peers. Κάθε peer θα δημιουργούσε είτε όλα μαζί τα advertisements κατά το bootstrapping, είτε κάθε advertisement την ώρα που το χρειάζονταν.

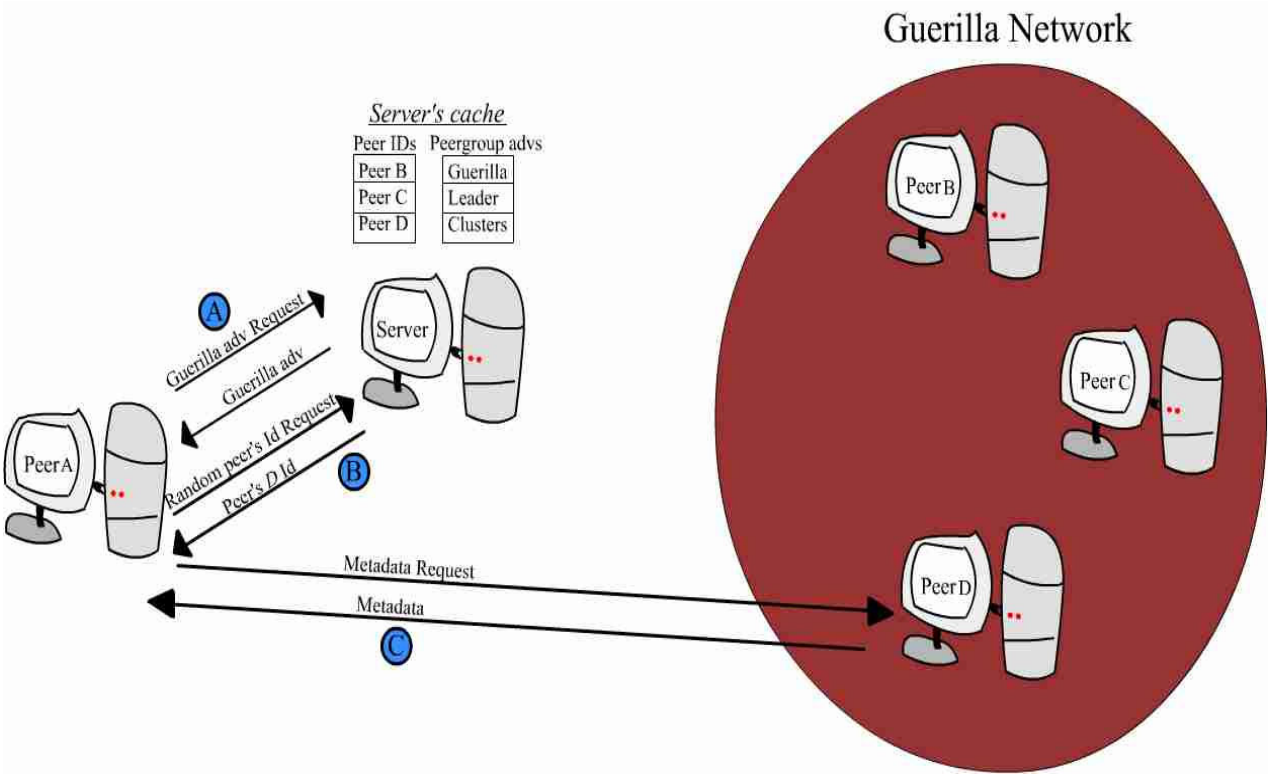
-Θα μπορούσαν να δημιουργούνται στον server, όπως και γίνεται, αλλά να αντιγραφοταν στην cache του peer (όπως γίνεται με τα pipe advertisements) ώστε η ανάκτηση τους να γίνεται τοπικά και όχι απομακρυσμένα.

Όσον αφορά τα pipe advertisements:

-Θα μπορούσαν να δημιουργούνται στους peers και όχι στον server.

-Θα μπορούσαν να δημιουργούνται στον server, όπως και γίνεται, αλλά να

μην αντιγραφοταν στην cache του peer ώστε η ανάκτηση τους να γίνεται απομακρυσμένα (όπως και με τα peer group advertisements).



Σχήμα 10: Server Functionality

Στο σχήμα 10, ο peer *A* θέλει συνδεθεί στο Guerilla network. Στο βήμα Α, ζητάει από τον γνωστό server το advertisement του Guerilla peer group και ο server του το επιστρέφει. Πλέον, ο *A* μπορεί να χρησιμοποιήσει τις υπηρεσίες του Guerilla. Για να μπορέσει να ενταχθεί στο δίκτυο, ο *A* ζητάει και ανακτά από τον server το ID ενός peer που είναι συνδεδεμένος στο δίκτυο (βήμα Β) . Στο τελευταίο βήμα, ο *A* γνωρίζοντας το μοναδικό ID του peer *D,* συνδέεται μαζί του για να ζητήσει τα metadata του. Ο *D* λαμβάνει την αίτηση και επιστρέφει σε ένα μήνυμα τους χρήσιμους πίνακες.

3.1.2 – Guerilla Peer Groups

Τα peer groups του Guerilla Sharing System σχεδιάστηκαν βάσει της cluster αρχιτεκτονικής που προτείνεται στο Ξ.Κ.Τ.Χ και είναι τα εξής:

- Guerilla PeerGroup:

Κάθε peer που τρέχει την εφαρμογή Guerilla Sharing System ανήκει στο Guerilla

PeerGroup. Ένας peer ανακτά το advertisement του Guerilla PeerGroup από τον server κατά το bootstrapping του. Αν ένας peer δεν μπορέσει να συνδεθεί στον server ή δεν μπορέσει να ανακτήσει το Guerilla advertisement η εφαρμογή τερματίζεται. Το Guerilla PeerGroup είναι γόνος του NetPeerGroup .

- Clusters PeerGroups:

Εφόσον το σύστημα είναι χωρισμένο σε clusters που περιέχουν σημασιολογικές κατηγόριες είναι βολικό να ταυτιστούν αυτοί οι clusters με jxta peer groups. Έτσι όλοι οι peers που ανήκουν σε έναν cluster, έστω τον cluster "i", ανήκουν ουσιαστικά στο peer group με όνομα "i". Η ταυτοποίηση αυτή βοηθάει γιατί υπάρχουν υπηρεσίες, όπως θα δούμε παρακάτω, που απευθύνονται σε peers του ίδιου cluster. εφόσον τα peer groups είναι σύνολο peers που έχουν προσυμφωνήσει σε ένα κοινό σύνολο από services είναι λογικό να θεωρούμε τον κάθε cluster ως ξεχωριστό peer group. Ο κάθε peer ανακτά το advertisement ενός cluster peer group είτε από τον server, είτε από κάποιον peer που είναι μέλος του συγκεκριμένου cluster όταν το χρειαστεί. Για παράδειγμα, όταν ένας peer θέλει να εκδώσει ένα αρχείο που ανήκει σε μια κατηγόρια, η οποία φιλοξενείται από τον cluster *3*, τότε ψάχνει στην cache κάποιου peer ή του server που έχει αποθηκευμένο το peer group advertisement του cluster *3*. Όταν το ανακτήσει μπορεί τότε και αυτός να γίνει μέλος του cluster και να γνωστοποιήσει σε όλα τα μέλη του ότι είναι και αυτός ενταγμένος στην κοινότητα τους. όλα τα cluster peer groups έχουν γονέα το Guerilla peer group.

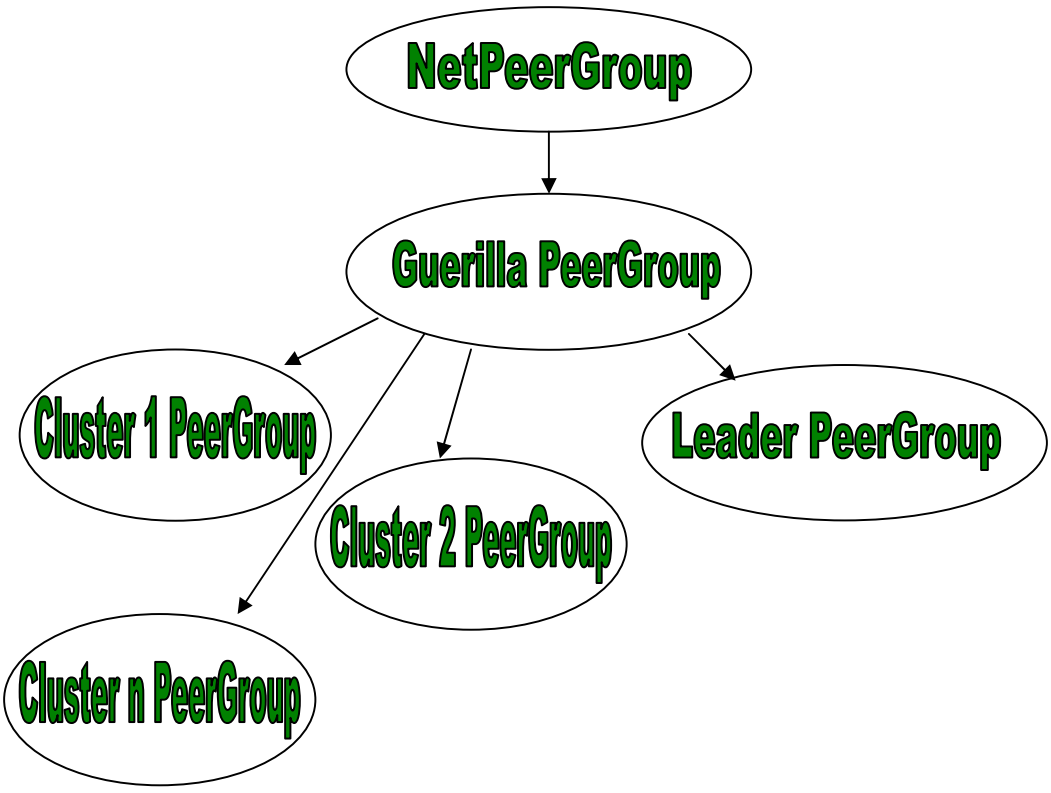
- Leader PeerGroup:

Στο Leader peer group ανήκουν όλοι οι leaders του συστήματος. Οι leaders είναι super-peers, δηλαδή peers που εξαιτίας κάποιων έξτρα δυνατοτήτων αποθηκεύουν κάποια επιπλέον metadata. Επειδή υπάρχει επικοινωνία μεταξύ των leaders ανεξάρτητη από τους υπόλοιπους peers τοποθετήθηκαν σε ένα ξεχωριστό peer group για να γίνεται αυτή η επικοινωνία πιο εύκολη. Ένας peer όταν ανακηρυχθεί leader ανακτά το advertisement του leader peer group είτε από τον server, είτε από κάποιον peer που είναι ήδη leader και έχει τοπικά αποθηκευμένο το advertisement αυτό. Το Leader PeerGroup έχει γονέα το Guerilla PeerGroup. Οι leaders αναλύονται εκτενέστερα στα επόμενα κεφάλαια .

Παρακάτω, βλέπουμε έναν πίνακα με μια συνοπτική περιγραφή των peer groups του Guerilla, καθώς και ένα σχεδιάγραμμα για την ιεραρχία τους:

Όνομα Peer Group:	*Ποιοι peers ανήκουν:*	*Ανάκτηση peergroup adv:*
Guerilla PeerGroup	Όσοι τρέχουν το Guerilla Sharing System.	Από τον server κατά το bootstrapping ενός peer.
i PeerGroup (όπου *i* ο αριθμός του cluster που ταυτίζεται με αυτό το peer group)	Όσοι peers έχουν εκδώσει αρχείο που ανήκει σε κατηγόρια η οποία φιλοξενείται από τον cluster *i*.	Από τον server ή από έναν peer που ανήκει στον cluster *i*.
Leader PeerGroup	Όλοι οι leaders.	Από τον server ή από έναν leader.

Πίνακας 1: PeerGroups του Guerilla Sharing System



Σχήμα 11: Ιεραρχία των PeerGroups

3.1.3 – Guerilla Peer

Οι peers του Guerilla Sharing System εντάσσονται στο σύστημα σαν edge peers. Όταν, όμως, γίνουν μέλη ενός peer group, δηλαδή είτε κάποιου Cluster PeerGroup, είτε του Leader PeerGroup, μετατρέπονται σε rendezvous peers για να προωθούν μηνύματα που αφορούν το εκάστοτε peer group. Όλοι οι peers αποθηκεύουν τις ίδιες δομές δεδομένων, εκτός από τους leaders που εξαιτίας της έξτρα υπολογιστικής τους ισχύς κρατάνε επιπλέον metadata χρήσιμα για την δυναμική προσαρμογή του συστήματος σε αλλοιώσεις του inter-cluster load balancing. Οπότε, το σύστημα αποτελείται από απλούς peers, που είναι edge ή rendezvous, και από super-peers leaders, που είναι αναγκαστικά rendezvous εφόσον ανήκουν στο Leader PeerGroup.

3.1.4 - Guerilla Protocols

Η λειτουργία του peer εξασφαλίζεται από μια σειρά πρωτοκόλλων που το καθένα από αυτά υλοποιεί κάποια services. Τα services αυτά εγγυούνται το bootstrapping του peer και την περαιτέρω κανονική του λειτουργία.

Guerilla Protocols : Set of Services → Peer bootstrapping + κανονική λειτουργία

Το κάθε πρωτόκολλο, δηλαδή, είναι ένα σύνολο από services που υλοποιούν μια συγκεκριμένη συμπεριφορά ενός peer. Τα πρωτόκολλα αυτά είναι τα εξής: *Dynamic Adaptation Protocol, Exhange Data Protocol, Join Protocol, Propagate Protocol, Publish Protocol* και *Query Protocol.*

- Dynamic Adaptation Protocol:

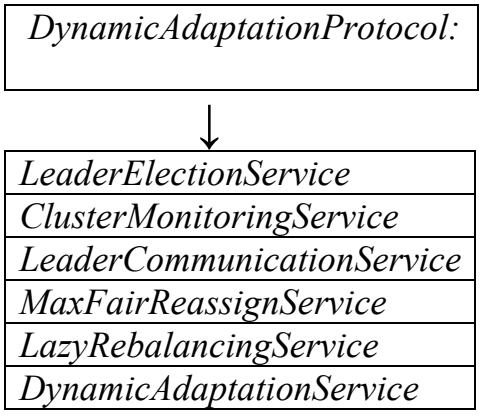
Προαναφέρθηκε, ότι στο σύστημα ανταλλαγής αρχείων που προτείνεται στο Ξ.Κ.Τ.Χ επιτυγχάνεται inter-cluster load balancing αν καταχωρούνται οι σημασιολογικές κατηγόριες σε clusters με τέτοιο τρόπο ώστε να εξασφαλιστεί μια δίκαιη κατανομή των document-category popularities στους clusters των peers. Όμως, τα popularities των σημασιολογικών κατηγοριών σε ένα δυναμικό P2P σύστημα μπορεί να μεταβληθούν με αποτέλεσμα να επηρεαστεί η ισορροπία που είχε επιτευχθεί.

Τα popularities μπορούν να μεταβληθούν για τρεις λόγους:
1) Να μεταβληθεί η ζήτηση των χρηστών πάνω σε κάποια αρχεία.
2) Να μεταβληθεί ο πληθυσμός των documents:
Συνήθως η προσθήκη ελάχιστων documents από χρήστες σε διάφορους clusters δεν επηρεάζει το inter-cluster load balancing. Η προσθήκη όμως πολλών documents σε μια κατηγόρια επηρεάζει αρκετά τα popularities των σημασιολογικών κατηγόριων ώστε να διαταραχτεί το inter-cluster load balancing.
3) Αλλαγές στον πληθυσμό των peers:
Με την προσθήκη ή την αποχώρηση peers μπορεί να επηρεαστεί το inter-cluster load balancing που έχει επιτευχθεί γιατί έχουμε αλλαγές στον αριθμό των documents (εφόσον προσθέτονται ή αφαιρούνται documents).

Οπότε, ακόμα και αν αρχικά τοποθετηθούν οι σημασιολογικές κατηγόριες στους κατάλληλους clusters, έτσι ώστε να εξασφαλιστεί μια δίκαιη κατανομή των document-category popularities στους clusters των peers, μπορεί σε κάποια στιγμή, όταν το popularity μιας κατηγορίας θα έχει μεταβληθεί σε σημαντικό βαθμό, να μην ισχύει αυτή η δίκαιη κατανομή, άρα ούτε το inter-cluster load balancing.

Για την διατήρηση του inter-cluster load balancing σε ένα δυναμικό περιβάλλον προτείνεται στο Ξ.Κ.Τ.Χ ένας Dynamic Adaptation μηχανισμός, ο οποίος αποτελείται από αλγόριθμους και πρωτόκολλα τα οποία διατηρούν το fairness index value κοντά στα επιθυμητά όρια. Ο Dynamic Adaptation μηχανισμός εκτελείται σε μια σειρά από βήματα, όπου το κάθε βήμα υλοποιείται από ένα service του *DynamicAdaptationProtocol*.

Συνοπτικά, το *DynamicAdaptationProtocol* αποτελείται από τα εξής services:



Πριν ξεκινήσει η εκτέλεση του Dynamic Adaptation μηχανισμού πρέπει στον κάθε cluster να έχει εκλεχθεί ένας αρχηγός. αυτό υλοποιείται με το service *LeaderElectionService* του *DynamicAdaptationProtocol*:

Cluster Leader Election
Υλοποιείται από: LeaderElectionService του DynamicAdaptationProtocol
Περιγραφή: Ορίζεται ανά περιόδους ένας peer ως leader σε κάθε cluster. Για να γίνει αυτό πρέπει οι peers να ενημερώνουν περιοδικά τους γείτονες του σε έναν cluster για τις συνολικές τους δυνατότητες και να προωθούν τις σχετικές πληροφορίες που λαμβάνουν από άλλους. Έτσι κάθε peer ενός cluster γνωρίζει για τις ικανότητες των άλλων peers του cluster αυτού. Ο πιο ισχυρός peer του cluster αναγνωρίζεται από τους υπόλοιπους ως leader. Σε περίπτωση που ο leader αποχωρήσει από το σύστημα εκλέγεται καινούριος αρχηγός.

Βήματα Dynamic Adaptation μηχανισμού και αντίστοιχα services:

1ο Βήμα:

Per Cluster Monitoring
Υλοποιείται από: ClusterMonitoringService του DynamicAdaptationProtocol
Περιγραφή: Οι leaders ζητούν από τους peers των clusters που είναι αρχηγοί τα load-figures τους. Έτσι, κάθε leader έχει ένα σύνολο με τα popularity των κατηγόριων που φιλοξενεί ο cluster στον οποίο είναι αρχηγός, με τα οποία υπολογίζει το normalized popularity του cluster αυτού. Το normalized popularity του κάθε cluster χρειάζεται αργότερα για τον υπολογισμό του fairness index.

2ο Βήμα:

Leader Communication Protocol
Υλοποιείται από: LeaderCommunicationService του DynamicAdaptationProtocol
Περιγραφή: Όλοι οι leaders επικοινωνούν μεταξύ τους ώστε να ανταλλάξουν τις σχετικές με τα popularity πληροφορίες που αποκόμισαν στο προηγούμενο βήμα. Τελικά, ο κάθε leader γνωρίζει το load distribution και το normalized popularity του κάθε cluster και της κάθε κατηγορίας.

3ο Βήμα:

Evaluation of fairness
Υλοποιείται από: MaxFairReassignService του DynamicAdaptationProtocol
Περιγραφή: Ο leader με το μεγαλύτερο normalized popularity ορίζεται ως chosen leader και επιλέγεται ώστε να υπολογίσει το fairness index. Το fairness index εξαρτάται από το normalized popularity του κάθε cluster που το ξέρει ήδη από το προηγούμενο βήμα. Οι τιμές που μπορεί να πάρει το fairness index είναι μεταξύ 0 και 1 και όσο πιο πολύ πλησιάζει στο 1 τόσο πιο δίκαιο γίνεται το load distribution. Αν η τιμή του είναι πάνω από ένα προεπιλεγμένο threshold τότε δεν χρειάζεται να γίνει τίποτα παραπάνω μέχρι την επόμενη περίοδο. Όμως, αν είναι κάτω από το threshold ο επιλεγμένος leader εκτελεί το επόμενο βήμα.

4ο Βήμα:

Load Rebalancing among clusters.
Υλοποιείται από: MaxFairReassignService του DynamicAdaptationProtocol
Περιγραφή: Ο επιλεγμένος leader τρέχει τον greedy αλγόριθμο MaxFair, ο οποίος προσπαθεί να πετύχει υψηλή τιμή στον fairness index ανακατατάσσοντας τις κατηγόριες στους clusters. Ουσιαστικά, υπολογίζει την τιμή fairness index που θα προέκυπτε κάθε φορά αν μεταφερόταν μια σημασιολογική κατηγόρια του cluster με το μεγαλύτερο normalized popularity c_i σε έναν cluster εκτός του c_i. όταν καλυφθούν όλοι οι πιθανοί συνδυασμοί η μετακίνηση που έφερε την μεγαλύτερη τιμή στο fairness index επιλέγεται να γίνει και πραγματικά.

5ο Βήμα:

Lazy Rebalancing Protocol
Υλοποιείται από: LazyRebalancingService του DynamicAdaptationProtocol
Περιγραφή: Αναλαμβάνει να μεταφέρει στον destination cluster τα documents που είναι αποθηκευμένα στον source cluster. Αν η τιμή του fairness index εξακολουθεί να είναι κάτω από το προεπιλεγμένο threshold επαναλαμβάνουμε το 4ο βήμα.

Όλα αυτά τα βήματα ελέγχονται και εκτελούνται από ένα service του DynamicAdaptationProtocol, το *DynamicAdaptationService*. Το ποτέ πρέπει να εκτελεστεί το κάθε service και από ποιον peer, αποφασίζεται από το *DynamicAdaptationService*. Το service αυτό είναι ουσιαστικά ο scheduler του

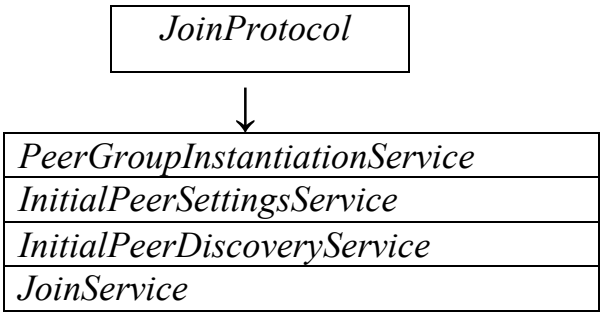
Dynamic Adaptation μηχανισμού.

- Join Protocol:

Το *JoinProtocol* αναλαμβάνει την αρχική ένταξη ενός peer στο Guerilla peer group. Είναι, δηλαδή, υπεύθυνο για το bootstrapping ενός peer.

Θεωρητικά, ο κάθε peer που θέλει να συνδεθεί στο σύστημα πρέπει να βρει έναν άλλον ήδη συνδεδεμένο peer για να του ζητήσει κάποια απαραίτητα metadata (βλ. υποσημείωση 2). Όταν τον βρει και ανακτήσει αυτές τις πληροφορίες ο peer είναι πια έτοιμος να εκδώσει ή να ψάξει για κάποια αρχεία.

Πρακτικά, ο προς ένταξη peer αναζητάει τον γνωστό και σταθερό guerilla server, εφόσον δεν μπορεί να ξέρει για κανέναν άλλον peer του δικτύου. Όταν τον βρει και συνδεθεί με αυτόν, ο server αναλαμβάνει να του επιστρέψει το ID ενός τυχαίου peer, εφόσον η δουλεία του είναι να κρατάει τις διευθύνσεις των συνδεδεμένων ομότιμων στο σύστημα. τότε, ο προς ένταξη κόμβος ζητάει από τον peer τα metadata του (βλέπε σχήμα 10). Αν σε ένα δεδομένο χρόνο ο peer δεν πάρει κάποια διεύθυνση από τον server θεωρεί ότι είναι το μοναδικό μέλος στο σύστημα και εντάσσεται σε αυτό χωρίς να ανακτήσει τα metadata. Αν βρεθεί ένας αρχικός peer από τον οποίο ζητήσει τα metadata, αλλά για κάποιο λόγο δεν τα παραλάβει, ζητάει από τον server την διεύθυνση ενός αλλού τυχαίου peer και ξαναπροσπαθεί με αυτόν. Όλη αυτή η διαδικασία (peer bootstrapping) αναλαμβάνεται από τα services του *JoinProtocol*. Συνοπτικά, αυτά είναι τα εξής:



Όλο το bootstrapping ελέγχεται ουσιαστικά από το *JoinService.* Το *JoinService* είναι αυτό που καλεί τα services του *JoinProtocol* με μια διαδοχική σειρά και αποφασίζει ποτέ ο peer είναι έτοιμος να χρησιμοποιήσει τις υπηρεσίες του Guerilla Sharing System. Οι διεργασίες που πρέπει να γίνουν από ένα peer κατά το bootstrapping περιγράφονται αμέσως παρακάτω.

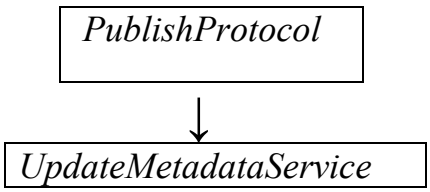
Αρχικά, ο peer πρέπει να βρει το advertisement του Guerilla peer group για να συνδεθεί σε αυτό. Στο Guerilla peer group είναι ενταγμένοι όσοι peers χρησιμοποιούν το Guerilla Sharing System και όλοι αυτοί οι peers μπορούν να βρίσκουν ο ένας τον άλλον και να χρησιμοποιούν τις υπηρεσίες που προσφέρει η εφαρμογή. Το advertisement του Guerilla peer group βρίσκεται στην τοπική cache του server, οπότε ο κάθε peer που θέλει να συνδεθεί στο Guerilla peer group ζητάει και ανακτά το advertisement από αυτόν (βλέπε σχήμα 10). Όλη αυτή η διαδικασία υλοποιείται από το *PeerGroupInstantiationService* του Join Protocol.

Αφού συνδεθεί στο Guerilla peer group, ο νέος peer αρχικοποιεί κάποια settings του με την *InitialPeerSettingsService.* Τα settings αυτά είναι το όνομα του peer, το μοναδικό ID του , το path ενός dataset που περιέχει αντιστοιχίες ταινιών με σημασιολογικές κατηγόριες και popularities, τον συνολικό αριθμό των clusters του Guerilla, τον αριθμό των υπολογιστικών μονάδων του, που χρησιμεύει κατά την εκλογή του leader, και κάποιοι πίνακες και arrays που περιέχουν χρήσιμα δεδομένα και θα αναλυθούν παρακάτω.

Μετά, μέσω της *InitialPeerDiscoveryService*, ζητάει το advertisement ενός τυχαίου peer που ο server έχει αποθηκευμένο στην cache του, οπότε είναι ήδη συνδεδεμένος στο δίκτυο (βλέπε σχήμα 10). Από το advertisement εξάγει το ID του peer, το οποίο του επιτρέπει να συνδεθεί με αυτόν και να του ζητήσει τα metadata του. Τα metadata αυτά είναι οι πίνακες DCRT και NRT που αντιστοιχίζουν categories με clusters και clusters με peers αντίστοιχα.

- Publish Protocol:

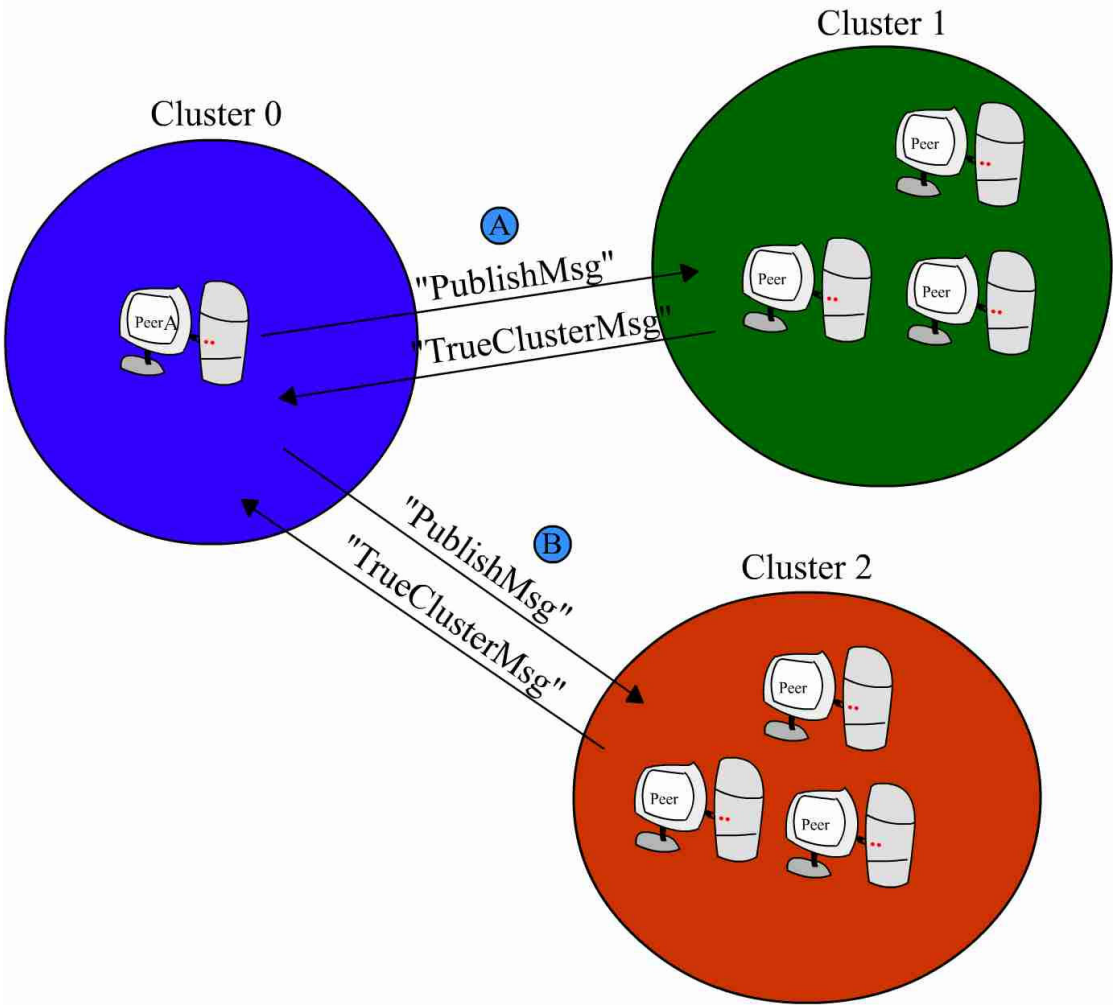
Όταν ένας peer ολοκληρώσει το bootstrapping του, τότε ο χρήστης έχει την δυνατότητα να εκδώσει μια εικονική ταινία. Για να την εκδώσει πρέπει να γίνει μέλος του cluster -αν δεν έχει ήδη γίνει- που φιλοξενεί την πραγματική σημασιολογική κατηγόρια στην οποία ανήκει η ταινία. Οπότε, ο peer πρέπει να ενημερώσει: i) Τα metadata όλων των peers αυτού του cluster ii) Τα δικά του metadata ώστε να ανταποκρίνονται στην αλλαγή. Για όλη αυτή την λειτουργικότητα που κάνει εφικτή την έκδοση αρχείων από χρήστες του Guerilla είναι υπεύθυνα τα services του *PublishProtocol*:



PublishService

Καταρχήν, το *UpdateMetadataService* είναι υπεύθυνο για την ενημέρωση των metadata κατά την έκδοση του νέου document. Αν ο peer κάνει publish σε έναν cluster στον οποίο έχει δημοσιεύσει ξανά documents κάποια στιγμή στο παρελθόν, ενημερώνει μόνο τα δικά του metadata. Αντιθέτως, αν θέλει να εκδώσει ένα document που ανήκει σε μια κατηγόρια *s*, η οποία σύμφωνα με τα δικά του metadata φιλοξενείται από τον cluster *c* στον οποίο δεν έχει εκδώσει ξανά, τότε μεταδίδει ένα message ειδικού τύπου σε όλους τους peers που ανήκουν στον *c*. Οι peers λαμβάνουν το μήνυμα, συμβουλεύονται τον DCRT τους (table που περιέχει αντιστοιχήσεις document categories σε clusters), εξάγουν τον cluster *norm_c* που φιλοξενεί την *s* και ανταποκρίνονται με ένα message στο οποίο εμπεριέχεται ο *norm_c*. Αν ο *norm_c* είναι ίδιος με τον *c*, τότε οι peers του *c* ενημερώνουν τα metadata τους για την ένταξη του νέου peer στον cluster τους, ο peer που έκανε το publish γίνεται rendezvous για τον *c* και κάνει update τα δικά του metadata. Αν ο *norm_c* είναι διαφορετικός από τον *c* (επειδή για κάποιους λόγους ο peer δεν έχει αναβαθμίσει τα metadata του και περιέχει εσφαλμένη αντιστοίχηση σημασιολογικής κατηγορίας σε cluster), τότε όλο το service ξανατρέχει από την αρχή, αλλά αυτή την φορά για τον *norm_c* και όχι για τον *c,* μέχρι να βρεθεί ο cluster που φιλοξενεί την *s*. Ο peer που κάνει το publish σε ένα cluster γίνεται rendezvous σε αυτόν για να μπορεί μελλοντικά να προωθεί messages που αφορούν αυτόν τον cluster (πχ *"PublishMsg"*).

Το *UpdateMetadataService* διαχειρίζεται από το *PublishService*, το οποίο με το κατάλληλο utility διαβάζει το document που θέλει να κάνει publish ο χρήστης, εξάγει τις σημασιολογικές κατηγόριες που ανήκει αυτό το document χρησιμοποιώντας ένα συγκεκριμένο dataset και βλέπει, ελέγχοντας τα metadata του, αν ο peer έχει ξανακάνει ή όχι publish σε αυτές τις σημασιολογικές κατηγόριες. Έτσι, το *UpdateMetadataService* ενημερώνεται από το *PublishService* όσον αφορά τις σημασιολογικές κατηγόριες και τους clusters.



Peer's A DCRT:

"Action"	0
"Sci-fi"	1
"Comedy"	2

DCRT of cluster's 1 peers:

"Action"	0
"Sci-fi"	2
"Comedy"	2

DCRT of cluster's 2 peers:

"Action"	0
"Sci-fi"	2
"Comedy"	2

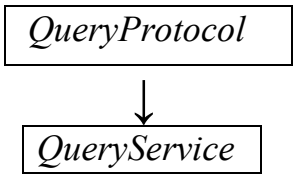
Σχήμα 12: Έκδοση Αρχείων

Στο παραπάνω σχήμα παρατηρούμε τα βήματα που χρειάζεται ο peer A για να

εκδώσει ένα εικονικό αρχείο που ανήκει στην κατηγόρια "sci-fi". Σύμφωνα με τον DCRT του, η κατηγόρια "sci-fi" ανήκει στον cluster 1, στον οποίο υποθέτουμε ότι δεν έχει εκδώσει ξανά. Ο peer A στέλνει ένα "PublishMsg" σε όλους τους peers του cluster 1 (βήμα Α). Τα μέλη του cluster λαμβάνουν το μήνυμα και βλέπουν στον DCRT τους αν ο Α εκδίδει σε σωστό cluster. Σύμφωνα, με τα δικά τους metadata, ο Α θα έπρεπε να εκδίδει στον cluster 2 και όχι στον 1. Οπότε, στέλνουν μια ανταπόκριση ("TrueClusterMsg") η οποία εμπεριέχει το όνομα του cluster στον οποίο ο Α θα έπρεπε να εκδίδει, δηλαδή τον cluster 2. Ο Α λαμβάνει το μήνυμα, αναβαθμίζει τον DCRT του, ώστε να δείχνει ότι η κατηγόρια" sci-fi" ανήκει στον cluster 2, και στέλνει το ίδιο "PublishMsg" σε όλους τους peers του cluster 2 (βήμα Β). Οι peers το λαμβάνουν, απαντάνε με ένα "TrueClusterMsg" επιβεβαιώνοντας στον Α ότι εκδίδει σωστά και αναβαθμίζουν τον NRT τους ώστε να φαίνεται ότι ο Α ανήκει και στον cluster 2.

- QueryProtocol:

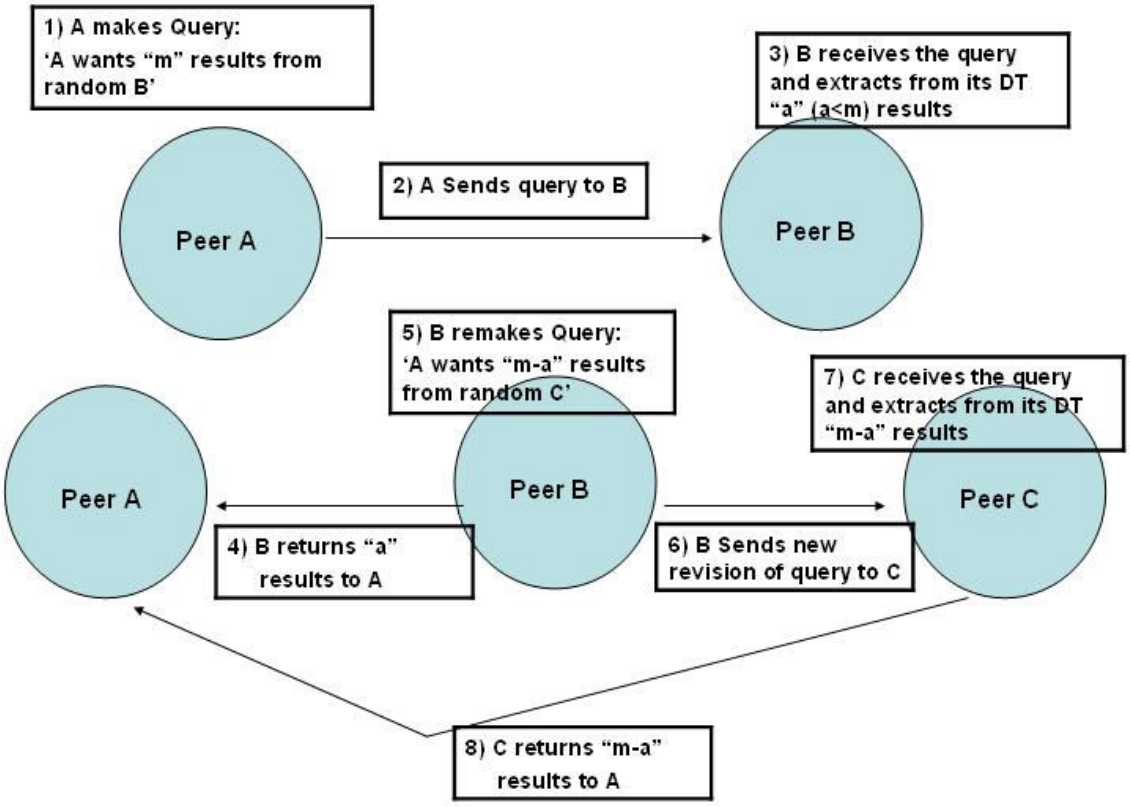
Ένας peer μπορεί να στείλει ένα query για να μάθει ποιοι peers φιλοξενούν κάποιες επιθυμητές εικονικές ταινίες. Το *QueryProtocol* είναι υπεύθυνο για την διαχείριση αυτών των queries που μπορούν να σταλθούν από όλους τους peers του Guerilla Sharing System. Το συγκεκριμένο πρωτόκολλο αποτελείται από ένα service, το *QueryService*:



Έστω ότι ένας peer p θέλει να στείλει ένα query ζητώντας έναν συγκεκριμένο αριθμό m επιθυμητών αποτελεσμάτων. Ο p εξάγει με την βοήθεια του dataset και των metadata του, τους clusters c_i που ανήκουν οι ζητούμενες ταινίες και στέλνει το query σε έναν τυχαίο peer p_i από κάθε c_i. Ο peer που θα λάβει το query επιστρέφει τα m επιθυμητά αποτελέσματα. Σύμφωνα με το θεωρητικό μοντέλο, αν ο p_i έχει λιγότερα από m αποτελέσματα, έστω a, να επιστρέψει στον p, προωθεί το μήνυμα σε όλους τους γνωστούς peers του c_i, μόνο που αντί για m, ζητάει m-a αποτελέσματα. Αυτό συνεχίζεται αναδρομικά μέχρι να βρεθούν m αποτελέσματα τα οποία και επιστρέφονται στον p από τον p_i. Με αυτόν τον τρόπο όμως, ο p_i συγκεντρώνει ουσιαστικά όλο τον φόρτο εργασίας, εφόσον αυτός πρέπει να αναλάβει να ζητήσει και να πάρει όλα τα επιθυμητά αποτελέσματα. Όποτε, πρακτικά, ο p_i επιστρέφει τα a αποτελέσματα στον p, προωθεί το μήνυμα σε έναν άλλον peer, έστω τον $p2_i$, του ίδιου cluster c_i και ο

$p2_i$ τρέχει από την αρχή το *QueryService* για να επιστρέψει αυτός τα υπόλοιπα αποτελέσματα και όχι μόνος του ο p_i. Αν ο $p2_i$ δεν έχει *m-a* αποτελέσματα να επιστρέψει προωθεί το μήνυμα σε έναν τρίτο peer κοκ, μέχρι τελικά να επιστραφούν *m* αποτελέσματα στον *p*.

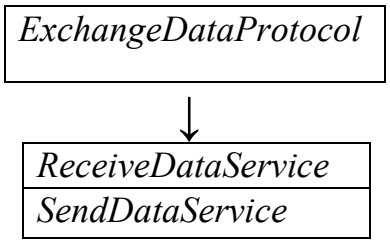
Το σχήμα 13 δείχνει την σειρά των βημάτων που χρειάζεται ο peer *A* για να συγκεντρώσει έναν επιθυμητό αριθμό αποτελεσμάτων σε ένα query που στέλνει αρχικά στον Peer *B*.



Σχήμα 13: Αποστολή Query

- ExchangeDataProtocol:

Παρατηρήσαμε, ότι για να διεκπεραιωθούν τα προηγούμενα πρωτόκολλα απαιτείται οι peers να ανταλλάσσουν μεταξύ τους πληροφορίες. Οι πληροφορίες αυτές στέλνονται με μηνύματα ("guerilla messages") μέσω unicast pipes μεταξύ δυο peers. Το *ExchangeDataProtocol* αναλαμβάνει να στείλει ή να λάβει ένα τέτοιο μήνυμα για λογαριασμό του peer. Τα services που αποτελείται είναι τα εξής:

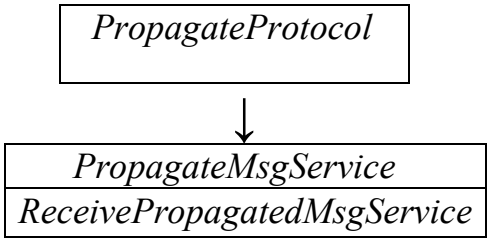


Όταν ένας peer θέλει να ανακτήσει κάποιες πληροφορίες από έναν άλλον peer, τότε χρησιμοποιώντας το *ReceiveDataService* στέλνει ένα μήνυμα "χαιρετισμού" σε έναν γνωστό απομακρυσμένο peer για να του ζητήσει τα επιθυμητά δεδομένα. Το μήνυμα "χαιρετισμού" ουσιαστικά γνωστοποιεί στον απομακρυσμένο peer δυο πράγματα:
1) Τι είδους δεδομένα θέλει ο αιτών peer.
2) Το ID του αιτούντος peer ώστε να μπορέσει να του επιστρέψει τα επιθυμητά δεδομένα

Ο απομακρυσμένος peer λαμβάνει το μήνυμα "χαιρετισμού" χρησιμοποιώντας την *SendDataService* και εξάγοντας την πληροφορία του μηνύματος καταλαβαίνει τι δεδομένα ακριβώς πρέπει να στείλει στον αιτών. Ο αποστολέας εισάγει τα δεδομένα που του ζητήθηκαν σε ένα "guerilla message", τα στέλνει στον παραλήπτη και ο τελευταίος τα εξάγει για να τα χρησιμοποιήσει το κατάλληλο service.

- PropagateProtocol:

Ένας peer, εκτός από το να στείλει ή να λάβει μηνύματα σε/από έναν άλλον συγκεκριμένο απομακρυσμένο peer, μπορεί επίσης να μεταδώσει ένα "guerilla message" σε όλους τους peers ενός συγκεκριμένου peergroup (Guerilla, Cluster ή Leader PeerGroup) .Το πρωτόκολλο που είναι υπεύθυνο για την αποστολή και την παραλαβή propagated "guerilla messages" ονομάζεται PropagateProtocol και υλοποιείται από τα παρακάτω services:



Το *ReceivePropagatedMsgService* αναλαμβάνει να παραλάβει ένα message που έχει μεταδοθεί σε ένα συγκεκριμένο peergroup. Όταν το μήνυμα ληφθεί, ανακτώνται τα δεδομένα που έχουν εισαχθεί σε αυτό και επεξεργάζονται από το κατάλληλο service.

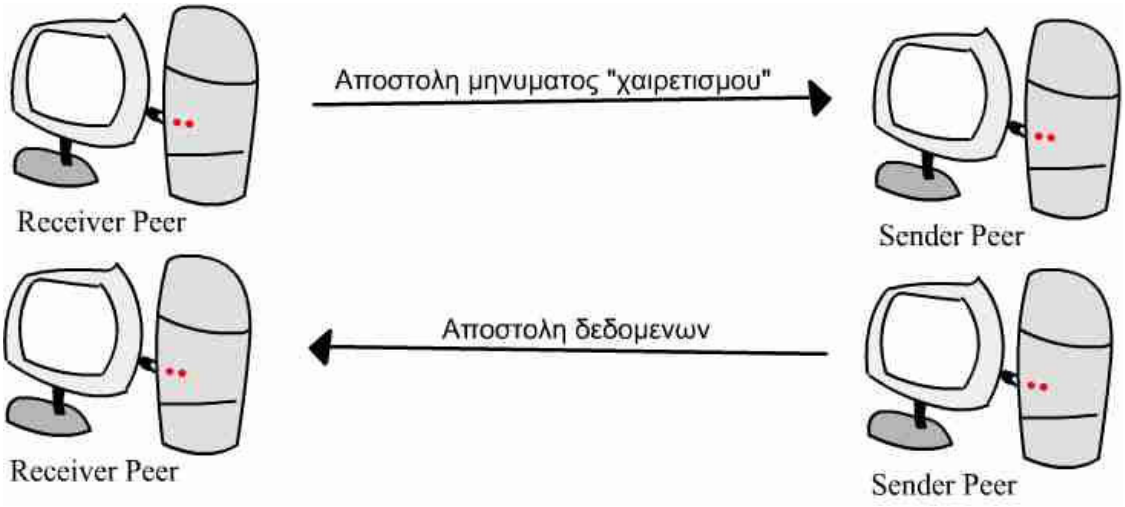
Το *PropagateMsgService* υλοποιεί όλα τα βήματα για την μετάδοση ενός "guerilla message" σε έναν συγκεκριμένο Cluster. Τα βήματα αυτά είναι:

- Αν ένας peer θέλει να στείλει το message σε ένα peergroup που ανήκει, τότε απλά εισάγει τα δεδομένα που θέλει να στείλει σε ένα guerilla message και το μεταδίδει.
- Αν ο peer θέλει να στείλει το message σε ένα peergroup που δεν ανήκει τότε:

1) Πρέπει να βρει το advertisement που περιγράφει αυτό το peergroup κάνοντας αναζήτηση στις τοπικές caches απομακρυσμένων peers ή του server χρησιμοποιώντας τις λειτουργικότητες του JXTA.
2) Όταν το βρει γίνεται μέλος του peergroup και χρησιμοποιώντας το *ReceivePropagatedMsgService* μπορεί να "ακούει" για guerilla messages που αφορούν το συγκεκριμένο peergroup.
3) Πρέπει να βρει και να συνδεθεί με έναν rendezvous peer. Όπως είχαμε πει ένας rendezvous peer έχει αποθηκευμένη στην τοπική του cache μια λίστα με τα resources άλλων rendezvous peers και απλών peers που έχουν συνδεθεί μαζί του. Επομένως, το προς μετάδοση μήνυμα πρέπει αρχικά να σταλθεί σε έναν rendezvous peer για να το στείλει εκ μέρους του αποστολέα στους γνωστούς του peers και οποίοι από αυτούς είναι rendezvous να προωθήσουν με τον ίδιο τρόπο το μήνυμα μέχρι να το παραλάβουν όλοι οι peers του peergroup.
4) Εισάγει τα δεδομένα που θέλει να στείλει σε ένα guerilla message και χρησιμοποιώντας τις κατάλληλες υπηρεσίες του JXTA το μεταδίδει.

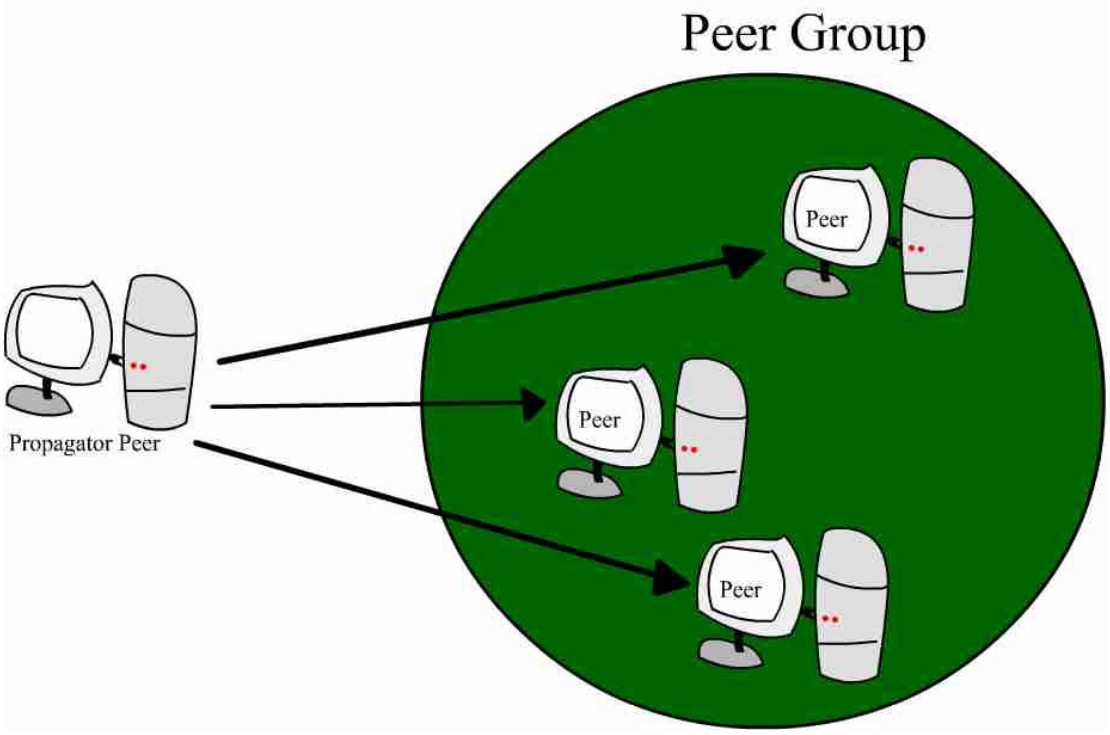
Τρόποι Επικοινωνίας: *Σύμφωνα με τα παραπάνω ο κάθε peer μπορεί να στείλει και να παραλάβει "guerilla messages" με τρεις διαφορετικούς τρόπους:*

1) Ο peer στέλνει ένα request message ζητώντας δεδομένα από κάποιον άλλον peer και αυτός του τα επιστρέφει σε ένα response message (μέσω unicast pipes).
Υλοποίηση: ExchangeDataProtocol



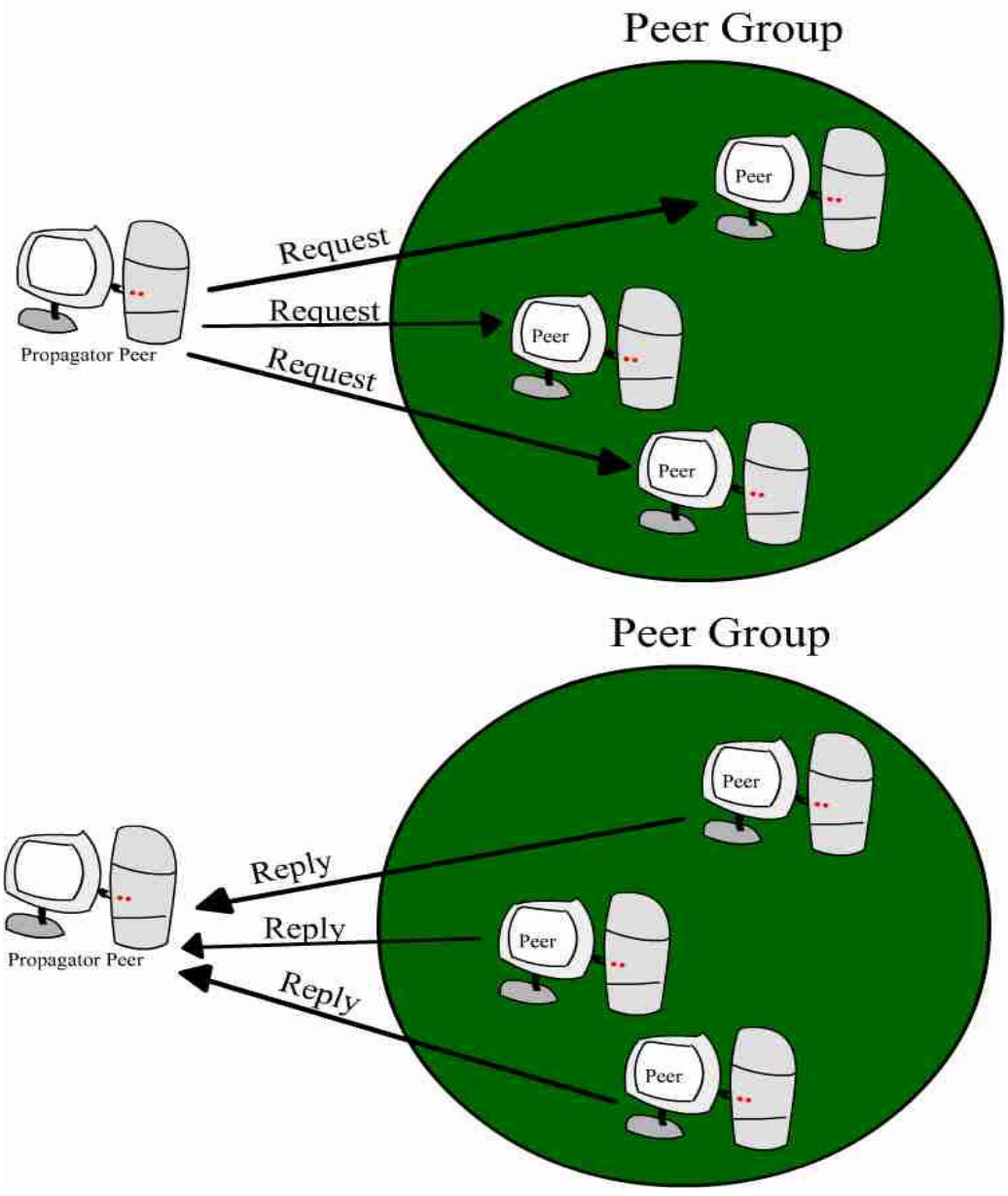
Σχήμα 14: RR (Request-Response)

2) Ο peer κάνει propagate ένα message εντός ενός peergroup χωρίς να περιμένει κάποια ανταπόκριση (μέσω propagated pipes). Υλοποίηση: PropagateProtocol



Σχήμα 15: GP (Group Propagation)

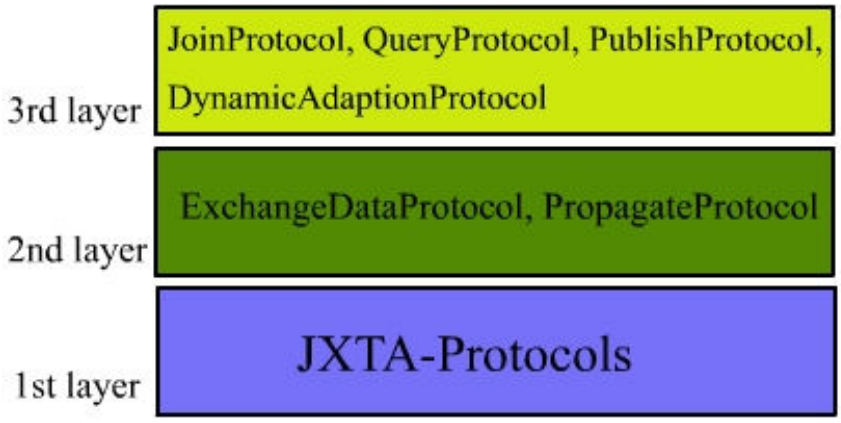
3) Ο peer κάνει propagate ένα request message εντός ενός peer group ζητώντας δεδομένα από όλα μέλη του (μέσω propagated pipes), τα οποία και του τα επιστρέφουν σε ένα response message (μέσω unicast pipes).
Υλοποίηση: PropagateProtocol σε συνδυασμό με το ExchangeDataProtocol.



Σχήμα 16: GPR (Group Propagation Response)

Για να γίνει πιο εύκολη η χρήση των πρωτοκόλλων επικοινωνίας αναπτύχθηκε ένα service, το GeneralDataTransferService, το οποίο εκτελεί τον επιθυμητό τρόπο επικοινωνίας χρησιμοποιώντας τα πρωτόκολλα αυτά.

Η λειτουργία του *JoinProtocol*, του *QueryProtocol*, του *PublishProtocol* και του *DynamicAdaptationProtocol* βασίζεται στο *PropagateProtocol* και στο *ExchangeDataProtocol* (σχήμα 17). Για να πετύχουν τον σκοπό τους τα high-level πρωτόκολλα χρειάζονται την επικοινωνία μεταξύ των peers, δηλαδή την ανταλλαγή δεδομένων μεταξύ τους, κάτι που τους τα παρέχουν τα πρωτόκολλα επικοινωνίας. Κάθε πρωτόκολλο αναλόγως με την λειτουργικότητα που θέλει να υλοποιήσει χρησιμοποιεί και έναν από τους τρεις παραπάνω τρόπους επικοινωνίας



Σχήμα 17: Ιεραρχία πρωτοκόλλων

Στο παρακάτω πίνακα παρατηρούμε ποιον τρόπο επικοινωνίας χρησιμοποιεί κάθε service για να επιτύχει το σκοπό του:

Μέθοδος:	*Services:*	*λειτουργικότητα:*
RR	JoinService (JoinProtocol)	Ο peer κατά το bootstrapping του ζητάει και ανακτά τα metadata από έναν άλλον τυχαίο peer.
	QueryService (QueryProtocol)	Ο peer στέλνει query σε έναν άλλον peer και παραλαμβάνει query response.
GP	LeaderElectionService (DynamicAdaptationProtocol)	Κάθε peer μεταδίδει εντός ενός cluster την τιμή της υπολογιστικής του ισχύς σε όλα τα μέλη του cluster.
	LeaderCommunicationService (DynamicAdaptationProtocol)	Κάθε leader μεταδίδει στο LeaderPeerGroup τα χαρακτηριστικά του cluster του.
	LazyRebalancingService (DynamicAdaptationProtocol)	Ο chosen leader μεταδίδει σε όλους τους peers του source και του destination cluster πληροφορία σχετική με την μετακίνηση της κατηγορίας.
	Αποχώρηση peer	Κατά την έξοδο του από το σύστημα ένας peer ενημερώνει τα υπόλοιπα μέλη των clusters που ανήκει (η λειτουργικότητα αυτή δεν καλύπτεται από κάποιο service).

GPR	UpdateMetadataService (PublishProtocol)	Ο peer που εκδίδει ένα αρχείο κατηγορίας *s* σε έναν cluster *c,* ενημερώνει όλους τους peers του *c* για την ένταξη του σε αυτόν. Κάθε peer απαντάει επιστρέφοντας τον cluster που φιλοξενεί πραγματικά την *s* σύμφωνα με τα metadata τους.
	ClusterMonitoringService (DynamicAdaptationProtocol	Ο leader μεταδίδει εντός του cluster του ένα μήνυμα για να ζητήσει τα load-figures των peers. Κάθε peer τα επιστρέφει στον leader μέσω unicast pipes.

Πίνακας 2: Μέθοδοι Επικοινωνίας που χρησιμοποιούν τα Guerilla services

Κεφάλαιο 4

ΥΛΟΠΟΙΗΣΗ
& ΑΞΙΟΛΟΓΗΣΗ

Το Guerilla είναι ένα mixed P2P σύστημα ανταλλαγής αρχείων στο οποίο οι peers μπορούν να εκδώσουν εικονικά αρχεία ή να ζητήσουν να μάθουν ποιοι peers φιλοξενούν έναν συγκεκριμένο αριθμό επιθυμητών αρχείων.

Οι peers ανήκουν στο Guerilla PeerGroup, στο Cluster PeerGroup που έχουν εκδώσει αρχεία και αν είναι leaders στο Leader PeerGroup. Οι peers γίνονται rendezvous αν ενταχθούν σε κάποιο peer group εκτός του Guerilla.

Στο Guerilla υπάρχει αναγκαστικά ένας server με γνωστό IP, ο οποίος έχει τον ρόλο του κόμβου εκκίνησης παρέχοντας στον προς ένταξη peer την διεύθυνση ενός άλλου γνωστού κόμβου που είναι ήδη συνδεδεμένος στο δίκτυο. Ο server δεν κρατάει τα πραγματικά δεδομένα, αλλά οι ίδιοι οι peers που ανακαλύπτουν ο ένας τον άλλον χρησιμοποιώντας μηχανισμούς αναζήτησης του JXTA.

Οι λειτουργίες των peers εκτελούνται από ένα σύνολο πρωτοκόλλων τα οποία απαιτούν την επικοινωνία μεταξύ των peers. Η επικοινωνία αυτή υλοποιείται με την αποστολή/παραλαβή σε/από έναν συγκεκριμένο peer ή την μετάδοση σε ένα σύνολο από peers ειδικού τύπου JXTA messages, τα *guerilla messages.*

Για να σταλθεί και να παραληφθεί ένα *message* χρειάζεται την δημιουργία μιας unicast pipe μεταξύ των peers που συμμετέχουν στην επικοινωνία. Οι *listeners* είναι JXTA objects που γνωστοποιούν ασύγχρονα μια υπηρεσία ποτέ δημιουργείται μια pipe και είναι υπεύθυνα για την δημιουργία του προς αποστολή μηνύματος ή την επεξεργασία του μηνύματος που έχει

παραληφθεί.

Τα δεδομένα που εισάγονται σε ένα μήνυμα ανακτούνται από ειδικές δομές δεδομένων, τα *guerilla metadata*. Αντίστοιχα, τα δεδομένα που εξάγονται από ένα μήνυμα που έχει παραληφθεί από κάποιον *listener* αποθηκεύονται στα *metadata*.

4.1 – Guerilla Listeners

Όλα τα πρωτόκολλα του Guerilla Sharing System είναι βασισμένα στην ανταλλαγή μηνυμάτων μεταξύ των peers. Ένας peer που θέλει να στείλει ένα μήνυμα δημιουργεί μια output pipe η οποία μπορεί να περιγραφτεί με μια pipe advertisement. Ο peer που δέχεται το μήνυμα πρέπει να δημιουργήσει μια input pipe που να περιγράφεται από μια ίδια pipe advertisement. Έτσι, δημιουργείται ένας δίαυλος επικοινωνίας μεταξύ των peers και πλέον μπορεί ο αποστολέας να στείλει ένα μήνυμα στον παραλήπτη. Όμως, οι peers πρέπει με κάποιο τρόπο να ειδοποιούνται όταν ένας δίαυλος δημιουργείται. Το JXTA παρέχει objects που γνωστοποιούν ασύγχρονα ένα service πότε λήφθηκε ένα response σε κάποιο request. Τα objects αυτά λέγονται listeners και γίνονται register από τα services του Guerilla, ώστε να ενημερώνονται κάθε φορά που ένα output pipe γίνεται bind σε κάποιο input pipe.Οι listeners του Guerilla Sharing System είναι χωρισμένοι στις εξής κατηγορίες:

- Listeners που ακούν για την δημιουργία κάποιου pipe ή κάποιου propagated message που λήφθηκε. Δηλαδή listeners που ξυπνάνε ασύγχρονα κατά την αποστολή ή την λήψη των messages που σχετίζονται με την λειτουργικότητα των πρωτοκόλλων του Guerilla Sharing System: Οι listeners αυτοί είναι οι εξής:

HandshakeOutputPipeListener: Γίνεται register όταν ένας peer θέλει να στείλει κάποιο μήνυμα «χαιρετισμού», όταν δηλαδή θέλει να ζητήσει κάποια δεδομένα από έναν απομακρυσμένο peer και ο τελευταίος μπορεί να τον εξυπηρετήσει μόνο μέσω unicast pipes.

HandshakePipeListener: Γίνεται register στο peer bootstrapping για να μπορεί να λαμβάνει όλα τα μηνύματα «χαιρετισμού» που θα απευθύνονται σε αυτόν. Κάθε peer μπορεί με αυτό τον τρόπο να εξυπηρετεί άλλους peers κάθε φορά που του ζητείται.

LoadFiguresOutputPipeListener: Γίνεται register όταν ένας peer πρέπει να στείλει τα load-figures του στον leader του αντίστοιχου cluster. Τα load figures συλλέγονται από τον leader για να υπολογίσει το normalized popularity του cluster του.

LoadFiguresPipeListener: Γίνεται register όταν ο leader ενός cluster πρέπει να λάβει τα load-figures του από έναν peer του cluster.

MetadataOutputPipeListener: Γίνεται register όταν ένας peer πρέπει να στείλει κάποια συγκεκριμένα metadata (DCRT και NRT) που έχει αποθηκευμένα σε έναν απομακρυσμένο peer κατά το bootstrapping του τελευταίου. Τα metadata αυτά είναι χρήσιμοι πίνακες που βοηθάνε έναν peer να ενταχθεί στο Guerilla.

MetadataPipeListener: Γίνεται register όταν ένας peer είναι έτοιμος να λάβει τα metadata που έχει ζητήσει από κάποιον απομακρυσμένο peer κατά το bootstrapping του.

QueryOutputPipeListener: Γίνεται register όταν ένας peer πρέπει να ανταποκριθεί σε ένα query που του έχει σταλθεί από έναν απομακρυσμένο peer.

QueryPipeListener: Γίνεται register όταν ένας peer είναι έτοιμος να λάβει την ανταπόκριση στο query που έχει στείλει σε έναν απομακρυσμένο peer.

TrueClusterOutputPipeListener: Γίνεται register όταν ένας peer έχει λάβει μια αίτηση από έναν απομακρυσμένο peer, με την οποία αίτηση ο τελευταίος ζητάει να μάθει τον cluster που φιλοξενεί μια συγκεκριμένη σημασιολογική κατηγορία επειδή θέλει να εκδώσει σε αυτόν ένα αρχείο. Η ανταπόκριση που στέλνεται περιέχει ουσιαστικά το όνομα του cluster που αντιστοιχεί στην ερωτούμενη κατηγορία.

TrueClusterPipeListener: Γίνεται register όταν ένας peer έχει στείλει μια αίτηση σε έναν peer για να μάθει τον cluster που φιλοξενεί μια συγκεκριμένη σημασιολογική κατηγορία. Όταν ο απομακρυσμένος peer στείλει την ανταπόκριση, ο συγκεκριμένος listener "ξυπνάει" και ανακτά τα δεδομένα (τον αριθμό, δηλαδή, του cluster που φιλοξενεί την κατηγορία).

PropagateListener: Γίνεται register όταν ένας peer γίνει μέλος ενός peergroup του Guerilla, ώστε να ακούει για τα messages που μεταδίδονται σε αυτό το peergroup.

- Listeners που ξυπνάνε ασύγχρονα κατά την ανακάλυψη κάποιου resource του Guerilla network (απλοί peers, rendezvous peers ή peergroups):

PeerGroupListener: Γίνεται register όταν ένας peer θέλει να ανακαλύψει το advertisement ενός peergroup για να μπορεί να γίνει μέλος σε αυτό. Ένας peer δεν μπορεί να ενταχθεί σε κάποιο peergroup αν πρώτα δεν έχει ανακαλύψει το advertisement του.

PeerListener: Γίνεται register όταν ένας peer θέλει να ανακαλύψει κάποιο peer advertisement μέσα σε ένα συγκεκριμένο peergroup. Όταν βρεθεί το advertisement, τότε ένα service μπορεί να εξάγει όλα τα χαρακτηριστικά του peer που έχει ανακαλυφθεί και να τα χρησιμοποιήσει αναλόγως με την λειτουργικότητα που αυτό εξυπηρετεί (για παράδειγμα εξάγοντας το ID του απομακρυσμένου peer ο τοπικός peer έχει την δυνατότητα να στείλει ένα μήνυμα συγκεκριμένα σε αυτόν).

RdvListener: Γίνεται register όταν ένας peer θέλει να συνδεθεί σε έναν rendezvous peer. Ο RdvListener ξυπνάει όταν ο rendezvous peer ανακαλυφθεί και επιστρέφει το advertisement που τον περιγράφει στην υπηρεσία που έκανε register τον listener. Η υπηρεσία ανακτά το ID του rendezvous peer και με τις κατάλληλες μεθόδους του JXTA γίνεται η σύνδεση με αυτόν. Και πάλι, το ID του rendezvous peer είναι απαραίτητο για έναν peer που θέλει να συνδεθεί σε αυτόν.

4.2 - Guerilla Metadata

Τα metadata στο Guerilla Sharing System είναι data structures αποθηκευμένα σε κάθε peer για να επικοινωνεί και να ανταλλάσσει πληροφορίες με άλλους ή να βρίσκει peers που έχουν αποθηκευμένες επιθυμητές πληροφορίες. Ουσιαστικά, είναι tables που αποθηκεύουν πληροφορίες για τον peer, για ένα peergroup ή ακόμα και για όλο το σύστημα. Τα tables αυτά είναι τα εξής:

DT (Document Table): Κάθε εικονικό αρχείο ανήκει σε μια σημασιολογική κατηγορία. Ο DT περιέχει το όνομα του κάθε αρχείο που έχει εκδώσει ο peer σε αντιστοιχία με την σημασιολογική κατηγορία που αυτό ανήκει. Όταν εκδίδεται ένα αρχείο, το πρωτόκολλο PublishProtocol αναλαμβάνει να

εξάγει από το dataset την κατηγορία που ανήκει αυτό το αρχείο και να αποθηκεύσει την δυάδα *όνομα αρχείου-σημασιολογική κατηγορία* στον DT. Ένα παράδειγμα ενός DT που μπορεί να βρίσκεται αποθηκευμένος σε έναν peer είναι το παρακάτω:

αρχείο	*Σημασιολογική κατηγορία*
"Wyatt Earp"	"Western"
"M. Butterfly"	"Drama"
"Rudy"	"Drama"
"Alphaville"	"Sci-fi"

Πίνακας 3: Υπόδειγμα DT

PopularityTable: Κάθε εικονικό αρχείο έχει μια τιμή που χαρακτηρίζει το popularity του. Η τιμή αυτή βρίσκεται εντός της κλίμακας [0-1] και όσο πιο πολύ προσεγγίζει το 1, τόσο πιο υψηλή ζήτηση έχει το αντίστοιχο αρχείο. Όταν εκδίδεται ένα αρχείο τότε το πρωτόκολλο PublishProtocol αναλαμβάνει να εξάγει από το dataset το popularity του και να αποθηκεύσει την δυάδα *όνομα αρχείου-popularity* στον PopularityTable. Παρακάτω, υποδεικνύεται το παράδειγμα ενός PopularityTable:

αρχείο	Popularity
"Jurassic Park"	0,86
"Life of Brian"	0,77
"Aliens"	0,67
"Once Upon a Time in West"	0,42

Πίνακας 4: Υπόδειγμα PopularityTable

DCRT (Document CategoryRouting Table): Το συγκεκριμένο table δείχνει σε ποιον cluster ανήκει η κάθε σημασιολογική κατηγορία. Ο DCRT ανακτάται κατά το bootstrapping ενός peer από έναν άλλον τυχαίο peer που βρίσκεται συνδεδεμένος στο σύστημα. Παρακάτω, φαίνεται ένας τυχαίος DCRT:

Σημασιολογική κατηγορία	*ClusterNo*
"Western"	0
"Comedy"	1

“Adventure”	1
“Sci-fi”	2

Πίνακας 5: Υπόδειγμα DCRT

NRT (NodeRoutingTable): Ο NRT περιέχει τα ID των peers που ανήκουν σε κάθε cluster. Ο NRT ανακτάται κατά το bootstrapping ενός peer από έναν άλλον τυχαίο peer που βρίσκεται συνδεδεμένος στο σύστημα. Έτσι, όταν για παράδειγμα ένας peer θέλει να στείλει ένα query σε έναν απομακρυσμένο peer για να τον ρωτήσει αν έχει αποθηκευμένο ένα συγκεκριμένο αρχείο *doc*, εξάγει από τον DT την σημασιολογική κατηγορία *cat* στην οποία ανήκει το *doc*, συμβουλεύεται τον DCRT ως προς τον cluster *c* που φιλοξενεί την *cat* και τελικά επιλέγει από τον NRT έναν τυχαίο peer που ανήκει στον *c* για να του στείλει το μήνυμα. Σε ένα σύστημα με τέσσερις clusters ο NRT θα μπορούσε να είναι ως εξής:

Cluster	*Peer IDs*			
0	urn:jxta:uuid-59616261646...	urn:jxta:uuid-35646757578...		
1	urn:jxta:uuid-9668544367...	urn:jxta:uuid-123EE768786...	urn:jxta:uuid-39282891991...	urn:jxta:uuid-78929ACFE34...
2	urn:jxta:uuid-123EE768786...			
3	urn:jxta:uuid-59616261646...	urn:jxta:uuid-7878CEA34FE...	urn:jxta:uuid-123EE768786...	

Πίνακας 6: Υπόδειγμα NRT

PowerTable: Κατά την διάρκεια εκλογής αρχηγού (LeaderElectionService του DynamicAdaptationProtocol), κάθε peer μεταδίδει σε όλα τα μέλη των clusters που ανήκει ένα μήνυμα με την τιμή της υπολογιστικής του ισχύς σε κλίμακα [1-100] και λαμβάνει παρόμοια μηνύματα από τους υπόλοιπους peers. Όλες αυτές οι τιμές αποθηκεύονται σε ένα array από tables, τα PowerTables. Αν ένας peer ανήκει σε δυο clusters ταυτόχρονα (έστω τον cluster *1* και τον cluster *3*), τότε έχει ένα PowerTable για να αποθηκεύει τις μονάδες της υπολογιστικής ισχύς όλων των peers του cluster *1* και ένα PowerTable για τους peers του cluster *3*. Τα PowerTables για αυτόν τον peer θα μπορούσαν να είναι ως εξής:

PowerTable[1] (αντιστοιχεί στον cluster 1):

PeerIDs	*Computational Units*
urn:jxta:uuid-9668544367...	80
urn:jxta:uuid-123EE768786...	23
urn:jxta:uuid-39282891991...	11
urn:jxta:uuid-78929ACFE34...	93

PowerTable[3] (αντιστοιχεί στον cluster 3):

PeerIDs	*Computational Units*
urn:jxta:uuid-59616261646...	45
urn:jxta:uuid-7878CEA34FE...	69
urn:jxta:uuid-123EE768786...	23

Πίνακας 7: Υπόδειγμα PowerTable

Παρατηρούμε, ότι όταν ο peer λάβει μήνυμα από όλους τους peers του cluster *1* και του cluster *3*, μπορεί πλέον να επιλέξει τον αρχηγό. Στο συγκεκριμένο παράδειγμα αρχηγός στον cluster *1* θα ήταν ο peer με ID *urn:jxta:uuid-78929ACFE34...* και στον cluster *3* ο *urn:jxta:uuid-7878CEA34FE...*

ClusterLoadFiguresTable: Όταν ένας peer εκλεχθεί αρχηγός σε ένα cluster c_i, ζητάει τα load-figures από όλους τους peers που ανήκουν στον c_i. Ο ClusterLoadFiguresTable αποθηκεύει τα load-figures που λαμβάνει από τον κάθε peer. Με αυτά τα load-figures θα μπορεί να υπολογίσει αργότερα το normalized popularity του c_i (τα normalized popularities των clusters αποτελούν όροι στην εξίσωση του fairness index). Το normalized polularity ενός cluster c_i δίνεται από τον τύπο:

$$\textit{norm-pop of cluster}_i = p(S_i) \Big/ \sum_{k \in N_i} \left(u_k \cdot p(D_i(k)) / p(D(k)) \right)$$

όπου k ένας peer του συστηματος και N_i το σύνολο των peers που ανήκουν στον cluster c_i. Η μεταβλητή p συμβολίζει το popularity, το σύνολο S

περιέχει όλες τις σημασιολογικές κατηγορίες του συστηματος και το S_i είναι το σύνολο των σημασιολογικών κατηγοριών που ανήκουν στον cluster c_i. Επομένως, το $p(S_i)$ εκφράζει το popularity του συνόλου S_i. Η μεταβλητή $p(D(k))$ εκφράζει το άθροισμα των popularities όλων των εικονικών αρχείων που έχει αποθηκευμένα ο peer k, η $p(D_i(k))$ το συνολικό popularity των αρχείων των οποίων οι κατηγορίες ανήκουν στον cluster c_i και είναι αποθηκευμένα στον peer k και η uk τις μονάδες της υπολογιστικής ισχύς του k.

Το ζητούμενο είναι τι καταχωρήσεις ακριβώς χρειάζεται ο ClusterLoadFiguresTable, δηλαδή τι πληροφορίες πρέπει να στείλει ο κάθε peer του N_i στον leader του c_i κατά το ClusterMonitoringService, ώστε να είναι επαρκείς για να υπολογίσει ο leader το normalized popularity του c_i. Ο κάθε peer εισάγει όλες τις πληροφορίες σε ένα message ειδικού format (*"LoadFiguresMsg"*) και τις στέλνει στον leader, ο οποίος τις εξάγει για να υπολογίσει το normalized popularity.Οπότε, ο leader πρέπει βάσει των δεδομένων που λαμβάνει να μπορεί να γνωρίζει τις τιμές των μεταβλητών $p(S_i)$, uk, $p(D(k))$ και $p(D_i(k))$ για κάθε peer k που ανήκει στον c_i.

- Το $p(S_i)$ υπολογίζεται αν κάθε peer του c_i στείλει στον leader του την τιμή της μεταβλητής $p(S_i(k))$, δηλαδή το άθροισμα των popularities των αρχείων που έχει εκδώσει ο peer k και ανήκουν στον c_i. Το $p(S_i(k))$ υπολογίζεται από τον κάθε peer συνδυάζοντας την πληροφορία που δίνει ο DT, ο DCRT και ο PopularityTable. Άρα, ο leader αθροίζοντας τα $p(S_i(k))$ που λαμβάνει βρίσκει την τιμή της μεταβλητής $p(S_i)$.

- Το $p(D_i(k))$ είναι γνωστό για κάθε peer k που ανήκει στον c_i γιατί είναι ισοδύναμο με το $p(S_i(k))$.

- Το $p(D(k))$ εκφράζει το άθροισμα των popularities όλων των αρχείων που έχει εκδώσει ο peer k. Άρα, ο κάθε peer υπολογίζει την τιμή της $p(D(k))$ αθροίζοντας όλα τα popularities που εξάγει από τον PopularityTable του.

- Η μεταβλητή uk είναι γνωστή για κάθε peer, οπότε αρκεί να στείλει την τιμή της στον leader.

Επομένως, ο κάθε peer πρέπει να εισάγει στο message τις τιμές των μεταβλητών $p(S_i(k))$, $p(D(k))$ και uk. Όμως, μπορεί αυτή η πληροφορία να είναι αρκετή για τον leader ώστε να μπορεί να υπολογίσει την τιμή του normalized popularity του cluster που είναι αρχηγός, αλλά θα είναι ανεπαρκής

για τον chosen leader όταν κατά την διάρκεια της εκτέλεσης του MaxFairReassignService θα θέλει να υπολογίσει τα normalized popularities που θα προκύπτουν από τις υποθετικές μεταφορές σημασιολογικών κατηγοριών σε άλλους clusters.

Ο κάθε peer πρέπει, επιπλέον, να στείλει το συνολικό popularity της κάθε κατηγορίας που ανήκει στον c_i και στην οποία έχει εκδώσει αρχεία.
Έτσι, για κάθε πιθανή μετακίνηση αυτής της κατηγορίας ο chosen leader θα μπορεί να υπολογίσει τους παράγοντες $p(S_i)$, uk, $p(D(k))$ και $p(D_i(k))$ για τον cluster στον οποίο μεταφέρεται η κατηγορία και για τον cluster από τον οποίο αυτή αποχωρεί.

Επομένως, κάθε καταχώρηση στο ClusterLoadFiguresTable είναι ένα array με τα παρακάτω στοιχεία:

[strClusterNo PeerID $p(S_i(k))$ $p(D(k))$ $u(k)$ $S_i1(k)$ $pS_i1(k)$ $S_i2(k)$ $pS_i2(k)$*.....]*

όπου, *strClusterNo:* Ο cluster που ο peer είναι αρχηγός
PeerID: Το ID του peer που έχει στείλει αυτές τις πληροφορίες
$S_i j(k)$: Το όνομα της j-ιοστής κατηγορίας στην οποία έχει εκδώσει αρχεία ο peer k και ανήκει στον c_i.
$pS_i j(k)$: Το popularity του συνόλου $S_i j(k)$.

Υποθέτουμε ότι ο *cluster 0* αποτελείται από τρεις peers οι οποίοι έχουν τα εξής metadata:

Peer 1

DT:

"Jade"	"Thriller"
"Georgia"	"Drama"

DCRT:

Thriller	Cluster 0
Drama	Cluster 0
Comedy	Cluster 1

PopularityTable:

“Jade”	0,35
“Georgia”	0,55

Αριθμός υπολογιστικών μονάδων: 40

Peer 2:

DT:

“Restoration”	“Drama”
“Apocalypse Now”	“Drama”
“Life of Brian”	“Comedy”

DCRT:

Thriller	Cluster 0
Drama	Cluster 0
Comedy	Cluster 1

PopularityTable:

“Restoration”	0,10
“Apocalypse Now”	0,90
“Life of Bryan”	0,80

Αριθμός υπολογιστικών μονάδων: 60

Peer 3:

DT:

“Annie Hall”	“Comedy”
“Amadeus”	“Drama”

DCRT:

Thriller	Cluster 0
Drama	Cluster 0
Comedy	Cluster 1

PopularityTable:

“Annie Hall”	0,25
“Amadeus”	0,35

Αριθμός υπολογιστικών μονάδων: 70

Βάσει αυτών των πινάκων ο *ClusterLoadFiguresTable* του leader (δηλαδή του Peer 3) θα έχει τις εξής καταχωρήσεις:

Cluster	*PeerID*	*$p(S_0(k))$*	*$p(D(k))$*	*$u(k)$*	*$S_01(k)$*	*$pS_01(k)$*	*$S_02(k)$*	*$pS_01(k)$*
0	Peer’s 1 ID	0,90	0,90	40	Thriller	0,35	Drama	0,55
0	Peer’s 2 ID	1,00	1,80	60	Drama	1,00		
0	Peer’s 3 ID	0,35	0,60	70	Drama	0,35		

Πίνακας 8: Υπόδειγμα ClusterLoadFiguresTable

NormalizedPopularityTable: Προαναφέρθηκε, ότι κατά το LeaderCommunicationService του DynamicAdaptationProtocol οι leaders

ανταλλάσσουν μεταξύ τους τις τιμές των normalized popularities που έχουν υπολογίσει για τους clusters στους οποίους είναι αρχηγοί. Κάθε leader αποθηκεύει τις τιμές αυτές στην δομή NormalizedPopularityTable. Επιπλέον, ανταλλάσσουν και αποθηκεύουν και τις τιμές των $p(S_i)$ του κάθε cluster γιατί ο chosen leader χρειάζεται αυτήν την πληροφορία στον υπολογισμό των normalized popularities που θα προκύπτουν κατά τις υποθετικές ανακατατάξεις των κατηγοριών. Το στιγμιότυπο ενός *NormalizedPopularityTable* που έχει αποθηκευμένο ένας leader σε ένα σύστημα με τρεις clusters θα μπορούσε να είναι το εξής:

Cluster i	*LeaderID*	*NormPopularity*	$p(S_i)$
0	Leader's ID of Cluster 0	0,0122	2,149
1	Leader's ID of Cluster 1	0,0354	3,544
2	Leader's ID of Cluster 2	0,0232	3,403

Πίνακας 9: Υπόδειγμα NormalizedPopularityTable

Έχοντας αυτές τις τιμές, ο cluster με το μεγαλύτερο normalized popularity μπορεί να υπολογίσει την τιμή του fairness index που δίνεται από τον τύπο:

$$\textit{fairness index value} = \left(\sum_{i=0}^{n-1} x_i\right)^2 / n\left(\sum_{i=0}^{n-1} x_i^2\right)$$

όπου n ο αριθμός των clusters και x_i το normalized popularity του cluster i.

4.3 - Guerilla Messages

Τα Guerilla messages είναι συγκεκριμένου τύπου JXTA messages που οι peers στέλνουν μεταξύ τους για να επιτύχουν κάποιες λειτουργικότητες. Όλα τα πρωτόκολλα του Guerilla Sharing System είναι βασισμένα στην ανταλλαγή μηνυμάτων μεταξύ των peers. Υπάρχουν διαφόρων ειδών messages, όπου το καθένα εξυπηρετεί άλλο σκοπό και ξεχωρίζει από τα υπόλοιπα από το tagname του. Ένα message μπορεί να στέλνεται μέσω unicast pipes (request ή response message) ή μέσω propagation. Η λειτουργικότητα του κάθε μηνύματος

φαίνεται στον πίνακα IV του Παραρτήματος.

4.4 - Επικοινωνία μεταξύ των peers.

Το παρών κεφάλαιο περιγράφει αναλυτικότερα κάθε πότε και πως επικοινωνούν μεταξύ τους οι peers και τι είδους *messages* χρησιμοποιούνται, ποιοι *listeners* γίνονται register και ποια *metadata* παρέχουν τις επιθυμητές πληροφορίες σε μια υπηρεσία για να επιτευχθεί μια συγκεκριμένη λειτουργικότητα στο Guerilla.

Για να στείλει ένα message *m* ο peer *A* στον peer *B*, πρέπει ο *A* να δημιουργήσει μια output pipe που είναι bind σε μια pipe advertisement *P* και να κάνει register τον κατάλληλο output pipe listener *out_lis*. Για να παραλάβει ο *B* το *m* πρέπει να έχει ήδη δημιουργήσει μια input pipe που να είναι bind στην *P* και να έχει κάνει register τον κατάλληλο input pipe listener *in_list*. Με λίγα λόγια, πρέπει αποστολέας και παραλήπτης να διαβάζουν την ίδια pipe advertisement. Για να μην δημιουργηθούν από κάποιο λάθος διαφορετικές pipe advertisements, αυτές δημιουργούνται στον κεντρικό server. Όταν εγκατασταθεί ο δίαυλος επικοινωνίας, ο *out_list*, που πρέπει οπωσδήποτε να γνωρίζει το ID του *B,* στέλνει το *m* και ο *in_list* το παραλαμβάνει και το επεξεργάζεται.

Για λόγους ευκολίας, υποδεικνύονται στο παρακάτω πίνακα οι τρεις τρόποι επικοινωνίας του Guerilla Sharing System:

Ways:	*Λειτουργικότητα:*	*Υλοποίηση:*
RR (Request-Response)	Ο peer στέλνει ένα request message ζητώντας δεδομένα από κάποιον άλλον peer και αυτός του τα επιστρέφει σε ένα response message (μέσω unicast pipes).	*ExchangeDataProtocol*
GP (Group Propagation)	Ο peer κάνει propagate ένα message εντός ενός peer group χωρίς να περιμένει κάποια ανταπόκριση (μέσω propagated pipes)	*PropagateProtocol*
GPR (Group Propagation-Response)	Ο peer κάνει propagate ένα request message εντός ενός peer group ζητώντας δεδομένα από όλα μέλη του (μέσω propagated pipes), τα οποία και του τα επιστρέφουν σε ένα response message (μέσω unicast pipes).	*ExchangeDataProtocol* + *PropagateProtocol*

Πίνακας 10: Τρεις Τρόποι Επικοινωνίας στο Guerilla

Από εδώ και στο εξής θα απευθυνόμαστε για το καθένα από τους τρεις διαφορετικούς τρόπους επικοινωνίας με το χαρακτηριστικό του όνομα, δηλαδή *RR, GP* ή *GPR*.

4.4.1 - Υλοποίηση RR

Ένας peer για να λάβει δεδομένα από έναν απομακρυσμένο ομότιμο του πρέπει πρώτα να του στείλει ένα *μήνυμα "χαιρετισμού"*. Όποτε, πρέπει να εξασφαλιστεί ότι όλοι οι peers μπορούν να δεχθούν *μηνύματα "χαιρετισμού"*, ώστε όλοι να έχουν την ικανότητα να στέλνουν πληροφορίες σε αιτήσεις που δέχονται.

Επομένως, κατά το bootstrapping ενός peer, το *JoinProtocol* εκτελεί την *ReceiveDataService* του *ExchangeDataProtocol* η οποία:
i) Δημιουργεί μια input pipe που είναι bind στην pipe advertisement *HandshakePipeAdv*.
ii) Γίνεται register ο *HandshakePipeListener*, ο οποίος ακούει για μηνύματα *"χαιρετισμού"* στην input pipe που εγκαταστάθηκε.

Ένας peer (*requested*) που θέλει να ζητήσει κάποια δεδομένα από έναν απομακρυσμένο ομότιμο του (*sender*) πρέπει να στείλει σε αυτόν ένα *μήνυμα "χαιρετισμού"* (ή *request message* όπως αναφέρθηκε στο προηγούμενο υποκεφάλαιο).

Για να γίνει αυτό, ο *requested peer* χρησιμοποιεί την *SendDataService* του *ExchangeDataProtocol* η οποία:
iii) Δημιουργεί μια output pipe, η οποία είναι bind στην pipe advertisement *HandshakePipeAdv*.
iv) Κάνει register έναν output pipe listener, τον *HandshakeOutputPipeListener*.

Εφόσον ο *sender peer* έχει ήδη από το bootstrapping του δημιουργήσει μια input pipe που είναι bind στην *HandshakePipeAdv*:
v) Δημιουργείται δίαυλος επικοινωνίας μεταξύ του *sender* και του *requested peer* γιατί και οι δυο διαβάζουν την ίδια pipe advertisement.
vi) Ο *HandshakeOutputPipeListener* "ξυπνάει" για να συντάξει και να στείλει το *μήνυμα "χαιρετισμού"* στον *sender peer*, εφόσον γνωρίζει το ID του.

vii) Ο *HandshakePipeListener* "ξυπνάει" για να λάβει και να επεξεργαστεί το *μήνυμα "χαιρετισμού"*.

Ένα *μήνυμα "χαιρετισμού"* αποτελείται από ένα string element το οποίο περιέχει δυο πεδία. Το πρώτο πεδίο είναι το *tagname*, που ορίζει τι είδους δεδομένα ζητάει ο *requested peer*, και το δεύτερο πεδίο ονομάζεται *data* και περιέχει τα πραγματικά δεδομένα του μηνύματος. Στον Πίνακα V του Παραρτήματος φαίνονται οι τιμές του πεδίου *data* για κάθε *μήνυμα "χαιρετισμού"*.

viii) Ο *requested peer*, αφού στείλει το *μήνυμα "χαιρετισμού*, δημιουργεί μια input pipe, η οποία γίνεται bind σε μια pipe advertisement, έστω την *p_adv*, και κάνει register τον inputpipe listener, έστω τον *in_list*, που θα λάβει τα δεδομένα. Η *p_adv* και ο *in_list* εξαρτώνται από τι είδους δεδομένα ζήτησε ο *requested peer*, δηλαδή από το tagname του *μηνύματος «χαιρετισμού»*. Έτσι έχουμε:

Tagname:	*p_adv:*	*in_list:*
"MetadataMsg"	*MetadataPipeAdv*	*MetadataPipeListener*
"QueryMsg"	*QueryPipeAdv*	*QueryPipeListener*

ix) Ο *HandshakePipeListener* δημιουργεί μια output pipe, η οποία γίνεται bind σε μια pipe advertisement, έστω την *p_adv*, και κάνει register τον outputpipe listener, έστω τον *out_list*, που θα στείλει τα δεδομένα. Η *p_adv* και ο *out_list* εξαρτώνται από τα δεδομένα που ζήτησε ο *requested peer*, δηλαδή από το tagname του *μηνύματος «χαιρετισμού»*. Έτσι έχουμε:

Tagname:	*p_adv:*	*out_list:*
"MetadataMsg"	*MetadataPipeAdv*	*MetadataOutputPipeListener*
"QueryMsg"	*QueryPipeAdv*	*QueryOutputPipeListener*

x) Δημιουργείται μια pipe μεταξύ των δυο peers, εφόσον είναι και οι δυο bind στην ίδια pipe advertisement *p_adv*, για να στείλει ο *out_list* του *sender peer* τα ζητούμενα δεδομένα σε μορφή μηνύματος και να τα λάβει ο *in_list* του *requested peer*. Το *tagname* και τα *data του μηνύματος απόκρισης (response message,* όπως αναφέρθηκε στο προηγούμενο υποκεφάλαιο*)* εξαρτώνται από το είδος του *μηνύματος "χαιρετισμού"*. Στον Πίνακα VI του Παραρτήματος φαίνονται οι τιμές του πεδίου *data* για κάθε *response message*.

Ο *HandshakePipeListener* διοχετεύει το ID του *requested peer,* που το γνωρίζει από τα data του *μηνύματος "χαιρετισμού"*, στον *out_list,* για μπορέσει αυτός να στείλει το μήνυμα στον συγκεκριμένο peer.

***Παράδειγμα**: Αποστολή και λήψη του πίνακα DCRT και NRT κατά το bootstrapping ενός peer*:

Έστω ότι ο peer A θέλει να ζητήσει τον πίνακα DCRT και NRT από τον peer B. Υποθέτουμε ότι οι πίνακες έχουν καταχωρημένες τις εξής πληροφορίες:

DCRT:

"Action"	*0*
"Documentary"	*1*

NRT:

0	`urn:jxta:uuid-59616261646...`	`urn:jxta:uuid-63AF6261007...`
1	`urn:jxta:uuid-49600B6F646...`	

1) Ο peer A κάνει register τον HandshakeOutputPipeListener και δημιουργεί μια output pipe που βλέπει στην HandshakePipeAdv.

2) Ο Peer B, ήδη από το bootstrapping του, έχει δημιουργήσει μια input pipe, η οποία βλέπει στην HandshakePipeAdv, και έχει κάνει register τον HandshakePipeListener.

3) Ο δίαυλος επικοινωνίας μεταξύ του A και του B εγκαθίσταται για να στείλει ο HandshakeOutputPipeListener το εξής μήνυμα "χαιρετισμού":

Tagname:	*data:*
"MetadataMsg"	*Peer's A ID*

Το ID του Peer B παρέχεται στον HandshakeOutputPipeListener από την InitialPeerDiscoveryService του JoinProtocol. Ο HandshakePipeListener "ξυπνάει" για να διαβάσει το μήνυμα "χαιρετισμού" του Peer A.

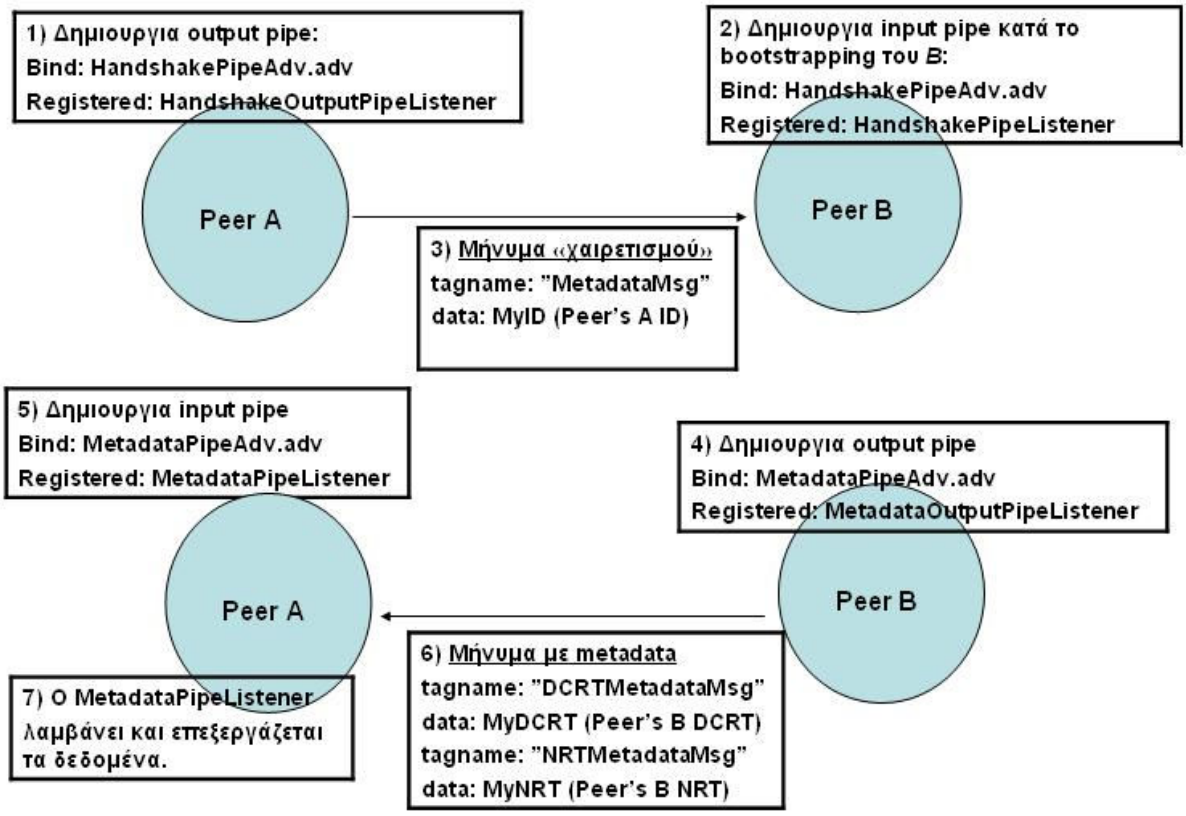
4) Ο Peer B καταλαβαίνει από το tagname του μηνύματος ότι πρέπει να στείλει τα metadata του στον Peer A . Έτσι δημιουργεί μια output pipe που είναι bind στην MetadataPipeAdv και κάνει register τον MetadataOutputPipeListener.

5) Ο Peer A, ακριβώς μετά το πρώτο βήμα, δημιουργεί μια input pipe που είναι bind στην MetadataPipeAdv και κάνει register τον MetadataPipeListener.

6) Ο δίαυλος επικοινωνίας μεταξύ του A και του B εγκαθίσταται για να στείλει ο MetadataOutputPipeListener τους πίνακες στον Peer A (του οποίου το ID το γνωρίζει από το μήνυμα "χαιρετισμού") σε μορφή μηνύματος:

Tagname:	*data:*
"DCRTMetadataMsg "	*" "Action" "0" "Documentary" "1" "*
"NRTMetadataMsg "	*" "0" "urn:jxta:uuid-9616261646..." "urn:jxta:uuid-63AF6261007..." "q" "1" "urn:jxta:uuid-49600B6F646..." "q" "*

7) Ο MetadataPipeListener "ξυπνάει" όταν θα δεχθεί το μήνυμα από τον Peer B για να το επεξεργαστεί: Εξάγει τα data από το 1ο String Element, το "DCRTMetadataMsg", για να τα εισάγει στον δικό του πίνακα DCRT που είναι άδειος, και τα data από το 2ο String Element, το "NRTMetadataMsg" , για να γεμίσει τον NRT του που είναι επίσης άδειος.



Σχήμα 18: Υλοποίηση RR.

Αναγκαιότητα της "χειραψίας":
Ένας peer για να λάβει δεδομένα πρέπει να ακολουθήσει τους κανόνες της "χειραψίας".Πρέπει, δηλαδή, να στείλει ένα μήνυμα "χαιρετισμού" στον αποστολέα και αυτός με την σειρά του να του επιστρέψει τα δεδομένα. Για μια φορά που ο peer θέλει να λάβει δεδομένα δημιουργούνται δυο unicast pipes, το οποίο σημαίνει επιπλέον χρόνος. Δυστυχώς, ο sender δεν θα μπορούσε να στείλει τα δεδομένα κατευθείαν, γιατί το μήνυμα "χαιρετισμού" περιέχει το ID του requested peer, το οποίο χρειάζεται στην επικοινωνία των peers.

4.4.2 - Υλοποίηση GP

Όταν ένας peer θέλει να ενημερώσει τα μέλη ενός peer group *pg* (Cluster ή Leader PeerGroup) για κάποια αλλαγή που συνέβη στο σύστημα ή για κάποια χαρακτηριστικά που αφορούν αυτόν ή ένα ολόκληρο peer group, μεταδίδει ένα *"guerilla message"* στο *pg* χωρίς να περιμένει κάποια ανταπόκριση. Άρα, πρέπει να εξασφαλιστεί ότι κάθε μέλος του *pg* λαμβάνει τα μεταδιδόμενα μηνύματα που στοχεύουν σε αυτό.

Όταν ένας peer γίνει μέλος είτε σε έναν Cluster PeerGroup επειδή κάνει publish ένα αρχείο σε αυτόν, είτε στο Leader PeerGroup επειδή ψηφίζεται leader σε έναν cluster, εκτελεί το *ReceivePropagatedMsgService* ώστε να "ακούει" για όλα τα *propagated guerilla messages* που αφορούν το συγκεκριμένο peergroup. Το service αυτό κάνει register έναν listener, τον *PropagateListener*, ο οποίος "ξυπνάει" όταν ένα propagated message ληφθεί για να εξάγει τα δεδομένα του και βάσει αυτών να ενημερώσει τα metadata του.

Ένα propagated message αποτελείται από ένα *string element* που περιέχει το πεδίο tagname, το οποίο υποδεικνύει στον *PropagateListener* το είδος της ενημέρωσης που θέλει να κάνει ο *propagator peer*, και το πεδίο data που περιέχει τα πραγματικά δεδομένα. Τα δεδομένα που αποθηκεύει το κάθε *propagated message* φαίνονται στον Πίνακα VII του Παραρτήματος.

Οι περιπτώσεις κατά τις οποίες χρησιμοποιείται το GP είναι οι εξής:

- Κατά την εκλογή του αρχηγού σε έναν cluster.

Ένας peer μεταδίδει ένα μήνυμα τύπου *"PowerMsg"* σε κάθε cluster στον οποίο ανήκει με σκοπό την εκλογή του αρχηγού.
Ο *PropagateListener* ενός peer που λαμβάνει το μήνυμα καταχωρεί το entry [*Propagator's ID, Propagator's Total Power*] που εξάγει από το μήνυμα στον *PowerTable* του για να εκλέξει αργότερα τον αρχηγό του cluster *ClusterNo* (δεύτερη μεταβλητή στο *"PowerMsg"*).

- Κατά την μετάδοση των χαρακτηριστικών (normalized popularity και $p(S_i)$) ενός cluster στους leaders.

Ένας leader μεταδίδει ένα μήνυμα τύπου *"ClusterLoadFiguresMsg"* για να ενημερώσει τους υπόλοιπους leaders για τα χαρακτηριστικά του cluster του.

Ένας leader μπορεί, επιπλέον, μεταδίδοντας τα χαρακτηριστικά του cluster του να δώσει εντολή στους παραλήπτες του μηνύματος να τον εκλέξουν ως chosen leader στέλνοντας τα ίδια δεδομένα που περιλαμβάνει ένα *"ClusterLoadFiguresMsg",* αλλά διαφοροποιώντας το tagname σε *"VoteMeMsg"*.

Ο αλγόριθμος που χρησιμοποιείται για την εκλογή του chosen leader είναι μια παραλλαγή του *bully election* και περιγράφεται παρακάτω:

Αποστολέας:
Όταν ένας leader *propagator_lead* έχει συλλέξει τα load-figures και έχει υπολογίσει την τιμή του *Normalized Popularity* του cluster c_p στον οποίο είναι αρχηγός (*ClusterMonitoringService*) ξεκινάει την εκτέλεση του *LeaderCommunicationService*. Πριν από την εκτέλεση του παραλλαγμένου *bully election*, ο leader εισάγει τα χαρακτηριστικά του c_p στον *NormalizedPopularityTable*. Ακριβώς μετά, ξεκινάει η εκτέλεση του αλγορίθμου με τον *propagator_lead* να εξετάζει αν υπάρχει κάποιος cluster στις καταχωρήσεις του πίνακα με μεγαλύτερο *Normalized Popularity* από τον c_p.
-Αν δεν υπάρχει, μεταδίδει ένα *"VoteMeMsg"* σε όλο το Leader PeerGroup και ανακηρύσσει τον εαυτό του ως chosen leader.
-Αν υπάρχει, μεταδίδει ένα *"ClusterLoadFiguresMsg"* σε όλο το Leader PeerGroup.

```
Insert my Normalized Popularity and p(Si) in NormalizedPopularityTable;
Run BullyElection();

BullyElection()
{
```

```
        boolean HigherNormPopPeers = FindHigherNormPopPeers();

        if ( HigherNormPopPeers == false )
        {
                Set me as Chosen Leader;
                Propagate in Leader PeerGroup a "VoteMeMsg";
        }

        else
                Propagate in Leader PeerGroup a "ClusterLoadFiguresMsg";
}

FindHigherNormPopPeers()
{
        Check in NormalizedPopularityTable if clusters with higher
         normalized Popularity than mine exist;

        if do exist
                return true;
        else
                return false;
}
```

Παραλήπτης:
Αν ο leader *receiver_lead* του cluster c_r παραλάβει μέσω του *PropagateListener* ένα μήνυμα που μεταδόθηκε στο Leader PeerGroup από τον leader *propagator_lead* του cluster c_p, εξάγει το περιεχόμενο του tagname του μηνύματος και πράττει αναλόγως:
-Αν το tagname του μηνύματος ισούται με *"VoteMeMsg"*, καταχωρεί στον *NormalizedPopularityTable* τα χαρακτηριστικά του cluster c_p και εξετάζει αν το *Normalized Popularity* του c_p είναι μεγαλύτερο από του c_r. Σε περίπτωση που είναι μεγαλύτερο, αναγνωρίζει ως chosen leader τον *propagator_lead*. Αν είναι μικρότερο, ξανατρέχει τον παραλλαγμένο αλγόριθμο από την αρχή γιατί ο c_p δεν είναι ο cluster με το μεγαλύτερο *Normalized Popularity*.
-Αν το tagname του μηνύματος ισούται με *"ClusterLoadFiguresMsg"*, απλά καταχωρεί τα χαρακτηριστικά του cluster c_p στον *NormalizedPopularityTable*.

```
PropagateListener grabs the message;
Extract message's tagname;
```

```
if( tagname == "VoteMeMsg")
{
        Extract data from "VoteMeMsg";
        Convert data into an array array_data;
        array_data= [Cluster LeaderID NormalizedPopularity pSi ];
        Insert data in NormalizedPopularityTable;

        if( array_data[2] > My Normalized Popularity )
                Set peer with ID array_data[1]  as Chosen Leader;
        else
                Run BullyElection();
}

if( tagname == "ClusterLoadFiguresMsg")
{
        Extract data from " ClusterLoadFiguresMsg ";
        Insert data in NormalizedPopularityTable;
}
```

- Κατά την μεταφορά μιας σημασιολογικής κατηγορίας από έναν cluster c_s σε έναν άλλον c_d.

Ο chosen leader εκτελώντας το *LazyRebalancingService,* το οποίο χρησιμοποιεί το *PropagateMsgService*, μεταδίδει σε όλους τους peers του c_s και του c_d ένα μήνυμα τύπου *"AssignMsg"*.

- Κατά την αποχώρηση ενός peer p_d από το σύστημα.

Ο p_d, κατά την έξοδο του από το Guerilla Sharing System, μεταδίδει σε κάθε cluster που ανήκει ένα μήνυμα τύπου *"DepartureMsg"*.

Ο peer που θα λάβει το μήνυμα εξάγει το ID του p_d από το *"DepartureMsg"* για να τον σβήσει από την τοπική του cache , από τον *NRT*, από τον *PowerTable* και από τον *ClusterLoadFiguresTable*, και να ξεκινήσει εκλογές στον cluster *ClusterNo* (δεύτερη μεταβλητή στο *"PowerMsg"*) σε περίπτωση που ο p_d είναι αρχηγός σε αυτόν.

4.4.3 - Υλοποίηση GPR

Ο τρίτος τρόπος επικοινωνίας συνδυάζει το *PropagateProtocol* με το *ExchangeDataProtocol*. Χρησιμοποιείται όταν ένας peer θέλει να ζητήσει δεδομένα από όλους τους peers ενός peer group (*Cluster* ή *Leader* PeerGroup) και οι peers αποκρίνονται επιστρέφοντας τα δεδομένα σε ένα *guerilla message*.

ένας peer για να μεταδώσει το *requested message* χρησιμοποιεί την *PropagateMsgService* του *PropagateProtocol* σε ένα peer group.
Αμέσως μετά δημιουργεί μια input pipe που είναι bind σε μια pipe advertisement *p_adv* και κάνει register έναν input pipe listener, έστω τον *in_list*. Ο *propagator peer* περιμένει να δεχτεί μέσω αυτής της pipe τα δεδομένα που ζήτησε με τη μετάδοση του μηνύματος, ώστε να τα επεξεργαστεί ο *in_list*. Ποιος input pipe listener γίνεται register και ποια pipe advertisement κάνει bind ο *propagator peer* εξαρτάται από τα δεδομένα που ζητάει, δηλαδή από το tagname του *propagated message*:

Tagname:	*p_adv:*	*in_list:*
" PublishMsg"	*TrueClusterPipeAdv*	*TrueClusterPipeListener*
"LoadFiguresRequestMsg"	*LoadFiguresPipeAdv*	*LoadFiguresPipeListener*

Ένα *propagated message* που στέλνεται από έναν peer για να ζητήσει κάποια επιθυμητά δεδομένα από ένα σύνολο peers, αποτελείται από ένα *string element*, το οποίο περιέχει το πεδίο *tagname*, που υποδεικνύει στον *PropagateListener* του παραλήπτη το είδος των δεδομένων που ζήτησε ο *propagator peer*, και το πεδίο *data* που περιέχει τα πραγματικά δεδομένα. Στον Πίνακα VIII του Παραρτήματος φαίνονται οι τιμές του πεδίου *data* για κάθε *propagated request message*.

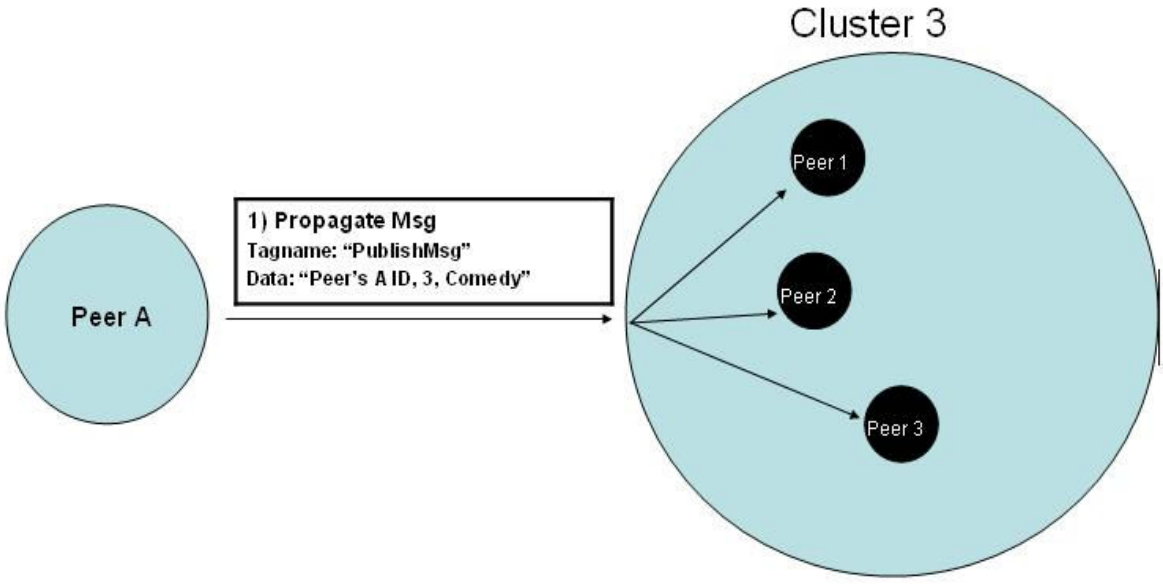
Όταν ο *PropagateListener* του παραλήπτη λάβει το *propagated message*, εξάγει το *tagname* του μηνύματος, δημιουργεί μια output pipe προσανατολισμένη στον *requested peer*, η οποία είναι bind στην pipe advertisement *p_adv*, και κάνει register έναν output pipe listener, έστω τον *out_list*, για να στείλει πίσω την απόκριση. Ποιος θα είναι συγκεκριμένα ο *out_list* και η *p_adv* εξαρτάται από τα δεδομένα που ζήτησε ο *propagator peer*, δηλαδή από το *tagname* του *propagated message*:

Tagname:	*p_adv:*	*out_list:*
"TrueClusterMsg"	*TrueClusterPipeAdv*	*TrueClusterOutputPipeListener*
"LoadFiguresMsg"	*LoadFiguresPipeAdv*	*LoadFiguresOutputPipeListener*

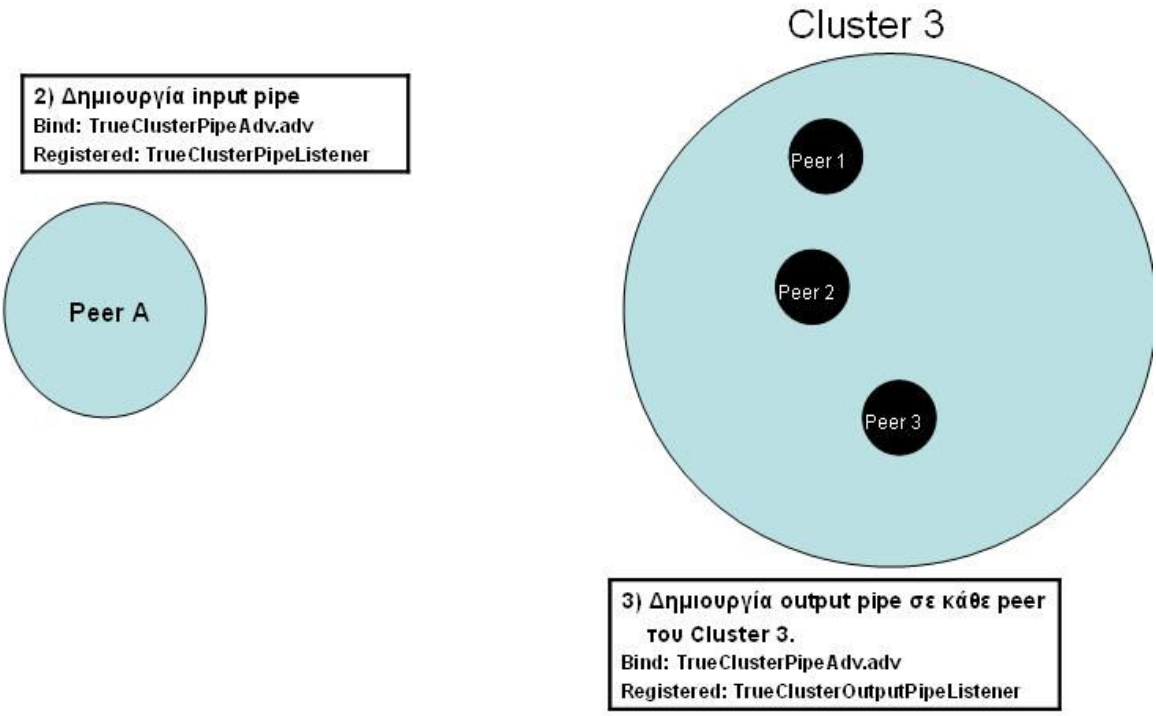
Έτσι δημιουργείται μια pipe μεταξύ των δυο peers, εφόσον είναι και οι δύο bind στην ίδια pipe advertisement *p_adv*, και ο *out_list* στέλνει τα επιθυμητά δεδομένα στον *propagator peer* για να τα λάβει ο *in_list*.

Τα δεδομένα των μηνύματος απόκρισης φαίνονται στον πίνακα VIII του Παραρτήματος.

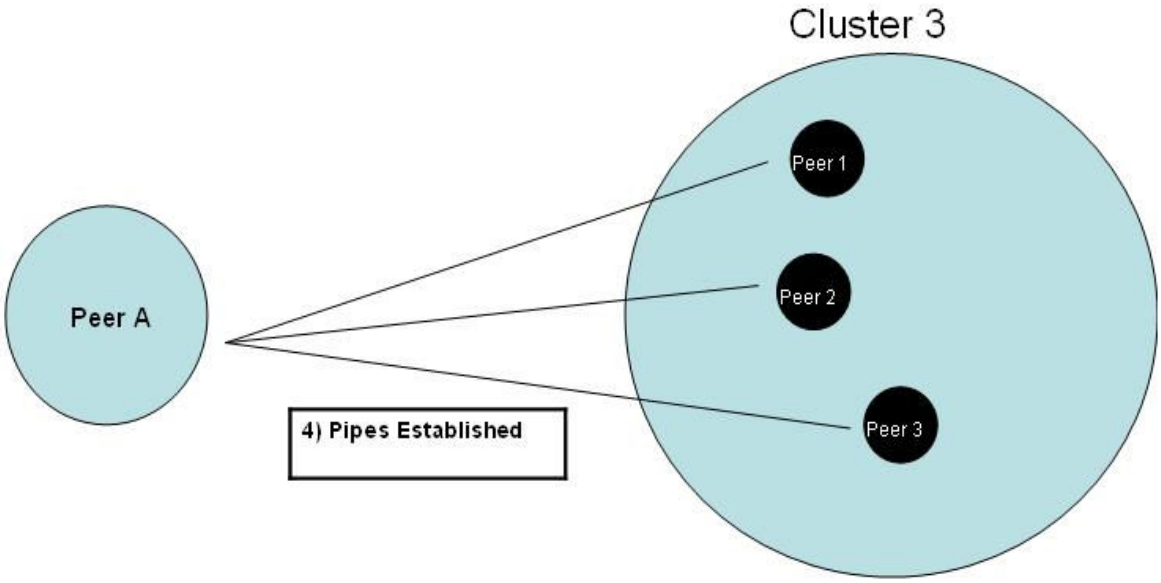
Παράδειγμα: *Στο παρακάτω σχήμα μπορούμε να δούμε τα βήματα που χρειάζεται ένας peer για να εκδώσει ένα αρχείο κατηγορίας comedy που σύμφωνα με τον DCRT του φιλοξενείται στον cluster 3:*



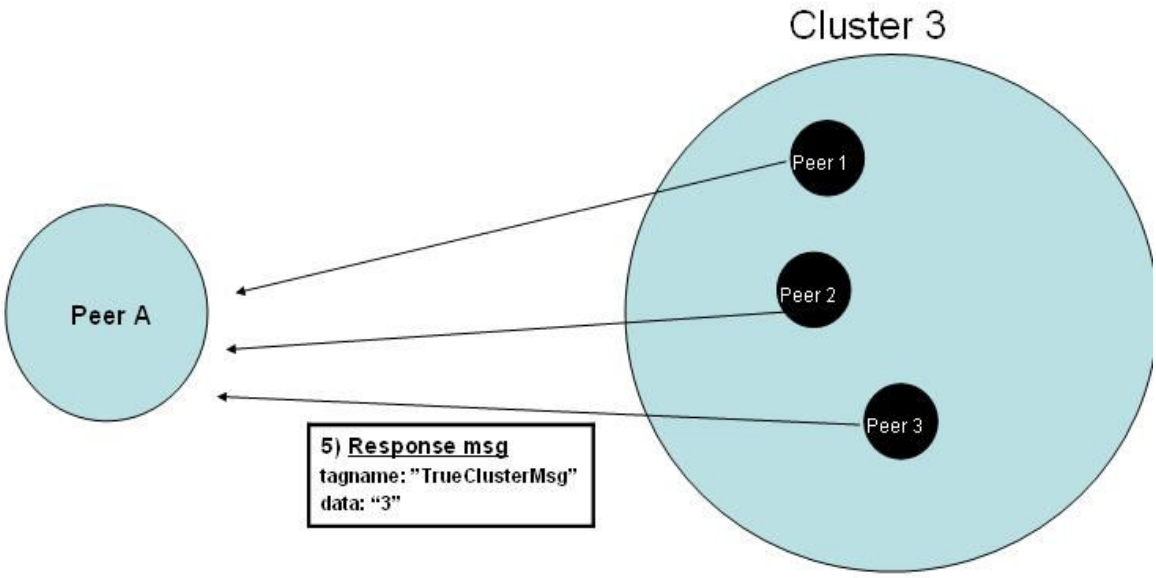
1) Ο peer A μεταδίδει ένα message τύπου "PublishMsg" στους peers του cluster 3.



2) Ο Peer A εγκαθιστά μια input pipe που είναι bind στην TrueClusterPipeAdv και κάνει register τον TrueClusterPipeListener.
3) Κάθε peer του cluster 3 εγκαθιστά μια output pipe που είναι bind στην TrueClusterPipeAdv και κάνει register τον TrueClusterOutputPipeListener.



4) Εφόσον η input pipe είναι bind στην ίδια pipe advertisement TrueClusterPipeAdv με τις output pipes δημιουργείται μια pipe για κάθε peer του cluster 3, μεταξύ αυτού και του Peer A.



5) Ο TrueClusterOutputPipeListener του κάθε peer που ανήκει στον cluster 3 στέλνει απάντηση στον Peer A ότι η κατηγορία comedy φιλοξενείται όντως από τον cluster 3.

4.4.4 - Αξιολόγηση των Μεθόδων Επικοινωνίας

Στο παρών κεφάλαιο δίνονται τα ποσοστά επιτυχίας για κάθε μέθοδο επικοινωνίας, δηλαδή οι πιθανότητες ένας peer να στείλει ή να λάβει επιτυχώς κάποια επιθυμητά δεδομένα αναλόγως με τον τρόπο επικοινωνίας που χρησιμοποιεί. Επιπλέον, φαίνονται οι χρόνοι που χρειάζεται ο peer για να διατελέσει μια τέτοια συνδιαλλαγή κάτω από ποικίλα σενάρια.. Όλα τα πειράματα διεξήχθησαν στο τοπικό δίκτυο υπολογιστών (LAN) του εργαστηρίου Softlab του Πολυτεχνείου Κρήτης.

- RR:

Ως *total_time* ορίζουμε το χρόνο που χρειάζεται ένας peer για να ανακτήσει δεδομένα χρησιμοποιώντας μόνο unicast pipes. Το *total_time* είναι το άθροισμα

του χρόνου που χρειάζεται ο peer για να στείλει το μήνυμα "χαιρετισμού" μέχρι την στιγμή που θα παραλάβει τα επιθυμητά δεδομένα. Στους παρακάτω πίνακες παρατηρούμε τις τιμές του *total_time* για 54 διαφορετικά πειράματα. Όλοι οι χρόνοι είναι μετρημένοι σε milliseconds.

total_time	*total_time*
1048	1310
1071	1312
1074	1312
1091	1342
1092	1350
1093	1360
1095	1372
1096	1382
1099	1384
1103	1395
1112	1398
1124	1401
1171	1402
1172	1421
1175	1423
1176	1444
1241	1446
1249	1452
1250	1462
1255	1565
1262	1723
1277	1724
1280	1726
1282	1822
1292	1823
1302	1901
1309	1943

Μέσος Όρος: 1340 msecs

Επιπλέον, μετρήθηκε ότι για το RR, από τις 112 προσπάθειες που κάνει ένας peer για να στείλει ένα μήνυμα "χαιρετισμού" και να να λάβει κάποια δεδομένα, οι 103 είναι επιτυχείς (ποσοστό 91, 96%). Επειδή οι one-way JXTA pipes δεν εγγυώνται αξιόπιστη μεταφορά ενός μηνύματος, μπορεί είτε το μήνυμα "χαιρετισμού", είτε τα πραγματικά δεδομένα να μην φτάσουν ποτέ στον προορισμό τους.

- GP:

Παρακάτω, παραθέτονται διάφοροι πίνακες με τους χρόνους που μετρήθηκαν στα πειράματα για την μετάδοση ενός μηνύματος σε ένα peer group (σε κάποιον Cluster ή στο Leader PeerGroup). Οι μετρήσεις πάρθηκαν για peer groups που αποτελούνταν μέχρι 8 peers και δεν ήταν δυνατή η διεξαγωγή πειραμάτων για περισσότερους peers λόγω περιορισμένου αριθμού υπολογιστών στο εργαστήριο. Όλοι οι χρόνοι είναι μετρημένοι σε milliseconds. Κάθε στήλη υποδεικνύει τους συνολικούς χρόνους που χρειάζεται για να μεταδοθεί το μήνυμα σε όλους τους peers του εκάστοτε peer group.

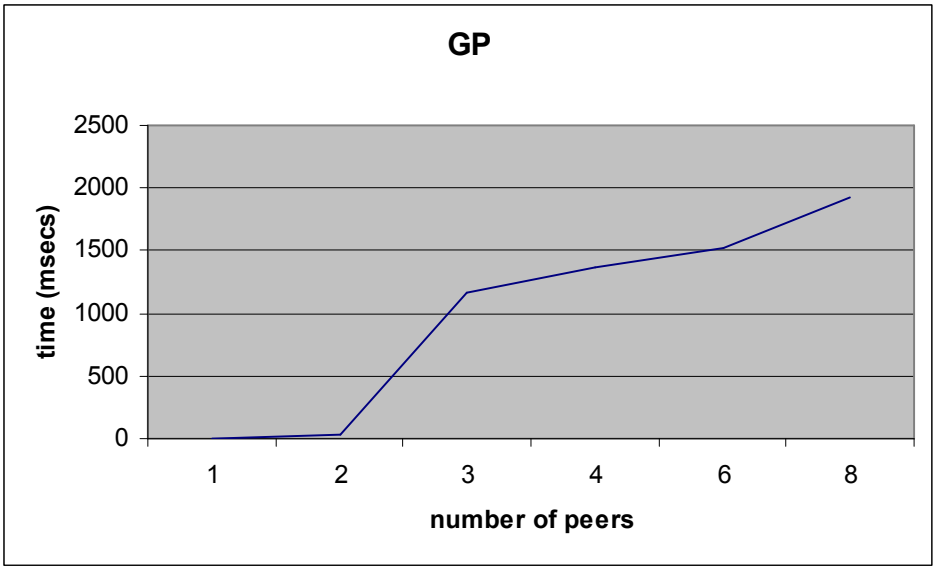
- *Χρόνοι Μετάδοσης μηνύματος σε PeerGroup με:*

1 μέλος	*2 μέλη*	*3 μέλη*	*4 μέλη*	*6 μέλη*	*8 μέλη*
5	8	79	9	27	30
5	8	80	20	57	102
5	8	100	24	58	205
5	9	105	34	59	242
5	9	158	86	75	284
6	9	180	99	171	295
6	9	209	121	208	303
6	10	261	138	263	398
6	11	273	185	408	402
6	11	441	256	439	567
6	11	490	286	447	941
6	12	601	298	533	1006
6	13	641	303	638	1270
6	13	712	328	722	1274
6	14	730	344	932	1558
6	16	731	458	986	1601
6	16	801	473	1142	1604
6	16	841	625	1300	1803
6	17	871	896	1502	1850
6	18	892	1018	1557	1956
6	18	1082	1202	1562	1989
7	19	1089	1314	1598	1992
7	20	1117	1330	1603	2001

7	20	1140	1366	1611	2009
7	20	1151	1398	1633	2013
7	20	1159	1409	1678	2108
7	21	1202	1500	1764	2301
7	21	1272	1685	1799	2445
7	22	1324	1733	1873	2502
7	22	1362	2222	2054	2565
7	23	1429	2297	2162	2677
8	23	1702	2320	2259	2702
8	25	1712	2695	2294	3169
8	29	2057	2794	2401	3200
8	30	2093	3081	2678	3508
8	31	2100	3331	2844	4001
9	34	2814	3844	3702	4128
10	37	3566	4002	3745	4101
11	190	3979	4380	4994	4232
20	207	3994	4996	5001	5567
Μέσος Όρος	7,025	26,75	1163,5	1372,5	1519,475	1922,525

Σε όλα τα πειράματα το μήνυμα του propagator μεταδιδόταν σε όλους τους peers του εκάστοτε peer group.

Η παρακάτω γραφική παράσταση δείχνει τους μέσους χρόνους που χρειάζεται για να ολοκληρωθεί ένα propagation, δηλαδή για να μεταδοθεί ένα μήνυμα σε όλους τους peers ενός peer group, σε σχέση με των αριθμό των μελών του peer group:



- GPR:

Οι παρακάτω πίνακες υποδεικνύουν τους χρόνους που μετρήθηκαν στα πειράματα για να συλλέξει ένας peer πληροφορίες από όλα τα μέλη ενός peer group (κάποιου Cluster ή του Leader PeerGroup) χρησιμοποιώντας το GPR. Όπως και προηγουμένως οι μετρήσεις πάρθηκαν για peer groups που αποτελούνταν μέχρι και 8 peers. Όλοι οι χρόνοι είναι μετρημένοι σε milliseconds.

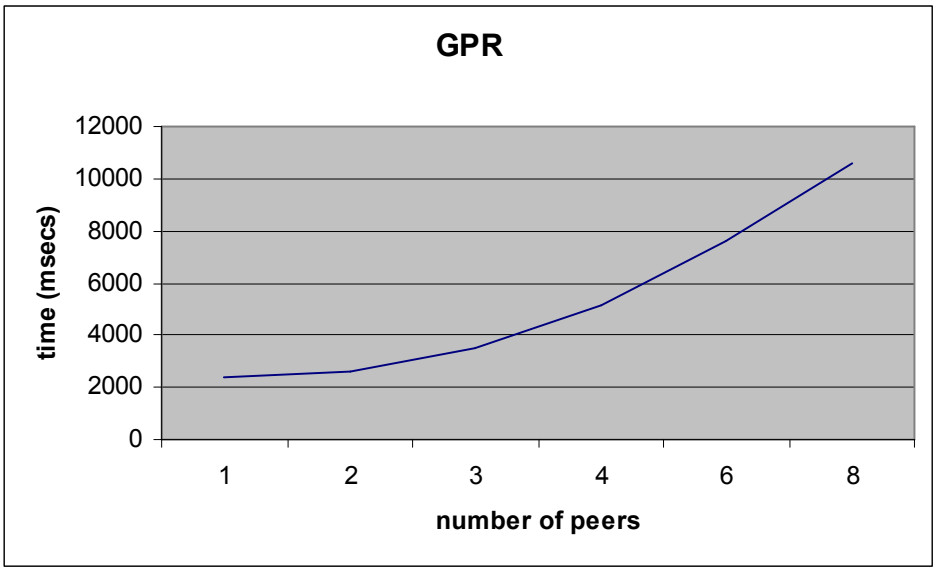
- *Χρόνοι Ανάκτησης Δεδομένων από PeerGroup με:*

1 μέλος	*2 μέλη*	*3 μέλη*	*4 μέλη*	*6 μέλη*	*8 μέλη*
401	851	1272	6622	3288	3252
442	987	1087	10779	4383	6625
604	1001	1292	5776	5142	6717
690	1131	1345	4921	6001	7107
932	1302	1653	5128	6010	7341
1090	1735	1913	3130	6013	7601
1152	1742	2003	5124	6032	7656
1214	1823	2013	5010	6132	7690
1222	1873	2109	7988	6276	7765
1368	1973	2198	4954	6295	7813
1403	2002	2300	3067	6578	7915
1442	2033	2341	3210	6723	8022

1462	2153	2390	5716	6807	8101
1633	2233	2493	4089	6904	8714
1908	2303	2804	7114	6951	8910
2013	2324	2844	4216	6971	9157
2045	2369	2898	4645	7004	9363
2092	2444	3157	6515	7114	9594
2100	2584	3175	2796	7172	9757
2274	2674	3521	7163	7214	9841
2284	2694	3565	3397	7412	10500
2293	2703	3735	8824	7413	10641
2294	2704	3865	4708	7419	10659
2343	2788	3898	5470	7500	10771
2473	2794	3908	5129	7578	10832
2543	2896	3985	3562	7632	11091
2596	2920	4036	3100	7634	11525
2709	3174	4096	4900	7865	12638
3114	3189	4106	5230	8053	13049
3247	3295	4226	5561	8130	13251
3255	3375	4236	3365	8418	13375
3384	3475	4329	4148	9013	13703
3410	3498	4867	5612	9030	14084
3663	3645	5067	3389	9125	14309
4323	3665	5097	6908	9828	14880
4407	3855	5138	2365	9919	15308
4411	3866	5267	7864	10760	15336
4890	3906	5548	3436	10831	15774
4991	3934	6608	4532	11444	16007
5478	5428	8993	5502	12715	16936
Μέσος Όρος	2389,875	2633,525	3484,45	5124,125	7568,225	10590,25
Ποσοστά Επιτυχίας	94%	90%	81%	80%	76%	72%

Επιτυχής ανάκτηση δεδομένων θεωρείται όταν ο propagator peer παραλαμβάνει δεδομένα από όλους τους peers του εκάστοτε peer group. Η αποτυχία του να λάβει ο αιτών peer δεδομένα από όλους τους peers οφείλεται στην αναξιοπιστία των one-way JXTA pipes.

Η παρακάτω γραφική παράσταση δείχνει τους μέσους χρόνους που χρειάζεται ένας peer για να παραλάβει δεδομένα από όλα τα μέλη ενός peer group, σε σχέση με τον αριθμό των μελών του.



Κεφάλαιο 5

ΣΥΜΠΕΡΑΣΜΑΤΑ

Ο σκοπός της συγκεκριμένης διπλωματικής εργασίας ήταν η ανάπτυξη μηχανισμών για peer-to-peer επικοινωνία, οι οποίοι θα διευκολύνουν την ανάπτυξη πολύπλοκων P2P εφαρμογών. Πάνω σε αυτούς τους μηχανισμούς υλοποιήθηκε ένα σύστημα ανταλλαγής αρχείων, το Guerilla, το οποίο είναι βασισμένο στο Ξ.Κ.Τ.Χ Project. Οι μηχανισμοί επικοινωνίας και το P2P Sharing System αναπτύχθηκε πάνω στην πλατφόρμα JXTA, το οποίο παρέχει βασικές συναρτήσεις απαραίτητες για ένα peer-to-peer δίκτυο.

Για την ανάπτυξη του Guerilla Sharing System σχεδιάστηκαν και υλοποιήθηκαν τα Guerilla components τα οποία "χτίστηκαν" πάνω στα JXTA components. Τα στοιχεία αυτά είναι τα εξής:
i) Ο server του Guerilla Sharing System που αποθηκεύει τις διευθύνσεις των ήδη συνδεδεμένων peer στο σύστημα και JXTA advertisements που αναπαριστούν κάποιους πόρους του δικτύου.
ii) Το Guerilla, τα Cluster και το Leader peer group.
iii) Απλοί JXTA peers, που είναι edge ή rendezvous σε κάποιον cluster που έχουν εκδώσει αρχεία, και JXTA super-peers leaders, που είναι αναγκαστικά rendezvous εφόσον ανήκουν στο Leader PeerGroup.
iv) Τα Guerilla Protocols, που είναι βασισμένα στα JXTA Protocols και εξασφαλίζουν το bootstrapping του peer και την περαιτέρω κανονική του λειτουργία.
v) Listeners που ξυπνάνε ασύγχρονα κατά την αποστολή ή την λήψη των messages που σχετίζονται με την λειτουργικότητα των πρωτοκόλλων του Guerilla Sharing System και listeners που ξυπνάνε ασύγχρονα κατά την ανακάλυψη κάποιου resource του Guerilla network (απλοί peers, rendezvous peers ή peergroups).
vi) Τα metadata, δηλαδή data structures αποθηκευμένα σε κάθε peer για να επικοινωνεί και να ανταλλάσσει πληροφορίες με άλλους ή να βρίσκει peers που έχουν αποθηκευμένες επιθυμητές πληροφορίες.

vii) Τα Guerilla messages, δηλαδή συγκεκριμένου τύπου JXTA messages που οι peers στέλνουν μεταξύ τους για να επιτύχουν κάποιες λειτουργικότητες.
Όλα τα πρωτόκολλα του Guerilla Sharing System είναι βασισμένα στην ανταλλαγή μηνυμάτων μεταξύ των peers.

Τα Guerilla Protocols διακρίνονται στα πρωτόκολλα επικοινωνίας, που υλοποιούν συνολικά τρεις διαφορετικούς τρόπους επικοινωνίας μεταξύ των peers (*RR:Response-Request, GP:Group Propagation, GPR:Group Propagation Response*), και στα πρωτόκολλα που συνθέτουν το Guerilla Sharing System και υλοποιούν την λειτουργικότητα αποστολής queries και έκδοσης εικονικών αρχείων, και τον μηχανισμό προσαρμογής του συστηματος σε ένα δυναμικό περιβάλλον για την διατήρηση του inter-cluster load balancing.

Οι μηχανισμοί επικοινωνίας μπορούν να χρησιμοποιηθούν για την υλοποίηση πολλών εφαρμογών που απαιτούν point-to-point επικοινωνία (*RR*), αποστολή δεδομένων σε μια κοινότητα από peers (*GP*) ή ανάκτηση δεδομένων από πολλούς peers (*GPR*). Αυτοί οι μέθοδοι όμως απαιτούν γνώση κάποιων πολύ βασικών JXTA στοιχείων. Οπότε, μια ιδέα για το μέλλον θα ήταν αν οι μηχανισμοί επικοινωνίας γινόντουσαν ακόμα πιο αφαιρετικοί και απλοί, ώστε ο προγραμματιστής που θα αναπτύσσει εφαρμογές πάνω σε αυτούς να μην χρειάζεται να γνωρίζει καν τα JXTA components, παρά μόνο απλές αφηρημένες P2P έννοιες. Επιπλέον, θα μπορούσε να γίνει βελτιστοποίηση των μηχανισμών επικοινωνίας με σκοπό την διεξαγωγή πειραμάτων μεγάλης κλίμακας (πχ σε δίκτυα MAN ή WAN). Μια ακόμη πιο ενδιαφέρουσα ιδέα θα ήταν η περαιτέρω ανάπτυξη των μηχανισμών επικοινωνίας με τέτοιο τρόπο ώστε να αποκρύβεται η δυναμική συμπεριφορά ενός P2P δικτύου και επιτέλους ο προγραμματιστής ενός τέτοιου χαοτικού συστηματος να μπορεί να χαμογελάσει και να επικοινωνήσει ξανά με τον κόσμο που έχασε.

ΠΑΡΑΡΤΗΜΑ

Πίνακας Ι – Λειτουργικότητα Πρωτοκόλλων

Πρωτόκολλα:	*Λειτουργικότητα:*
DynamicAdaptationProtocol	Εγγυείται το fair load distribution σε ένα δυναμικό περιβάλλον.
ExchangeDataProtocol	Αποστολή/παραλαβή guerilla messages μέσω unicast pipes.
JoinProtocol	Υπεύθυνο για την αρχική ένταξη του peer στο Guerilla PeerGroup.
PropagateProtocol	Αποστολή/ παραλαβή propagated guerilla messages.
PublishProtocol	Δημοσίευση εικονικών αρχείων.
QueryProtocol	Διαχείριση των queries.

Πίνακας ΙΙ – Λειτουργικότητα Listeners

Listeners:	*Λειτουργικότητα:*
HandshakeOutputPipeListener	Στέλνει msg "χαιρετισμού".
HandshakePipeListener	Παραλαμβάνει msg "χαιρετισμού".
LoadFiguresOutputPipeListener	Στέλνει τα load figures του local peer.
LoadFiguresPipeListener	Παραλαμβάνει τα load figures ενός peer.
QueryOutputPipeListener	Στέλνει ένα query.
QueryPipeListener	Παραλαμβάνει ένα query response.
MetadataOutputPipeListener	Στέλνει τον DCRT και τον NRT του local peer.
MetadataPipeListener	Παραλαμβάνει τον DCRT και τον NRT ενός peer.
TrueClusterOutputPipeListener	Στέλνει το όνομα του cluster που αντιστοιχεί σε μια συγκεκριμένη σημασιολογική κατηγορία.

TrueClusterPipeListener	Παραλαμβάνει το όνομα του cluster που αντιστοιχεί σε μια συγκεκριμένη σημασιολογική κατηγορία
PropagateListener	Ακούει και επεξεργάζεται τα propagated msgs.
PeerGroupListener	Ακούει για αποκρίσεις σε αιτήσεις εύρεσης PeerGroup advertisements.
PeerListener	Ακούει για αποκρίσεις σε αιτήσεις εύρεσης Peer advertisements.
RdvListener	Ακούει για αποκρίσεις σε αιτήσεις εύρεσης Rendezvous advertisements.

Πίνακας ΙΙΙ – Λειτουργικότητα Metadata

Metadata	*Λειτουργικότητα*
DT	Αντιστοιχεί ονόματα documents σε σημασιολογικές κατηγορίες. Αποθηκεύεται σε κάθε peer.
DCRT	Αντιστοιχεί σημασιολογικές κατηγορίες σε clusters. Αποθηκεύεται σε κάθε peer.
NRT	Αντιστοιχεί clusters σε peers. Αποθηκεύεται σε κάθε peer.
PowerTable	Αντιστοιχεί peers σε μονάδες υπολογιστικής ισχύς. Αποθηκεύεται σε κάθε peer.
PopularityTable	Αντιστοιχεί ονόματα documents σε popularities Αποθηκεύεται σε κάθε peer.
NormalizedPopularityTable	Περιέχει το normalized popularity και την τιμή της $p(S_i)$ για κάθε cluster. Αποθηκεύεται σε κάθε leader.
ClusterLoadFiguresTable	Λίστα με τα load-figures των peers.

	Αποθηκεύεται σε κάθε leader.

Πίνακας IV - Λειτουργικότητα Messages

- Messages που στέλνονται μέσω unicast pipes:

Είδος Μηνύματος	*Tagname*	*Λειτουργικότητα*
Request Message	*"MetadataMsg"*	Στέλνεται από έναν peer κατά το bootstrapping του για να ζητήσει τους πίνακες DCRT και NRT από έναν ήδη συνδεδεμένο peer στο σύστημα. Χρησιμοποιείται από το JoinService του JoinProtocol.
Request Message	*"QueryMsg"*	Τα δεδομένα του μηνύματος αποτελούν το query που στέλνει ένας peer κατά την εκτέλεση του QueryService.
Response Message	*"DCRTMetadataMsg"* *"NRTMetadataMsg"*	Το συγκεκριμένο message αποτελείται από δυο πεδία, το πεδίο *"DCRTMetadataMsg"* που περιέχει τον πίνακα DCRT και το πεδίο *"NRTMetadataMsg"* που περιέχει τον πίνακα NRT. Στέλνεται σαν απόκριση σε ένα *"MetadataMsg"*.
Response Message	*"QueryResponseMsg"*	Στέλνεται ως απόκριση σε ένα query.
Response Message	*"LoadFiguresMsg"*:	Στέλνεται κατά το ClusterMonitoringService, όταν ο leader ενός cluster

		ζητάει τα load-figures των peers που ανήκουν σε αυτόν. Κάθε peer "πακετάρει" τα load-figures του σε ένα *"LoadFiguresMsg"*, το οποίο στέλνεται στον leader για να εισάγει τα δεδομένα του μηνύματος στον ClusterLoadFiguresTable.
Response Message	*"TrueClusterMsg"*	Όταν ένας peer θέλει να κάνει publish ένα αρχείο σημασιολογικής κατηγορίας *s*, συμβουλεύεται τον DCRT του για το όνομα του cluster c_i που φιλοξενεί την *s*. Όμως, αν το σύστημα πέρασε από ανακατατάξεις κατηγοριών (DynamicAdaptationProtocol) και ο peer δεν ενημερώθηκε για τις αλλαγές αυτές, ίσως ο c_i να μην φιλοξενεί την *s*. Έτσι, πριν εκδώσει το αρχείο συμβουλεύεται τους peers του c_i για το ποιος cluster φιλοξενεί, σύμφωνα με τα δικά τους metadata, την *s*. Οι peers απαντάνε στέλνοντας το όνομα του cluster αυτού σε ένα *"TrueClusterMsg"*.

- Messages που στέλνονται μέσω propagate pipes:

"PublishMsg"	Στέλνεται κατά το PublishProtocol για να γνωστοποιηθεί στα μέλη ενός cluster ότι ο αποστολέας του μηνύματος θέλει να εκδώσει σε αυτόν. Οι peers που λαμβάνουν το μήνυμα, είτε απαντάνε με ένα *"TrueClusterMsg"* αν ο αποστολέας μεταδίδει σε λάθος cluster, είτε καταχωρούν το νέο μέλος στον NRT τους αν ο αποστολέας μεταδίδει σε σωστό cluster.
"DepartureMsg"	Κατά την αποχώρηση του από το σύστημα, ένας peer μεταδίδει ένα *"DepartureMsg"* σε όλους τους clusters στους οποίους ανήκει. Οι peers που λαμβάνουν το μήνυμα τον σβήνουν από την τοπική τους cache , από τον NRT, από τον PowerTable και από τον ClusterLoadFiguresTable, και ξαναγίνονται εκλογές σε περίπτωση που αυτός είναι αρχηγός σε κάποιο cluster.
"PowerMsg"	Μεταδίδεται κατά την διάρκεια εκλογής αρχηγού (LeaderElectionService του DynamicAdaptationProtocol) από κάθε peer σε κάθε cluster στον οποίο ανήκει. Ουσιαστικά, το "PowerMsg" περιέχει τις μονάδες της υπολογιστικής ισχύς του αποστολέα, οι οποίες καταχωρούνται στα PowerTables των παραληπτών για να εκλεχθεί αργότερα ο αρχηγός.
"LoadFiguresRequestMsg"	Μεταδίδεται από κάθε leader στους peers του cluster που είναι αρχηγός (ClusterMonitoringService του

	DynamicAdaptationProtocol) για να ζητήσει τα load-figures τους. Απάντηση σε αυτό το request είναι το "*LoadFiguresMsg*" που στέλνει ο κάθε peer στο leader του.
"ClusterLoadFiguresMsg"	Κατά την διάρκεια του LeaderCommunicationService, κάθε leader μεταδίδει σε όλους τους υπόλοιπους leaders του LeaderPeerGroup τα χαρακτηριστικά του cluster του (normalized popularity και $p(S_i)$). Τα δεδομένα αυτά παραλαμβάνονται από τον PropagateListener και καταχωρούνται στο NormalizedPopularityTable του κάθε leader, ώστε βάσει αυτών ο εκλεκτός αρχηγός να υπολογίσει την τιμή του fairness index.
"VoteMeMsg"	Το *"VoteMeMsg"* εξυπηρετεί την ίδια λειτουργικότητα με το *"ClusterLoadFiguresMsg"*, αλλά επιπλέον δίνει εντολή στον παραλήπτη να εκλέξει ως chosen leader τον αποστολέα.
"AssignMsg"	Το LazyRebalancingService του DynamicAdaptationProtocol αναλαμβάνει να μεταφέρει μια κατηγορία *s* από έναν cluster c_s σε έναν άλλο c_d. Ο chosen leader πρέπει να ενημερώσει για την αλλαγή αυτή όλους τους peers των δυο clusters. Για να γίνει αυτό, μεταδίδει ένα *"AssignMsg"* στα μέλη των c_s και c_d, με το οποίο τους υποδεικνύει το όνομα της *s*, του c_s και του c_d ώστε να ενημερώσουν τα metadata τους για την αλλαγή αυτή.

Πίνακας V – Δεδομένα Μηνύματος "Χαιρετισμού"

Tagname	*"MetadataMsg"*
Data:	*"Receiver's PeerID"*

Tagname	*"QueryMsg"*
Data:	*"k1, k2... ki, m, Receiver's PeerID, ClusterNo, q"*

Οι μεταβλητές *k1, k2, ki* περιέχουν τα ονόματα των ζητούμενων αρχείων, η *m* τον αριθμό των επιθυμητών αποτελεσμάτων, η *ClusterNo* τον cluster στον οποίο ανήκουν τα ερωτούμενα αρχεία και το σύμβολο *q* υποδηλώνει το τέλος του query.

Πίνακας VI – Δεδομένα Response Msgs σε επικοινωνία τύπου RR

Request msg	*Response msg:*
"MetadataMsg"	Tagname: *"DCRTMetadataMsg"*
	Data: *"s_0, c_{s0}, s_1, c_{s1}...s_i, c_{si}"*
	Tagname: *"NRTMetadataMsg"*
	Data: *"c_0, N_{c0}, q, c_1, N_{c1}, q...c_i,N_{ci},q"*
"QueryMsg"	Tagname: *"QueryResponseMsg"*
	Data: "*k1, k3, k6, PeerName, PeerID "*

Αν ο *requested peer* στείλει ένα *request message* τύπου *"MetadataMsg",* ο *sender* στέλνει ένα *response message*, το οποίο αποτελείται από δυο *String Elements.* Το 1ο, που ονομάζεται *"DCRTMetadataMsg"*, περιέχει όλον τον πίνακα DCRT σε string format και το 2ο, που ονομάζεται *"NRTMetadataMsg"*, περιέχει όλον τον πίνακα NRT σε string format. Οι μεταβλητές s_0,s_1..s_i περιέχουν τα ονόματα όλων των σημασιολογικών κατηγοριών του συστήματος και η μεταβλητή c_{si} περιέχει τον αριθμό του cluster στην οποία ανήκει η κατηγόρια s_i. Παρομοίως, οι μεταβλητές c_0,c_1..c_i συμβολίζουν τα ονόματα όλων των clusters του συστήματος και η μεταβλητή N_{ci} περιέχει το σύνολο των peers που ανήκουν στον cluster c_i. Τα σύμβολα q υποδεικνύουν στον *MetadataPipeListener* το σημείο που τελειώνει κάθε entry και ξεκινάει το επόμενο (entry θεωρείται κάθε γραμμή του NRT).

Αν ο *requested peer* στείλει ένα *request message* τύπου *"QueryMsg",* ο *sender* στέλνει ένα *response message* τύπου *"QueryResponseMsg"*. Το μήνυμα αυτό είναι η απάντηση στο query του *requested peer* και περιέχει τα ζητούμενα ονόματα αρχείων *k1, k2...ki* και τα χαρακτηριστικά (*όνομα* και *ID*) του *sender*.

Πίνακας VII – Δεδομένα Propagated Msgs σε επικοινωνία τύπου GP

Tagname:	*"PowerMsg"*
Data:	*"Propagator's ID, ClusterNo, Propagator's Total Power"*

Η μεταβλητή *Propagator's ID* περιέχει το ID του peer που μεταδίδει το μήνυμα, η *Propagator's Total Power* τις μονάδες της υπολογιστικής ισχύς του και η *ClusterNo* τον αριθμό του cluster που γίνεται η μετάδοση.

Tagname:	*" ClusterLoadFiguresMsg "*
Data:	*"ClusterNo, MyPeerID, NormPopularity, pSi "*

Η μεταβλητή *ClusterNo* περιέχει τον αριθμό του cluster στον οποίο είναι leader ο peer που μεταδίδει το μήνυμα, η *MyPeerID* περιέχει το ID του, η *NormPopularity* την τιμή του Normalized Popularity του cluster *ClusterNo* και η *pSi* το συνολικό popularity του. Τα δεδομένα αυτά καταχωρούνται στο *NormalizedPopularityTable* του leader που λαμβάνει το μήνυμα ώστε βάσει αυτών ο εκλεκτός αρχηγός να υπολογίσει την τιμή του fairness index.

Tagname:	*"VoteMeMsg "*
Data:	*"ClusterNo, MyPeerID, NormPopularity, pSi "*

Βλέπε *"ClusterLoadFiguresMsg"*

Tagname:	*"AssignMsg"*
Data:	*"Category, DestCluster"*

Στην μεταβλητή *Category* εκχωρείται το όνομα της μεταφερόμενης σημασιολογικής κατηγορίας και στην μεταβλητή *DestCluster* ο αριθμός του

cluster στον οποίο γίνεται η μεταφορά. Έτσι οι peers του c_s και του c_d ενημερώνουν τον *DCRT* τους ότι η *Category* ανήκει πλέον στον *DestCluster*.

Tagname:	*"DepartureMsg"*
Data:	*"Peer's ID, ClusterNo"*

Η μεταβλητή *Peer's ID* περιέχει το ID του p_d και η *ClusterNo* τον αριθμό του cluster που γίνεται η μετάδοση.

Πίνακας VIII – Δεδομένα μηνυμάτων επικοινωνίας τύπου GPR

Propagated request message:

Tagname	*"PublishMsg"*
Data:	*"MyPeerID, ClusterNo, Category"*

Tagname	*"LoadFiguresRequestMsg"*
Data:	*"MyPeerID, ClusterNo"*

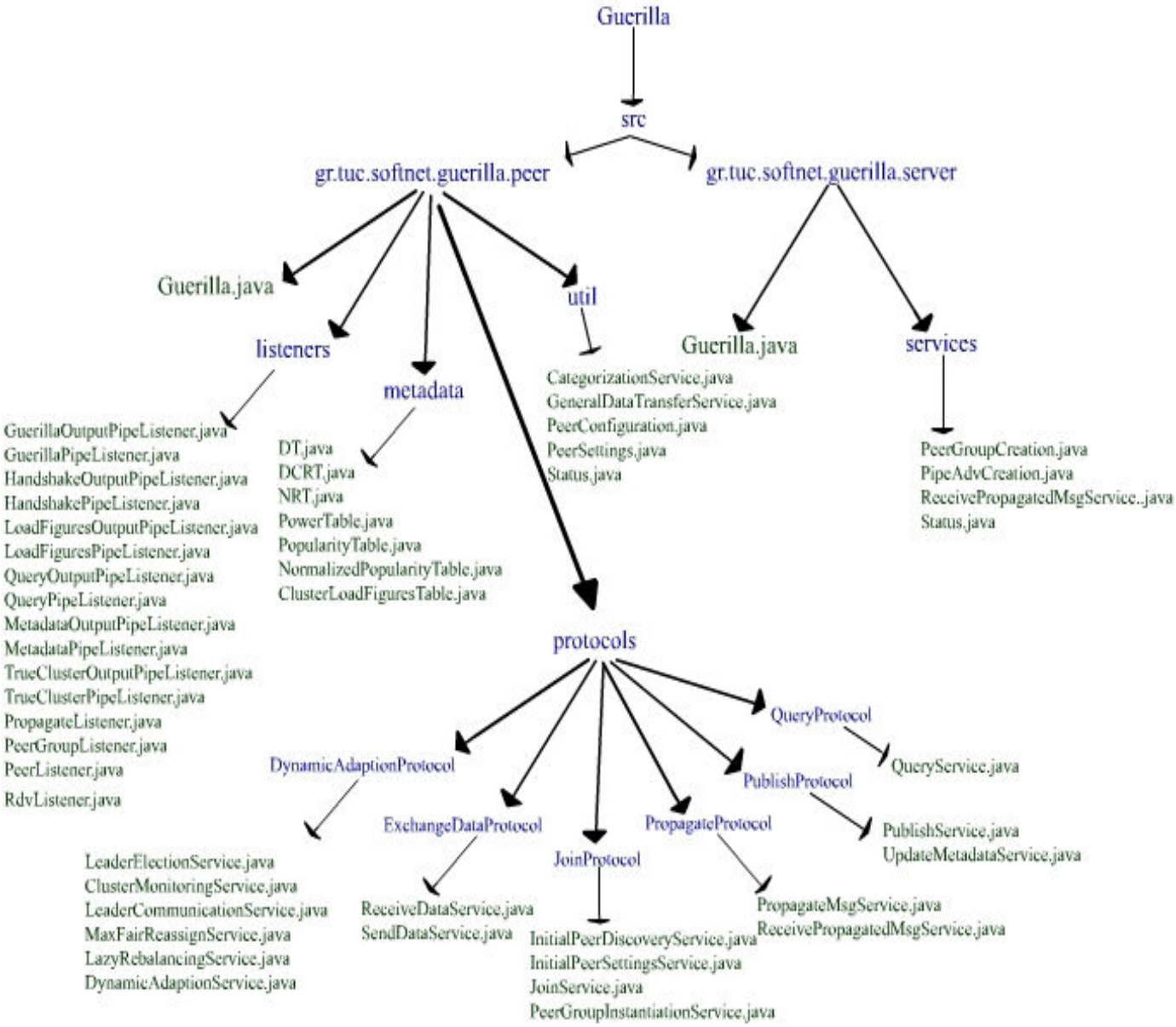
Η μεταβλητή *MyPeerID* περιέχει το ID του *propagator peer*, ώστε όταν θα λάβει το μήνυμα ο παραλήπτης να ξέρει σε ποιον θα στείλει πίσω την απάντηση. Η μεταβλητή *ClusterNo* συμβολίζει τον αριθμό του cluster στον οποίο γίνεται η μετάδοση και η *Category* περιέχει το όνομα της σημασιολογικής κατηγορίας στην οποία θέλει να κάνει publish ο *propagator peer*.

Response message in propagated request message:

Propagated request msg:	*Response msg:*
"PublishMsg"	Tagname: *"TrueClusterMsg"*
	Data: *"TrueCluster"*
"LoadFiguresRequestMsg"	Tagname: *"LoadFiguresMsg"*
	Data: *" strClusterNo PeerID* $p(S_i(k))$ $p(D(k))$ $u(k)$ $S_i1(k)$ $pS_i1(k)$ $S_i2(k)$ $pS_i2(k)$*…"*

Η μεταβλητή *TrueCluster* περιέχει το πραγματικό όνομα του cluster το οποίο φιλοξενεί την σημασιολογική κατηγορία στην οποία *ο propagator peer* θέλει να κάνει publish, ενώ τα data του *LoadFiguresMsg* είναι τα load-figures του κάθε peer που ανήκει στον cluster *strClusterNo* για να τα καταχωρήσει ο *propagator peer* στον *ClusterLoadFiguresTable*.

Directory Layout



ΒΙΒΛΙΟΓΡΑΦΙΑ

[BGKS02] D.Brookshier, D.Govoni, N.Krishnan and J.C.Soto. *JXTA: JAVA™ P2P Programming*. Sams Publishing, March 2002.

[GG02] J.D.Gradecki, J.Gradecki. *Mastering JXTA:Building Java Peer-to-Peer Applications*. John Wiley & Sons, November 2002.

[Gnu] Gnutella web site: http://www.gnutella.com/

[JXTA] JXTA web site: http://www.jxta.org

[JXTA05] *JXTA v2.3.x: Java™ Programmer's Guide*. Sun Microsystems, Inc. January 2005.

[Kaz] Kazaa web site: http://www.kazaa.com

[KR03] J.F.Kurose, K.W.Ross. *Computer Networking A Top-Down Approach Featuring the Internet*. Pearson Education, Inc. 2003

[SETI] SETI@home web site: http://setiathome.berkeley.edu/

[Tor] Bit Torrent web site: http://www.bittorrent.com

[W02] B.J.Wilson. *JXTA*. New Riders, June 2002.

[Wiki] en.wikipedia.org

[XKTC03] C. Xiruhaki, M.Koubarakis, P.Triantafillou, S.Christodoulakis. *Load Balancing in Peer-to-Peer Networks: System Architecture and Algorithms*. Technical University of Crete, March 2003.