Improving Routing in Unstructured Peer-to-Peer Systems

Harris Papadakis

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

University of Crete, 2011

UNIVERSITY OF CRETE

DEPARTMENT OF COMPUTER SCIENCE

Improving Routing in Unstructured Peer to Peer Systems

dissertation submited by

Harris Papadakis

in partial fulfillment of the requirements for

the PhD degree in Computer Science

Author:

[illegible]

Harris Papadakis, Department of Computer Science

[illegible]

Evaggelos Markatos, Professor, Dept. of Computer Science, University of Crete, advisor

[illegible]

Christos Nikolaou, Professor, Dept. of Computer Science, University of Crete

[illegible]

Dimitris Pleksousakis, Professor, Dept. of Computer Science, University of Crete

[illegible]

Antonis Savvidis, Assistant Professor, Dept. of Computer Science, University of Crete

[illegible]

Ioannis Titzikas, Assistant Professor, Dept. of Computer Science, University of Crete

[illegible]

Maria Papadopouli, Assistant Professor, Dept. of Computer Science, University of Crete

[illegible]

Isidora Roussopoulou, Assistant Professor, Dept. of Informatics and Telecommunications, National Kapodistrian University of Athens

Aproved by:

[illegible]

Angelos Bilas

Chairman of Graduate Studies

Heraklion, September, 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω κατ'αρχήν τους γονείς μου Γιώργο και Ανθούλα των οποίων η συνεχής υποστήριξη όλα αυτά τα χρόνια μου έδωσε την δυνατότητα να φέρω εις πέρας την παρούσα διατριβή, αλλά και την αδερφή μου Μαριέλα για το κουράγιο που μου έδινε σε δύσκολους καιρούς.

Θερμές ευχαριστίες αναλογούν στον επιβλέποντα καθηγητή μου Ευάγγελο Μαρκάτο χωρίς την καθοδήγηση του θα ήταν αδύνατο η ολοκλήρωση αυτής της εργασίας. Η υπομονή και στήριξή του επι σειρά ετών ήταν καταλυτική.

Επιπλέον, πολλά χρωστάω στην επί σειρά ετών συνεργάτιδα καθηγήτρια Παρασκευή Φραγκοπούλου. Η βοήθεια της, η συνεργασία της και οι πολύωρες συζητήσεις μας βοήθησαν σε κάθε βήμα διεκπεραίωσης της παρούσας εργασίας και υπήρξαν καταλυτικές στο να έρθει εις πέρας με επιτυχία.

Θα ήθελα επίσης να ευχαριστήσω τα μέλη της 3-μελής επιτροπής Ισιδώρα Ρουσσοπούλου και Μαρία Παπαδαπούλη για την σημαντική καθοδήγηση και βοήθεια τους.

Τέλος, σημαντική βοήθεια πρόσφεραν οι φίλοι μου όλα αυτά τα χρόνια του διδακτορικού. Η ψυχική υποστήριξη με πολλές ευχάριστες ώρες και διασκέδαση μου πρόσφεραν το ψυχικό σθένος απαραίτητο για την προσπάθεια που απαιτεί μια διδακτορική διατριβή. Θέλω λοιπόν να ευχαριστήσω ονομαστικά και με τυχαία σειρά τους Αντωνη, Γιώργο, Βασίλη, Ειρήνη, Γιώργο, Αριστέα, Μαρία, Σιλένα, Άννα, Σοφία, Αδάμ, Αντώνη, Γιάννη, Κώστα, Κοραλία, Μίκη, Χρυσούλα, Σταυρούλα και όσους αυτή την στιγμή μου διαφεύγουν.

Abstract

Recent years have seen the emergence of the deployment of many distributed systems of global scale. In addition, one of the age-long requirements and goals of distributed systems is the attainment of large degree of scalability, the ability of a system to cope with ever increasing number of participants. As one of the solutions to this problem, the Peer-to-Peer (P2P) paradigm was introduced and immediately received increasing attention. This leads to the quick evolution of the P2P systems, which branched out into two main categories, namely the structured and the unstructured systems. Structured systems are based on a more sophisticated way of interconnection between the participants of the system, which enables more efficient communication primitives. At the same time, this rigidness limits the scalability of those systems. Unstructured systems, on the other hand, are based on a more loose interconnection structure. Although this structure makes it easier for those systems to scale to global deployment, the communication primitives are less efficient, thus limiting their scaling ability. In this thesis we present an effort to tackle this problem of unstructured systems in many levels. We develop novel algorithms which enable the communication primitives of unstructured systems to better scale to a higher number of participants by reducing the functional costs not only for the P2P system itself but also for the network medium used. Contributions of this thesis include (i) a novel broadcast-like technique which generates a much smaller amount of network traffic, (ii) a new participants interconnection structure which enables more sophisticated search methods without transforming an unstructured system to a structured one and (iii) another, complementary or stand-alone method to create the P2P overlay so as to allow the system to work in the same manner a before, but at the same time reducing the traffic load imposed on the network medium used by the system.

Περίληψη

Τα τελευταία χρόνια αναπτύχθηκαν πολλά κατανεμημένα συστήματα παγκόσμιας κλίμακας. Επιπλέον, χρόνια απαίτηση και στόχος των κατανεμημένων συστημάτων είναι η επίτευξη όσο το δυνατόν μεγαλύτερου βαθμού επεκτασιμότητας, της ικανότητας δηλαδή ενός συστήματος να ανταπεξέρχεται σε συνεχώς αυξανόμενο αριθμό μελών. Σαν λύση στο θέμα αυτό, προτάθηκε η φιλοσοφία των Ομότιμων Συστημάτων (ΟΣ) και αμέσως απέκτησε δημοτικότητα και την προσοχή της ερευνητικής κοινότητας. Αυτό οδήγησε στην γρήγορη εξέλιξη των ΟΣ, τα οποία διακλαδίστηκαν σε δύο βασικές κατηγορίες, τα δομημένα και τα μη-δομημένα συστήματα. Τα δομημένα συστήματα βασίζονται σε μια πιο πολύπλοκη μέθοδο διασύνδεσης μεταξύ των μελών του συστήματος, κάτι που επιτρέπει ποιο αποτελεσματικές βασικές λειτουργίες επικοινωνίας. Παράλληλα όμως, αυτή η “ακαμψία” μειώνει την επεκτασιμότητα αυτών των συστημάτων. Τα μη-δομημενα συστήματα, από την άλλη μεριά, βασίζονται σε μια πιο χαλαρή δομή διασύνδεσης των μελών. Παρόλο που το γεγονός αυτό διευκολύνει τα συστήματα αυτά να επιτύχουν παγκόσμια ανάπτυξη, οι λειτουργίες επικοινωνίας είναι λιγότερο αποτελεσματικές, με αποτέλεσμα να μειώνεται η επεκτασιμότητα.

Σε αυτή την διατριβή θα παρουσιάσουμε μια προσπάθεια αντιμετώπισης αυτού του προβλήματος σε πολλά επίπεδα. Αναπτύξαμε και θα παρουσιάσουμε πρωτότυπους αλγορίθμους που επιτρέπουν στις βασικές λειτουργίες επικοινωνίας να επιτύχουν μεγαλύτερο βαθμό επεκτασιμότητας μειώνοντας τα λειτουργικά κόστη όχι μόνο για το ίδιο το ΟΣ αλλά και για το μέσο επικοινωνίας που χρησιμοποιεί. Οι συνεισφορές της παρούσας διατριβής συμπεριλαμβάνουν (i) μια πρωτότυπη τεχνική μαζικής επικοινωνίας που παράγει πολύ λιγότερο ποσό κίνησης στο δίκτυο, (ii) μια νέα δομή διασύνδεσης των μελών του συστήματος που επιτρέπει την ανάπτυξη ποιο

αποτελεσματικών μεθόδων αναζήτησης και επικοινωνίας, χωρίς να μετατρέπεται το όσο σύστημα από μη-δομημένο σε δομημένο και (iii) μια, είτε συμπληρωματική είναι ανεξάρτητη, μέθοδο δημιουργίας του ΟΣ που επιτρέπει στο σύστημα το λειτουργεί με τον ίδιο τρόπο όπως πριν αλλά παράγοντας λιγότερη κίνηση στο δίκτυο.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Distributed Systems

Distributed Systems are one of main and most popular fields in the Computer Science. The reason for this is multi-fold. From the need for greater computational power (Distributed and Grid computing), to the demand for ever-increased robustness of the software systems, to the development of systems that require a distributed solution (computation on data residing at disparate locations). Distributed Systems is a broad field and as such, many broad definitions have been defined.. We describe a system as distributed using the following definition:

A distributed system is comprised of autonomous computational entities that communicate with each other using some type of network.

In the case of the aforementioned field of Distributed Computing, the distributed system is comprised of computational units, which share the work load and exchange information with the purpose of reducing the time required for the execution of the computation. The appeal of Distributed Computing lies not only in the appeal of increased computational power but also in the performance/price ratio it offers. Generally speaking the performance of a computer does not increase linearly with its price. Therefore one can achieve better performance for the same price, employing a large number of cheaper computers. In addition, Distributed Systems generally aim at (and generally exhibit) increased robustness, due to the fact that the comprising autonomous entities fail independently. As a result, when one of them fails, most of the system still remains functional. Finally, one could find a lot of systems which are inherently distributed and thus require a distributed solution. Those cases include such well-known systems in Computer Science as the World Wide Web, E-mail, FTP, DNS and lately Peer-to-Peer (P2P) systems.

However, the most important case in favour of Distributed Systems is the need of people around the world, to collaborate and and share information through their independent, scattered computers. This was the main reason for the creation of the Distributed Systems design paradigm of “Peer to Peer” (also P2P) systems. P2P systems were introduced to enable the design of global-wide systems comprised of large numbers of commodity computers and are the main subject of this dissertation.

Distribution introduced a large number of notions and issues in software design, such as multiple points of both control and failure, resource sharing, transparency and so on. It also introduces a new kind of algorithm cost, besides those of space and time, which is the communication cost (i.e. the amount of data exchanged during the operation of the distributed system). The purpose of this dissertation, which we will describe in detail further on, is the study of the communications costs and mechanisms of a certain category of P2P systems.

1.2 Peer-to-Peer Systems

In both the areas of Distributed and Parallel Systems, a popular trend has emerged over the last few years, based on the “power of the mass” notion. In Parallel Systems this trend was incarnated with the emergence of the Grid Systems, whereas in the area of Distributed Systems emerged the Peer to peer (P2P) paradigm. The main notion behind both the Grid systems and the P2P paradigm is the desire to utilize the enormous power contained in the sheer mass of simple desktop computers and end users, in many terms such as computation power and storage (file sharing). This idea has been made possible by the rapid increase in the speed of the network at the edges of the Internet, but is a natural evolution of the increasing trend in software evolution of moving away from monolithic systems towards increasingly distributed ones.

With the possible exception of some early distributed systems, the majority of them was designed, for many years, using the Client/Server (C/S) paradigm. In this design philosophy, the entities comprising the distributed system are divided into two categories/types. The servers, which are responsible for the provision of almost the entirety of the functionality of the system, and the clients, which simply use the system, without providing some kind of functionality. This paradigm, in essence, defines a one-to-many relationship between the sever and its clients. The P2P paradigm, which we will describe next, is therefore the natural evolution of Distributed Systems since it allows for (m)any-to-(m)any relationships.

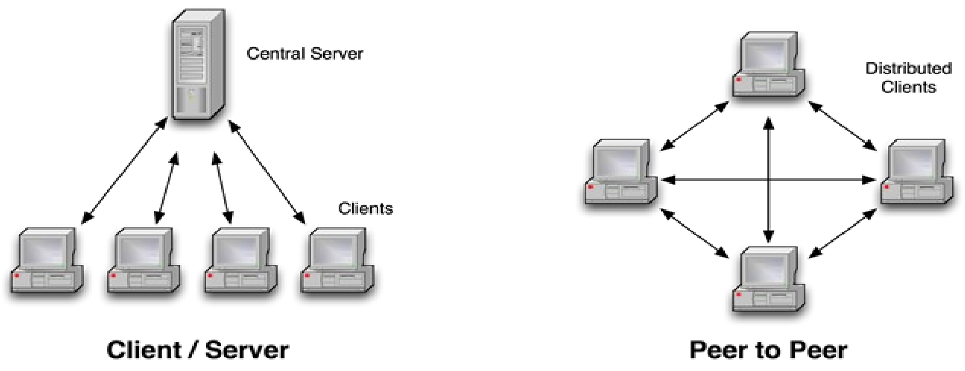

Figure 1.1: Client/Server vs P2P architecture

Well-known examples of distributed systems, which were developed based on the C/S paradigm include the World Wide Web (web servers and browsers), the File Transfer Protocol (FTP servers and FTP clients) and the Domain Name System (DNS). Soon enough, however, the problems inherent in this paradigm became apparent.

The main disadvantage of this design paradigm is the limited degree of scalability it offers. It is a common fact that the number of clients in a system is much larger than the number of servers in it. This means that as the number of clients in the system increases, so will the work load of the servers. As a result, a given number of servers unavoidably has an upper limit in the number of clients they can serve.

In addition, servers comprise the Achilles's heel of the distributed system. Since they implement and provide almost all the functionality of the system, a possible malfunction will often result in the cessation of operation of the whole system. This obviously conflicts with one of the main reasons of the existence of Distributed Systems, as we mentioned before, which is increased robustness and availability of the system. Even the use of a large number of replicated servers will not solve this issue, since usually all of them reside in the same geographic space or belong to the same authority. As a result, it is easier to for the functionality of the system to be impaired either by accident (a malfunction in the network of the horsing site) or on purpose (a Denial of Service attack).

For these, and other reasons, the Peer-to-Peer paradigm was introduced. In the recent years there has been an increased interest in the P2P systems by the research community. Interestingly enough and in contrast to other research topics over the years, this field became popular among the research community after P2P applications had already been popular and in use around the world. This is therefore a paradigm which already has had an impact in the digital world even at its infancy.

P2P systems are comprised by equal entities with similar, if not the same, functionality but also importance to the system. These entities are called "peers".

Thus, each server also functions as a client, by requesting some service from the rest of the members of the system. At the same time, each clients also serves as a server, by offering resources (computational power, storage space, bandwidth e.t.c) and/or functionality to the system. This differentiation of the P2P systems from traditional distributed systems has several effects in the characteristics of a system based on the P2P paradigm, which we will describe in detail next.

1.3 Inherent Characteristics

The term "peer to peer" prepositions that it describes a system where all individual components are equal in the tasks they perform. However, this equality can be interpreted in two ways, which lead to two different definitions. If this equality is interpreted as the fact that each peer both acts as a server and a client, one should label as P2P systems, those that rely on some central entity for coordination of the peers. On the other hand, a definition that demands complete equality in all aspects and functionalities would rule out systems such as Gnutella 2, one of the most widely used and studied P2P systems today.

In our belief, a modified version of the second definition is closer to the truth. Thus,

a peer-to-peer system is a distributed system where peers both offer and demand service and coordination is not based on a well-known authority.

This definition rules out centralized systems while still allowing for different roles in the system, provided that any peer may assume any role.

The first and most important characteristic of a P2P system, **scalability**, is derived from the fact that each new peer joining the system, as mentioned before, offers services as well as demanding them. Thus, P2P systems are inherently more scalable than traditional client-server systems. One important thing to notice here is that in order for a P2P system to be scalable, peers should contribute not only in the sense of services offered but also in the work required for the coordination of the cooperating peers. This means that the peers need to share the load of the functionalities offered by the system. In this sense, as we said before, a system where each peer does offer services but relies on a centralized server for the coordination of the offer with the demand of those services, is not scalable and thus not pure P2P (e.g Napster). Peers offer both the computational power and bandwidth required for the operation of the P2P system.

Another important characteristic inherent in the P2P systems is, of course, **decentralization**. Since, according to the definition, there is no well known, central authority there is no critical component for the operation of the system, a component, which could prove to be a bottleneck. This, of course, is a very important aspect not only in terms of efficiency and scalability but also in terms of robustness, another important characteristic of P2P systems.

The fact that no peer is irreplaceable leads to three more important characteristics of P2P systems, namely **robustness**, **security** and **autonomy**.

Apart from an increased degree of scalability, P2P systems are inherently more robust than their traditional counterparts. Possible malfunctions in parts of the system have a smaller, limited impact in the functionality of the system for many reasons. The peers that comprise the system are (in a large degree) equal and similar in their functionality. As a result, there is no "special" or unique part of the system, whose error-free operation is essential to the well-being of the whole system, there is no role that any peer cannot assume when some other peer fails. In addition, P2P systems are usually large-scaled systems. It is thus a common characteristic of these systems, the entities which comprise them to be geographically distributed all over the globe. This means that any malfunction in some geographical area will only affect a small part of the system.

This robustness also offers greater resilience and security against malicious attacks since there is no central point of operation that an attacker could attack in order to bring the system down or hamper its operation.

The replace-ability of any peer also offers great autonomy to the operation of the P2P system as a whole and each peer in particular. Each peer can come and go as it pleases. In fact, one of the negative characteristics of P2P systems is the transient nature of their components, the fact that is that peers do come and go as they please.

Indeed, not all characteristics inherent in the P2P paradigm are beneficial. A P2P system is also very **dynamic** and **unstable**, since each peer is under the control of a different authority. No peer can be inherently deemed as dependable both in terms of lifetime but also in terms of the services it provides and its participation in the operation of the system.

Anonymity is another aspect of P2P systems, which can be both beneficial and harmful. Since each peer is under different authority, it is difficult to track the identity of a peer. In many cases, the only thing that identifies a peer is an IP address, which can easily be changed or spoofed [74]. This is an advantage in cases where anonymity is desirable (for instance, to avoid censorship) but also this is the main reason behind the existence of the free-riding effect, where peers use the system without offering anything to it. Anonymity and autonomy make it very easy for any peer to misbehave and hamper the operation of the system. Peer-to-peer systems however owe their operation in the fact that disposability ensures than a large number of peers need to misbehave in order for the system to stop working and the majority of people do not generally want to harm the hand that feeds them. That is, people know that trying to take advantage of the system to get more out of it might mean that they end up getting nothing due to the collapse of the system.

A peer-to-peer system is also usually highly **heterogeneous** in many aspects such as the hardware every peer runs on, networking (connection speed, whether the peer is behind NATs, firewalls, etc), services offered and more. P2P systems should not only take into account this heterogeneity but indeed try to take advantage of it.

In addition to the above, it is important to note another kind of characteristic of the P2P systems and that is their better performance to price ratio. As we mentioned before, according to the philosophy of the Client/Server paradigm, a small number of servers undertake the service of a much larger number of clients. As one can expect, this means that it is essential for the servers to be either as high-end equipment as possible and/or as many as possible, in order to cope with the increased load. This means an increased cost of obtaining and maintaining this equipment.

On the other hand, the majority of P2P systems are comprised of a large number of commodity computational units with less resources and capabilities but also greatly reduced cost. These units, apart from their low cost, usually have to do with computers already present (i.e. home and laboratory computers). As a result, P2P systems also allow us to utilize hardware and resources already existent but under-utilized. Even if the hardware is not already in place, it can be acquired for a reduced price compared to the one required by the C/S paradigm.

1.4 Desirable characteristics

Despite the important characteristics already inherent in P2P systems, there are still more whose existence is required in an efficient, functional P2P system.

First of all, a P2P systems designer should take care not to nullify any of the inherent characteristics mentioned above. For instance, the flooding mechanism employed in Gnutella 1 [14] greatly reduced the **scalability** of the system, even though Gnutella 1 was a purely decentralized P2P system. As we will describe below, the reason for this is that even though the available bandwidth of the system increased with the addition of a new peer in the system, so did the cost of the flooding.

One of the most important additional aspects of a P2P system is, of course, **efficiency**. This is usually interpreted in quality of service and cost for each functionality of the P2P system. Costs are also usually divided into space and time costs, that is, the bandwidth and time required to perform any of the functionalities.

Robustness is still another issue, depending on the architecture of the P2P system. Even though any peer could take the place of a failed peer, information residing on the failed peer cannot be recovered unless by some mechanism of the P2P system, designed for that purpose. If that information is critical to the operation of the system, the system itself is less robust.

Fairness is a very important aspect in the context of P2P system, where there is no trust. A correct P2P system needs to make sure that each peer offers to as well as takes advantage of the system, otherwise many of the above characteristics, such as scalability, are cancelled and the whole idea behind the P2P paradigm is nullified. One of the largest issued in P2P today is the problem of free-riding, where users participate in the system without offering to it. This effect is another instance of the "Tragedy of the Commons", described by Garrett Hardin[3]. Fairness does not necessarily mean that everyone offers the same in the system. Rather, it means that a peer offers as much as it takes.

Another desirable characteristic is **expressiveness**. Peers requesting a service need to locate it first and the querying and lookup mechanism offered by the system needs to allow of an expressive query language to support advanced queries.

Security is another issue with many facets.

First of all, the system should protect the integrity of the data it holds. Malicious entities should not be able to substitute correct data with forged data; neither supply a requester with forged data. This is a very important issue in P2P systems today, since the whole idea of the P2P paradigm is the fact that the peers depend on each other for the correct operation of the system. On the other hand, however, as we said before, peers not only cannot be trusted, but also can usually remain anonymous and intractable. This makes it easy for a malicious peer to lie and disrupt the functionality of the system.

Another aspect of security is the control over the accessibility of data. The system needs to ensure that any kind of information is accessible only by authorized peers. Unauthorized peers should be aware of neither the data exchanged between two peers nor the peers which exchange the data, even if the unauthorized peers participate in the execution of the exchange. This is an aspect of security that the inherent anonymity in the P2P systems is helpful.

Recent research also showed that security in P2P systems needs to take into account not only attacks on the system itself, but also any security holes which might enable a malicious user to exploit any global-scale system for a coordinated distributed, denial of service attack.

Another important aspect of the lack of some central authority is that peers need to not only cooperate by themselves, in an ad-hoc fashion, but also need to cooperate to preserve the aforementioned characteristics. This means that peers need to organize themselves and self-heal the system. Peers need to have some distributed way of bootstrapping in the system. A P2P system is highly dynamic and peers need to be able to re-organize after addition or removal of a peer from the network. As we said, every peer is replaceable, however the system must be able to detect the failure or departure of a peer and replace it quickly and efficiently.

1.5 Uses of P2P Systems

The aforementioned advantages of P2P systems resulted in the use of this paradigm in a large number of applications, during the few years since its introduction, applications whose majority has been deployed in a global-scale. The most well-known and popular example is the file-sharing applications, which allow their users to exchange their files directly, without the use of servers which would act either as index services or storage space providers. The file-sharing applications are the first type of application designed using the P2P paradigm, even though a number of people believe that this design philosophy pre-existed in older systems, albeit without being given a name. An example of this could well be the architecture of the Internet itself.

The P2P paradigm, as a distributed system design philosophy was also used in other large-scale applications, such as the Grid Systems. The Grid Systems share many common goals with the P2P systems, like the joint use of computational, storage and other resources. Computational units, throughout the planet, under the authoritative charge of different entities, combine together to form a super-computer, which could be used by all participants. It was only thus expected that parts of the Grid Systems would be designed using the P2P paradigm.

Another type of application widely using the P2P philosophy are the Content Delivery Networks, such as Akamai Technologies[75]. Similar to the file-sharing applications these networks aim at the fastest delivery of content to their users. This content usually has to do with large files such as real-time video and sound. The storage space of the various computational entities that take part in the system is used in order to create and store replicas of the content. The request for the delivery of any content is served by the peers closest to the requester.

In the recent years, a revolution has taken place in the computer games industry, with the rapid spread of Massively Multi-player Online Games. Games of this kind aim at simulating virtual worlds containing millions of inhabitants-users. The adoption of the P2P technology promises to allow the attainment of such an ambitious goal.

There have been a number of other applications, not readily related to P2P systems, which however have made successful use of the paradigm. A well-known example of such an application is the Skype application for online telephony. In the context of this application, the distributed philosophy of the P2P systems was used both in order to implement a distributed index of users as well as to enable the communication between users which both reside behind a Network Address Translator.

P2P philosophy was also naturally, easily and directly applied to the field of wireless sensor networks as well as the field of Ubiquitous computing.

In the case of sensor networks, algorithms designed for P2P systems are used with success, in order to facilitate the synchronization and cooperation of sensors with limitations regarding the radius beyond which it is impossible to transmit and receive information. These networks are comprised of small computational units equipped with small range antennas. These units work independently and cooperatively in a loose fashion, in order to achieve a common goal. Their nature renders them ideal for the application of the P2P paradigm.

As far as Ubiquitous computing is concerned, they are similarly comprised of computational units of small scale and reduced power, built in objects of every-day use. These devices can communicate with one another and cooperate in a P2P fashion. Often, these devices' geographical condition changes all the time, and so, at each given moment, new groups are created and dissolved in an ad-hoc fashion, based on the proximity of these devices with one another.

1.6 Types of P2P Systems

P2P systems are divided into two large categories, depending on the degree of structure they contain. Since P2P systems were created with the goal of allowing applications to achieve global scale, it is not desirable (and often not possible) to require for peers to have global knowledge of the existence of every other peer in the system. Thus, each peer has a (very) limited (view) of the whole system and knowledge of a small subset of other peers, with which it is possible to directly communicate and cooperate.

The cooperation links of peers form an overlay network, which forms some kind of structure. The rigidity or looseness of this structure classifies the system in one of the two categories, the structured and the unstructured systems.

Unstructured Systems

As we mentioned before, since there is no one entity to single-handedly synchronize the peers, each one is aware of and cooperates-communicates directly with a small number of peers (”neighbours”), which also serve as that peer's access to the rest of the system, with which there is no direct connection (i.e. the peers it is not directly aware of). In the case of unstructured systems, the choice of the neighbouring peers is almost random. Each peer in the system has equal probability of being chosen as neighbour by another peer. This fact has both positive and negative consequences for the operation of the system. The system created in such a fashion is simpler and therefore more robust in the face of things such as churn (i.e. the constant arrival and departure of peers in the system) and malicious attacks. At the same time, however, the simplicity inhibits the development of more sophisticated operations and functions based on this system. For example, the search for another peer for instance, is done via a broadcast-like search mechanism called “flooding”. In this mechanism, each peer forwards the search to all of its neighbouring peers, which in turn does the same, until the peer in question is located. The simplicity of the overlay network construction fails to provide any information about the location of the requested peer and so there is no way to direct the search. Thus, flooding may have to contact each and every peer in order to locate the one it is looking for. In order to avoid such a costly operation, a numeric limit is usually employed in order to limit the number of times a search is forwarded in the overlay network. However, this comes at the cost of an increased possibility of failing to locate the requested peer.

Structured Systems

In the structured systems, on the other hand, there are specific rules which govern the connectivity of each peer, that is which peers are suitable as neighbours for each specific peer. Each structured system has its own rules, but they all have one thing in common. The utter purpose of the overlay creation rules is the construction of an overlay network built in such a way as to allow the design and development of more complicated and efficient operations. The vast majority of these systems is based in the assignment of a single, unique, numeric identity to each peer in the system. This identity defines an order among the participating peers and the neighbours of each peer are chosen based on this ordering and the system's rules. Those rules are chosen as to, in essence, create a certain structure in the overlay, which in turn provides information regarding the location of the requesting peer in the network. This directional information makes possible the lookup of cooperation of any two peers in the system in a cheaper way than in unstructured systems. The cost for such a capability is that one ends up with a more complicated system, more sensitive both to the constant change in the system and the possibility of more complicated malicious attacks which may take advantage of the nature of the system's structure.

Structured versus Unstructured Systems

The research carried out in the context of this dissertation focuses on unstructured systems, due to the belief that the introduction of rigidity and too much structure in P2P systems contradicts some of the fundamental ideas in the P2P philosophy, such as the high replacability for each peer. For this reason, increased degree of structure has adverse effects in many of the systems' inherent and desirable characteristics, as we mentioned beforein the previous paragraph.

The global deployment of P2P systems designates the scalability issue as a very important one. Unstructured systems lack in this aspect due to the cost of the operation of the flooding mechanism. On the other hand, the concept of scalability also includes other aspects such as tolerance and endurance to fast and continuous changes in the system, known in the bibliography as "churn".

In global scale systems such as P2P systems, the large number of participating peers means that the actual peers which comprise the system changes constantly. A robust system should be able to function normally not only in the face of problems but also under conditions which are normal for a global-scale system such as heavy churn. Unstructured systems are good in this aspect due to their excellent self-healing and quick re-organizational capabilities in cases of problems and/or changes. The following table summarizes a qualitative comparison of the two kinds of P2P systems, based on the following criteria:

- *Scalability (in time)*: The time required for a lookup.
- *Scalability (in traffic)*: The number of network messages required for a lookup.
- *Robustness*: Resilience in many and quick changes in the structure of the system.

P2P System kind	***Scalability (in time)***	***Scalability (in traffic)***	***Robustness***
Unstructured	O(logN)	O(N)	Yes
Structured	O(logN)	O(logN)	No

As far as the scalability of the system in time and in traffic is concerned, the asymptotic notation we mention, assume a system of N peers. In addition, we assume an unstructured system which constructs a "random" overlay network, which will, therefore, have logarithmic diameter to the number of peers in the system.

1.7 Problem Description

Despite the aforementioned advantages of P2P systems, they themselves do not constitute the panacea for the solution of every distributed systems problem. Fundamental role in this fact play the problems that still plaque P2P systems design. Those problems, which we will describe in this Section, can be divided in two basic kinds. Some of them steam from the nature of the P2P systems themselves while others do not. These issues usually have to do with desirable characteristics of (distributed) systems, which P2P systems at the moment may lack, but appropriate research can introduce them.

A large part of a P2P system is the communication and cooperation mechanisms among the peers which comprise it, P2P systems being, after all, a widely scaled distributed system. As a result, many of the open issues in P2P systems have to do with the way peers exchange information with one another. At this point, we shall describe and analyse some open issues regarding the routing of messages and the efficiency of the cooperation capabilities offered by unstructured P2P systems today.

Scalability

As mentioned before, one of the key goals of P2P systems is to allow for a distributed system to achieve global-scale deployment. It is therefore obvious that scalability is one of the most important issues. In unstructured systems, any limitations in scalability exist in the communication mechanisms, having to do with the amount of traffic generated in the network. A high amount of traffic is generated during the operation of unstructured systems, which limits their scalability, which is due to more than one reasons.

The first reason is the fact that the flooding process used most often in these systems generates a number of messages between peers which increases exponentially with each forwarding from neighbour to neighbour. This number essentially equals with the average number of neighbours per peer to the power of the times the flooding messages were forwarded.

This stems from the following: As mentioned earlier, each peer receiving a message which is part of a flood, forwards it to all its neighbours except the one it received it from. In addition, this only happens the first time each peer receives a message belonging to the same flood. Additional messages are simply discarded. These facts have two outcomes. Assuming an (average) number of d neighbours per peer, the first is that each peer will forwarded the flood message to d-1 neighbours. Given that the system is comprised of N peers in total, a total number of N*(d-1) messages will be sent throughout the system and during the execution of a single flood. In addition, in each (parallel) forwarding, the number of messages in transit increases by a factor of d-1, since each of the messages already in transit (and about to be forwarded again) will generate d-1 replicas which will be sent to the corresponding neighbours.

This is mostly due to the fact that unstructured systems offer almost no information to facilitate the search for a specific peer, so as to allow the

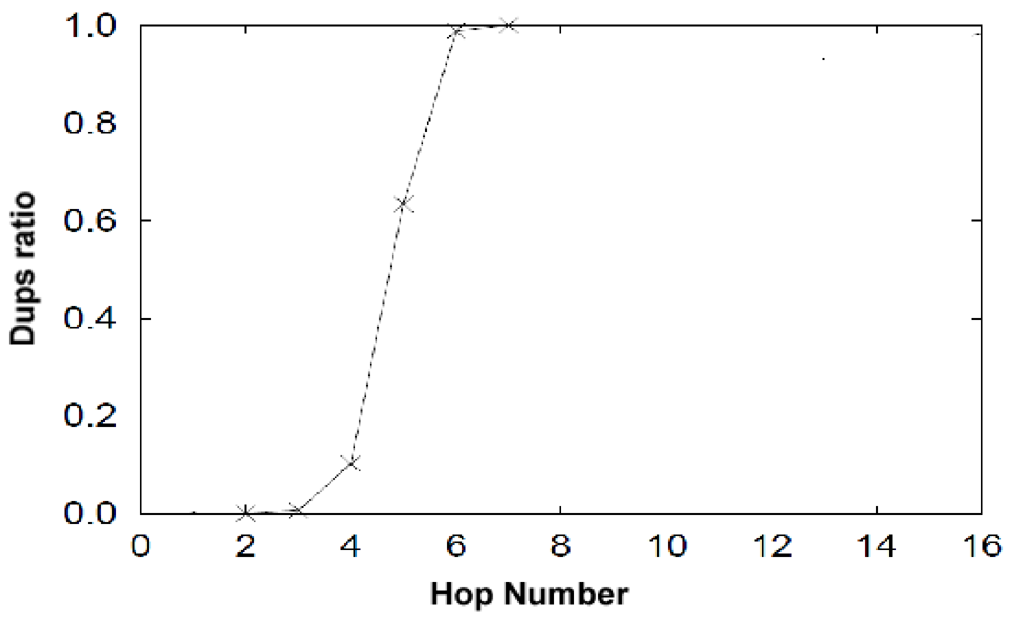

Figure 1.2: Ratio of duplicate messages per hop

communication of any two peers not directly aware of each other. As a result, any search mechanism would require at least N messages to locate a specific peer. As we have seen however, the number of messages exchanged during a single flood is N*(d-1). This means that there is a number of N*(d-2) messages which are not necessary in order for the flood process to produce the same results. This is due to the fact that each peer will receive one flood message once from each of its neighbours. This of course allows for greater robustness of the whole process, since this redundancy ensures that each peer will receive the message (unless that peer is completely separated from the entire overlay network). However, as we have seen, this also generated a large number of redundant, duplicate messages which each peer simply discards. We can see the amount of duplicate messages as a ratio of the total messages generated during each phase of the flood in the Figure 1.2.

It is therefore necessary to develop a method to reduce those duplicate messages, without sacrifices to the robustness of the flood process to churn.

The second cause of the large number of messages in unstructured systems is the fact that the almost total lack of structure does not allow for a more sophisticated and efficient lookup mechanism, as a replacement to

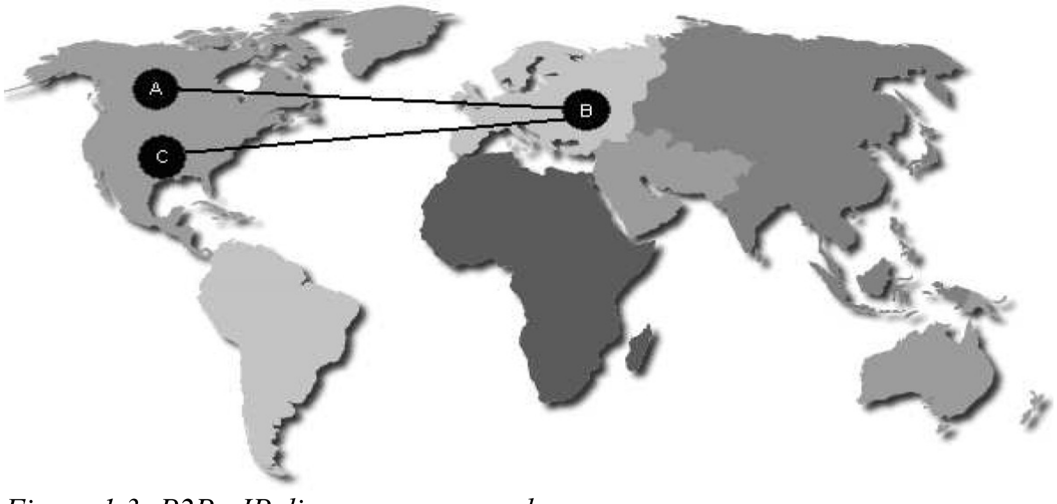

Figure 1.3: P2P - IP discrepancy example

flooding. The question here is whether we can manage to induce such information in the system without increasing the degree of structure, which would move it to the structured systems category and deprive it of its excellent self-healing properties. Even though these two seem contradictory, fortunately this is not the case, as we shall see in the work presented in this dissertation.

Finally, often in P2P systems research, the cost of a communications mechanism is measured in the number of messages exchanged between peers. For the sake of simplicity, the fact that a single message between two peers corresponds to a larger number of IP messages is usually overlooked. This disproportion between those two is due to the fact that peers which are directly connected with one another on the P2P network (and thus are neighbours) are not necessarily adjoin in the IP network. A simple example of this fact is illustrated in Figure 1.3, where peer B is a neighbour of both A and C.

In this example, a broadcast initiated by peer A would have to cross the Atlantic twice, since peer C will receive the message through B. On the other hand, one could create a new neighbouring link between peer A and peer C and remove the direct connection between either peer A and peer B or peer B and peer C. In this manner, peer A's message will only cross the Atlantic once, to get to peer B, which is after all, unavoidable.

This lack of correlation between P2P connectivity and IP connectivity creates additional traffic in the actual (IP) network. This fact, viewed from bottom-up, means that even if we do not reduce the number of P2P messages, the introduction of a higher degree of correlation between the P2P and the IP networks will lead to a reduction in P2P system traffic nevertheless, since each P2P message will generate a smaller number of IP messages than before.

As is the case with research generally however, there are some pitfalls which must be taken into consideration. As we have already mentioned, the manner in which the overlay network of a P2P system is constructed is tightly coupled with the way lookup is performed in that system. Therefore, any changes in the rules of peer connectivity will have to take into consideration the rest of the algorithms also operating in the same system and make sure these changes do not do more harm than good.

1.8 Contribution and structure of this thesis

This thesis is organized as follows. Chapter 2 presents some necessary in depth description of the algorithms widely used in unstructured P2P systems today. These algorithms already are an integral part of the systems. In Chapter 3 we present an algorithm for the reduction of the duplicate messages generated during the flood process. We shall show that the developed algorithm greatly reduces the number of redundant messages, without greatly affecting the reach of the flood, even in the face of heavy churn. Chapter 4 presents and analyzes a new scheme for adding location information in unstructured systems so as to allow for the use of a more efficient flood lookup which generates a greatly reduced number of messages compared to the original used today. We also show that this is possible without adding almost zero degree of structure to the system. Finally, Chapter 5 deals with the problem of P2P overlay and IP network discrepancy. We present a new way to create the P2P overlay as to increase the IP topology-awareness of the P2P overlay, while again leaving the amount of structure in the system almost intact.

Chapter 2

Unstructured P2P Technologies and Systems

2.1 Introduction

Unstructured P2P systems comprise the second generation of P2P systems [76] (the first being the Napster architecture, which however did not fit our definition of P2P systems), which is the prevailing architecture in use today. The vast majority of P2P-based systems in deployment today are based on unstructured systems and yet open issues still remain. One of the main reasons for this is that most of the systems in use today were developed by companies, without the participation of academia or in-depth research, or by a community of developers which worked on them in their spare time. Even though there has been a third generation of P2P systems, the robustness and resilience exhibited by unstructured systems makes them still very popular. In this chapter we shall try to present the basic characteristics of those systems. Then, we shall present the technologies and mechanisms in use today in unstructured systems and give a description and analysis of their advantages and disadvantages.

2.2 Common characteristics

The first purely decentralized, unstructured P2P system to be deployed was Gnutella 1 in 2000 [77]. Gnutella 1 was a file-sharing network. The architecture employed in Gnutella 1 is common to every unstructured P2P system. Avoiding the existence of some centralized server, the network was consisted of only equal, interconnected peers. Each peer was connected to and aware of a small number of other peers, called neighbours. Through those neighbours, the peer was indirectly connected to the rest of the network. All connections were bi-directional, meaning that is peer A is connected to peer B, likewise peer B is connected to A.

The ad-hoc nature of the unstructured systems dictates that, in principle, any peer should be able to connect to any other peer, assuming that the other peer would also want to connect to the first one. This fact makes the processes of both bootstrapping in the network and replacing a departed neighbour very cheap, both in terms of time and messages (network load). Those advantages however, do come at a cost.

The most important function of a P2P system is the lookup mechanism. This functionality is used whenever a peer needs to find the location of some desirable resource in the network. The ad-hoc and transient nature of the unstructured systems makes it very difficult to have any clue as to where the resource is located. This forces unstructured P2P systems to employ some broadcast-like mechanism to implement the lookup process. The generic form of this broadcast-like mechanism is called flooding. Flooding was employed in its general form, which we present below, in Gnutella 1.

Each peer needs to query the network for two things. The first is new peers to connect to during its bootstrapping process or when some of its old neighbours have left the system. The second type of query is of course a query for a piece of shared data (in the case of Gnutella, a file). In both cases, with the absence of centralized index, each requesting peer asks its neighbours not only if they contain the information it seeks but also requesting them to ask their own neighbours on its behalf, in turn. This leads to a Breadth-First-Search-like broadcast process, which was dubbed “Flooding”, which is illustrated in Figures 2.1 to 2.3. Flooding is necessary in systems like that, since there is no information about the location of any piece of data. Each peer may contain any kind of information and at the same time connect to anyone in the network. This kind of non-determinism makes unstructured systems highly resilient and robust at the same time however at the same time greatly reduces the scalability of the lookup process. The reason for this is that in order to locate some desired data or peer, the requesting peer has to query the entire network. The cost of doing exactly that increases with every new peer that joins the network. This was the reason for the meltdown of the Gnutella 1 network in 2001, with the demise of Napster. After Napster was shutdown, a large number of P2P users moved to the Gnutella network, increasing its size beyond the limits its then current architecture could cope with. The Gnutella network was saturated and could not operate [78]. This led to a redesign of the whole system and the introduction of some ideas to remedy the situation, ideas which we shall discuss shortly.

As in every P2P network up to now, in order to connect to it, a would-be peer needs to be, in some way, aware of at least one peer already connected to the network. This fact holds for both unstructured and structured systems. This is usually done with the use of dedicated services, which peers query, to receive a list of recently connected peers. The use of this service by some peer automatically inserts it in that server’s list of recently connected peers. This is the only process in P2P systems today, which remains centralized, even in purely decentralized P2P systems. The reason, of course, is that if that service was provided by some other P2P system, recursively, one would need to be aware of at least one peer in that system in order to use the service. However, this minimal degree of centralization is not a problem for two reasons. First, this service is very lightweight, meaning that any server can serve many peers without becoming a bottleneck in the rate at which peers enter the network. In addition, such a server can be set up anywhere, leading to the easy deployment of a large number of said servers, which will share the load between them. When a peer is finally aware of at least one already connected peer, it uses that peer to send a flood-search, looking for other peers in the network. Any peer that receives the flood and is willing to accept new neighbours sends back a reply. The joining peer then connects to them. If the number of neighbours is not satisfactory, the process is repeated.

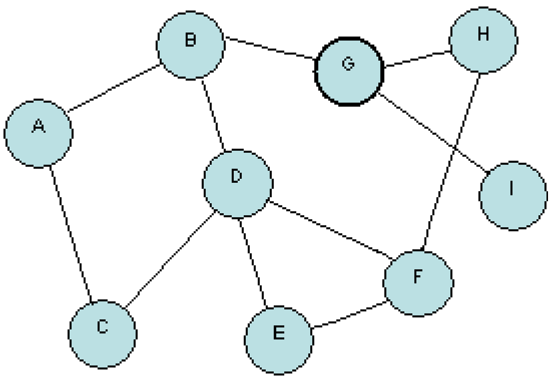

Figure 2.1. An example of Gnutella overlay network. Black lines indicate neighbour relationship. For example, A has B and C as its neighbours

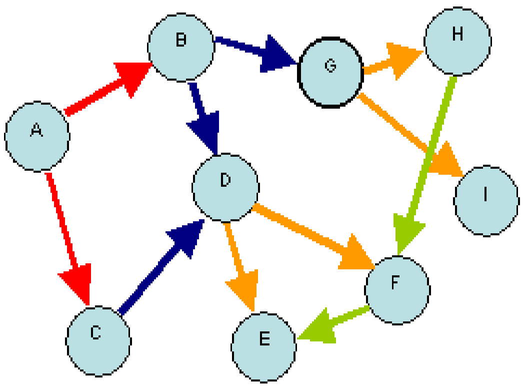

Figure 2.2. Client A makes a query to his neighbours B and C (red line). B and C forward the query to their neighbours (blue line), them to their neighbours (orange line) and so on

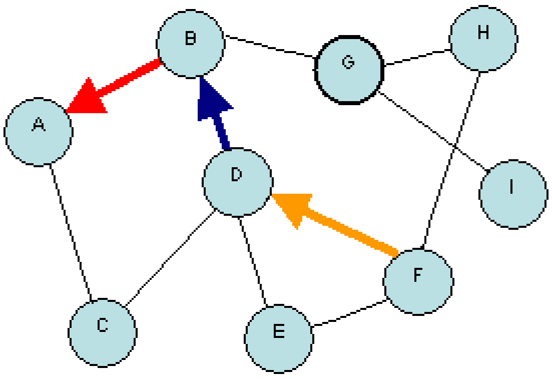

Figure 2.3. Assuming that F contains the data A is looking for, the reply will follow the backward path that A's flood followed to reach F.

After the description of a peer's bootstrapping process, we shall discuss in the depth the search mechanism we mentioned just before. The peer that initiates he flood sends the flood message to all its neighbours. The message contains, along with control information, which we shall discuss later, all the necessary information to identify the data being looked for. During this process, each peer that receives a flood message through one of its connections to other peers should propagate it to all of its own neighbours except the one it received it from. In addition, it should, of course, process it to see if it can satisfy the query. If it contains the requested information, it sends back a reply in the manner we shall describe shortly.

The first important thing to notice is that, each time a peer initiates a flood, it every message generated by it with a globally unique identifier (called GUID, in the case of Gnutella). When a receiving peer propagates the message of the flood, it tags the messages it sends to its own neighbours with the same identifier. This means that messages that were generated by the same flood share the same identifier, while messages belonging to different floods have different identifiers.

The purpose of this identification is two-fold. The first one is to avoid the retransmission of messages already propagated. Because the network of the connected peers forms a connected graph rather than a tree, there will be a lot of cycles in the paths formed by the connections between the peers. This means that a peer that has already received and forwarded a message to its neighbours may receive a message for the same flood, through another neighbour. In Figure 2.2, for instance, peer D receives the flood from both B and C. What is more, since C has already received the message from A (and that is why it sent the duplicate message to D), it will also receive a duplicate message from D. This is because, assuming D receives B's message first and it will forward the message to all of it neighbours, except B. This means that it will also send the message to C, regardless of the fact that C has already sent him the same message. Even if C's message had already arrived at D, D will process them on message at a time.

The other reason is that in order to retain the anonymity of the peer initiating the flood, there is no information inside the message being propagated, that could identify it. This, of course, rules out the possibility that a peer that contains the information requested can send a reply directly back to the originating peer. The only way to notify the requesting peer that the information has been located is through the same path the message followed to reach the peer with the information. This is made possible by having each peer that receives a flood message store, along with the identifier of the message, the connection (i.e.: the neighbour) through which it received the message. Thus, by having the replying peer tag the reply message with the same identifier as the flood messages, the reply (also called queryhit) can be propagated backwards, to the peer that initiated the flood. This process is illustrated in Figures 2.2 and 2.3.

One should notice that the tagging of the messages of a flood serves yet another purpose. Without this mechanism, the flooding process would continue indefinitely. However now, the flooding process will stop generating new messages, as soon as every peer in the network has been contacted. This corresponds to a number of steps equal to the diameter of the network, i.e. the largest number of hops between any two peers in the network. In the case of random networks, this is roughly equal to the logarithm of the size of the network.

Even with this limit, however, the load in messages of the flooding process can be too much for the underlying network. For this purpose, the Time-To-Live (TTL) field in the messages' format was introduced. This field is similar and with the same purpose as the TTL field in the IP header. This field contains a small number, which signifies the number of times a message can be propagated before it should be discarded. Each peer that receives a flood message first checks the TTL field. If the value in the field it greater than 0, it decreases that value by one and then forwards the message to all its neighbours (except the one it received it from). Otherwise the message is processed, in terms of checking whether that peer contains the information requested, but it is not forwarded. This mechanism decreases the cost of flooding at the cost of efficiency, since it also reduces the number of peers that will receive the flood. For instance, a flood of TTL 2 in the network of Figure 2.1 will only reach peers B, C, D and G and thus, not reply will be sent back from F.

Recent P2P systems also allow a search to be conducted, using the SHA-1 hash of the data, especially in the case of file-sharing applications. This kind of search is usually used after a keyword-based search is conducted and the desired file is located. The requesting peer can then learn the SHA-1 hash of the file, from the peer that replied and initiate a new flood using that hash. The result of this search will be the discovery of other peers with the same file under different filenames, which the peer can use to initiate a multiple sources transfer. I.e.: the peer can transfer only pieces of the file from each source, which will make the transfer of the whole file faster, if his link is not the bottleneck.

2.3 Techniques

The flooding process is common to most unstructured P2P systems today. Several schemes and mechanisms have (and are) also been used to improve various aspects of the systems. These techniques usually (but not all) aim to reduce the message cost of the flooding process. We shall present those in this section.

Ultrapeers

As mentioned above, the meltdown of Gnutella 1 was brought about by the sudden influx of ex-Napster users to the Gnutella network [78]. The traffic cost of flooding increases with the size of the network and that increase in the cost was too much for the majority of the peers to bear, since they were connected to the network through modems.

The idea that was used at that point was to reduce the flood-capable size of the network and this was accomplished by exploiting the heterogeneity of the users' connections. As we have mentioned before, P2P systems are highly heterogeneous. A fraction of the peers participating in the system sport high bandwidth connections. A measurement study published by Stutzbach et.al. [24] found that the majority of users in P2P systems have fast, stable connections to the Internet (Cable, DSL, T1 or T3) and 30% of them have very high bandwidth connections (at least 3Mbps).

People noticed that Gnutella and Napster comprised the two opposite extremes. Gnutella achieved scalability by distributing the system load between all peers. However, the fact is that some peers cannot handle that load. Napster imposed no load on its peers since the operational load (lookup) was all handled by the centralized server, resulting to reduced scalability. It was obvious that there was a need for a hybrid approach. This approach was called “Ultrapeers”.

Ultrapeers were first introduced in KaZaA, which was however a proprietary system and thus, there is little information in their implementation. Gnutella 2 was the first open system to implement the Ultrapeers approach. By curbing a little the definition of the P2P systems, which states that all peers are

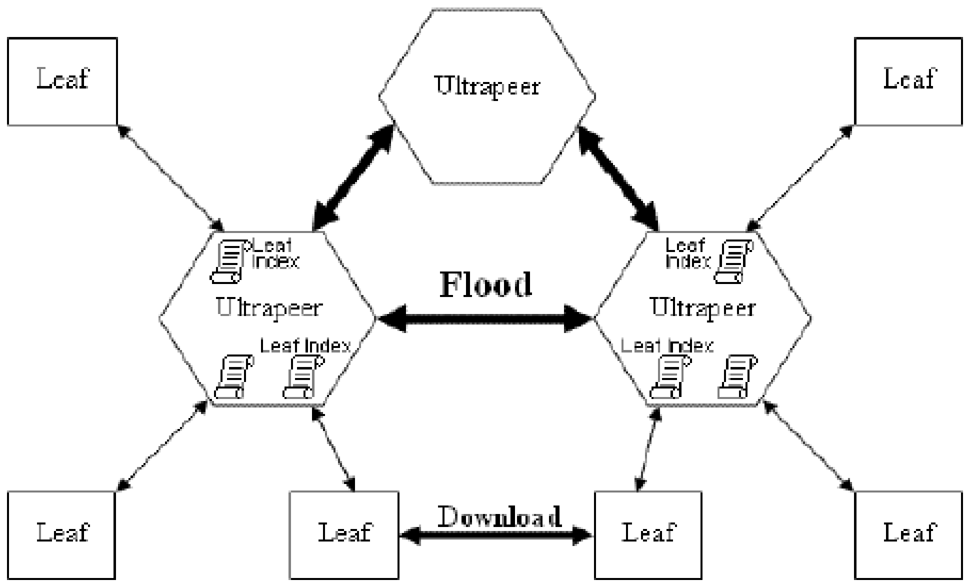

Figure 2.4. The Gnutella 2 architecture

equal, two distinct roles for each peer were defined, that of the Ultrapeer and that of the Leaf. In Gnutella 2, each Ultrapeer may play only one role at a time.

Ultrapeers, in essence, function as mini-Napster servers for a small number of other peers, which are called Leaves. Only high-bandwidth peers with high uptime are usually elected as Ultrapeers. As was the case in Napster, each Leaf connects to some Ultrapeer and sends an index of the files it shares to it. Leaves connect only to Ultrapeer and are not connected to each other. The Ultrapeer themselves do connect to each other to form a random network much in the fashion of the original Gnutella 1 network. Thus Leaves are, in essence, removed from the actual network. Instead, each Ultrapeer acts as a representative of each one of its Leaves to the network. Ultrapeers both perform queries and share files on behalf of their leaves as well as their own. Since Leaves are not interconnected, queries are thus flooded only on the Ultrapeer level. Ultrapeers propagate queries only to other Ultrapeers. Queries are propagated to Leaves only if they contain some desirable resource (file). In other words, Leaves do not participate in query routing and in the flooding procedure in general.

Given the fact that Ultrapeers have high bandwidth connections, they are more capable of handling the traffic load of the flooding process. What is more, the traffic of the flooding process is itself reduced with this scheme, since the number of peers being flooded is reduced.

The number of Leaves an Ultrapeer can serve is not defined by the protocol and is up to the implementation. However, recent measurement of the characteristics of the deployed Gnutella [24] network shows that the vast majority of the Ultrapeers support at most thirty Leaves. In addition, each Ultrapeer connects to thirty more neighbours at the Ultrapeer level. This degree is an order of magnitude larger than the degree of peers In Gnutella 1. This high degree has some important implications. The first and most important on is its impact on the diameter of the network. In randomly constructed graphs, such as Gnutella’s, the diameter of the network is equal to the logarithm of the size of the graph divided by the logarithm of the average degree of the nodes. In figures, this means that the Gnutella 2’s diameter is half the diameter of Gnutella 1 that has the same number of peers. The second benefit of having a high degree is related to the 1-hop replication technique, which we discuss below.

Another implementation detail is the fact that each Leaf peer connects to three distinct Ultrapeers, instead of one. The reason for this is two-fold. The first reason is that, should an Ultrapeer fail (i.e.: leave the network or crash), the Leaves connected to it will not lose all connection to the network. The second reason is that the index of each Leaf is thrice replicated in the network, making it easier to locate it when flooding the network with a TTL that is much smaller than the diameter of the network and thus does not reach every Ultrapeer. Figure 2.5 illustrates briefly the effect of this replication. A TTL that reaches about 5% of the network has a 16% chance of locating the desirable resource. Of course, in order to have a 100% chance, one would still need to reach 100% of the network, as is the case with no replication at all. This simple replication scheme was employed due to the fact that the diameter of the Gnutella network today has long ago exceeded the TTL employed in flooding, meaning that no flood will ever reach all of the network. One should note here that the brief analysis we presented is based of course on the assumption that each resource is identical. The degree of replication of each resource differs if several copies of the same resource reside in more than one peer.

Finally, it should be noted here that Leaves do not send their index itself to their Ultrapeers. Rather, they send a kind of bloom filters of their index rather than the index itself. The bloom filters are a compact approximation of a set. It supports membership queries, i.e.: can indicate whether the Leaf contains some keyword or not, albeit of course with reduced precision. Bloom filters are the next technique used in P2P systems today that we will discuss.

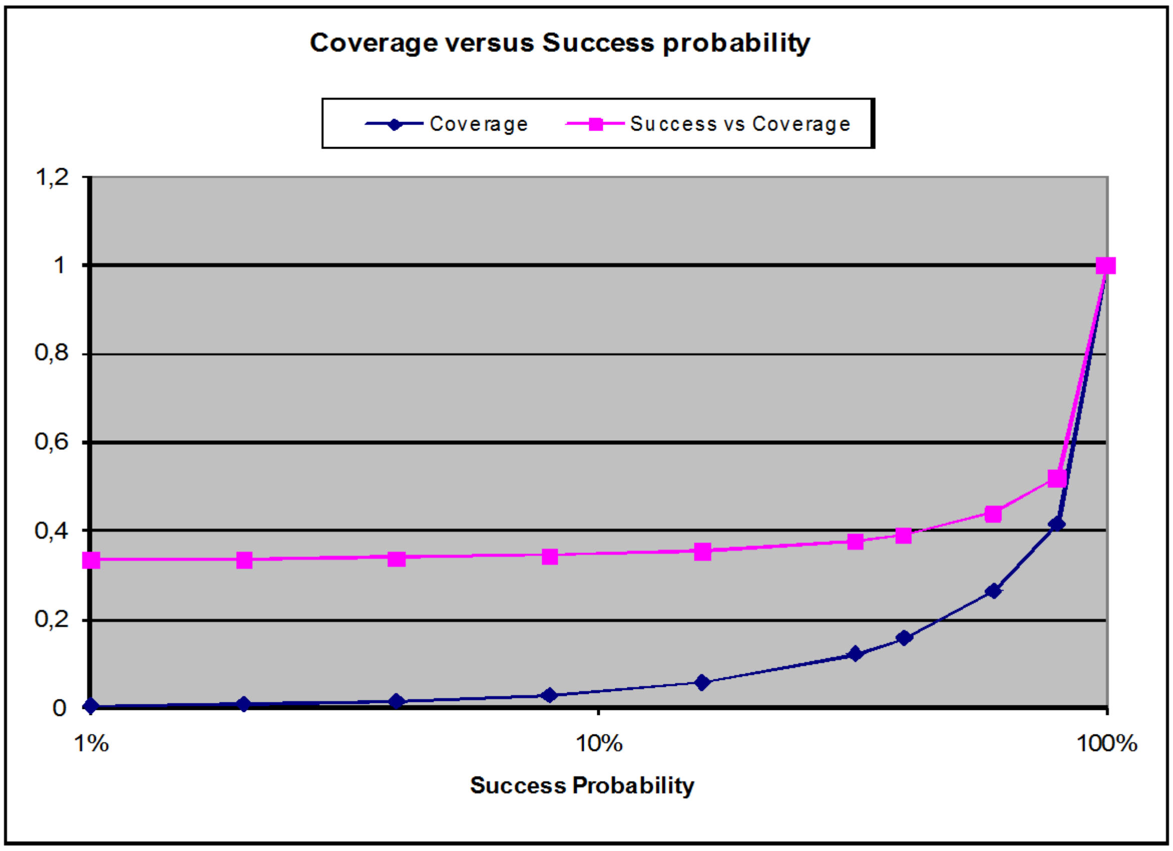

Figure 2.5. Effect of replication

Bloom filters

The Bloom filter was first conceived by Burton H. Bloom in 1970 [16]. It is a space efficient way to represent a set of objects (keys). In provides a way to test whether some key is part of the set the filter corresponds to, or not. Since it requires much less space than the actual set, there is some loss of precission translated in the posibility of false positives. This means that the bloom filter may indicate the existence of some key in the set even though it does not exist. Bloom filters however have no false negatives. This means that there is no way that the filter may indicate that some key is NOT in the set, when, in fact, it really is.

Algorithm description

An empty Bloom filter is a bit array of *m* bits, all set to 0. There must also be *k* different hash functions defined, each of which maps a key value to one of the *m* array positions.

To insert an element, feed it to each of the *k* hash functions to get *k* array positions. Set the bits at all these positions to 1.

To query for an element (test whether it is in the set), feed it to each of the *k* hash functions to get *k* array positions. If any of the bits at these positions are 0, the element is not in the set – if it were, then all the bits would have been set to 1 when it was inserted. If all are 1, then either the element is in the set, or the bits have been set to 1 during the insertion of other elements.

Unfortunately, removing an element from this simple Bloom filter is impossible. The element maps to *k* bits, and although setting any one of these *k* bits to zero suffices to remove it, this has the side effect of removing any other elements that map onto that bit, and we have no way of determining whether any such elements have been added. The result is a possibility of false negatives, which are not allowed.

It is possible to remove a key if all keys are available, where in this case, the bloom filter is recreated from all the keys, except the one to be removed. This is difficult in many cases, for two reasons. One is that usually the original keyword list is not available any more (The reason for using bloom filters in the first place is that we cannot afford to store all the original keys). It is also often the case that all the keys are available but are expensive to enumerate (for example, requiring many disk reads). Thus recreating the bloom filter is possible, but this should be a relatively rare event.

Counting Bloom filters have been introduced by Fan et al in [79] to remedy this drawback. For each bucket, a counter is maintained which counts the number of keys associated with this bucket. The removal of a key simply decreases the appropriate counter by one. When a counter is reduced to zero, the bucket is set to 0.

Space and time advantages

Bloom filters have a strong space advantage over other data structures for representing sets such as self-balancing binary search trees, tries, hash tables, or simple arrays or linked lists of the entries. Most of these require storing at least the data items themselves, which can require anywhere from a small number of bits, for small integers, to an arbitrary number of bits, such as for strings. This benefit of course comes at the cost of the existance of false positives, since Bloom filters achieve this low space cost by not storing the actual data. As mentioned in [80], “a Bloom filter with 1% error and an optimal value of *k*, on the other hand, requires only about 9.6 bits per element — regardless of the size of the elements. This advantage comes partly from its compactness, inherited from arrays, and partly from its probabilistic nature. If a 1% false positive rate seems too high, each time we add about 4.8 bits per element we decrease it by ten times.”

As far as time costs are concerned, Bloom filters can either add items or to check whether an item is in the set, only fixed constant, O(k), time, regardless of the number of items already in the set (again, at the cost of experiencing false positives). Especially in a hardware implementation the Bloom filter shines because its k lookups are independent and can be parallelized. This fact can be exploited for instance as a very fast first step in pattern matching algorithms.

To understand its space efficiency, it is instructive to compare the general Bloom filter with its special case when $k = 1$. If $k = 1$, then in order to keep the false positive rate sufficiently low, only a small fraction of bits should be set, meaning that the array must be either (or both) very large or (and) contain a large number of zeros. The information content of the array relative to its size is low. Such a Bloom filter can obtain a larger degree of space efficiency since it can be compressed to a large degree (due to the large degree of redundance contained in the large series of zeros) by a compression algorithm such as Huffman encoding [27]. The generalized Bloom filter (k greater than 1) allows many more bits to be set while still maintaining a low false positive rate; if the parameters (k and m)

are chosen well, about half of the bits will be set, and these will be apparently random, minimizing redundancy and maximizing information content.

It should be noted that inserting an element never fails since each insertion will simply, at most, change the value of a bucket. However, each insertion will increase the false positive rate. Also another interesting fact is that the rate at which the precision is reduced with each element being added decreases since when the filter is already almost full, the probability of setting a position of the array, which is already set, is high. This is when we need to test pairs of elements instead of single ones. In that case, the number of keys in the filter remain the same, but the precision is the square of the precision when testing single keys. One example of the case we described is Gnutella itself, where the vast majority (around 80%) of the queries contain at least two keywords.

Union and intersection of Bloom filters with the same size and set of hash functions can be implemented with bitwise OR and AND operations, respectively.

Probability of false positives

Assume that a hash function selects each array position with equal probability. The probability that a certain bit is not set to one by a certain hash function during the insertion of an element is then

$$1-\frac{1}{m}.$$

The probability that it is not set by any of the hash functions is

$$\left(1-\frac{1}{m}\right)^{k}$$

If we have inserted *n* elements, the probability that a certain bit is still 0 is

$$\left(1-\frac{1}{m}\right)^{kn}$$

the probability that it is 1 is therefore

$$1-\left(1-\frac{1}{m}\right)^{kn}$$

The probability of a false positive is can be calculated with the following formula. Each of the *k* array positions computed by the hash functions is 1 with a probability as above. The probability of all of them being 1, which would cause the algorithm to erroneously claim that the element is in the set, is then

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k}$$

Obviously, the probability of false positives decreases as *m* (the number of bits in the array) increases, and increases as *n* (the number of inserted elements) increases. For a given *m* and *n*, the value of *k* (the number of hash functions) that minimizes the probability is

$$\frac{m}{n}\ln 2 \approx \frac{9m}{13n} \approx 0.7\frac{m}{n},$$

which gives a probability of

$$\left(\frac{1}{2^{\ln 2}}\right)^{m/n} \approx 0.62^{m/n}.$$

The value of m, for the Gnutella Bloom filters is 65536 (2^{16}). In addition, Gnutella uses bloom filters with a *k* value of 1. There are several reasons for this, the most important being the fact that even though the Leaves' bloom filter is by far less than 70% full, the fullness percentage at which the most efficient number of hash functions is one, this is not the case for the Bloom filter of Ultrapeers. Since the Bloom filter of an Ultrapeer combines the bloom filters of thirty Leaves, that filter is on average, 60% full. Since the Ultrapeers have no knowledge of the actual key set of each Leaf, Leaves need also use Bloom filters with on hash function so that Ultrapeers can perform a union on all those filters.

One-hop replication

One-hop replication dictates that each peer informs all of its (immediate) neighbours of the information it contains. One-hop replication saves us the cost of the last hop during a flooding process. This is because the peers reached during the hop before the last one will have all the information reached in the last hop. It is obvious that the presence of 1-hop replication is of no use during all the hops before the last one, since even though we may always be aware of the information contained in any next hop, if that hop is not the last one, we cannot take advantage of this knowledge since each peer should propagate the query to every neighbour anyhow, so that they, in turn, pass it on. A quick computation, however, can show that, the number of messages generated during the last hop of a flood comprise the vast majority of the total messages generated, especially in networks with a large average degree, such as Gnutella today. We remind the reader that each Ultrapeer in Gnutella today connects to, on average, thirty Ultrapeers, which we will use in the formulas we describe in this section.

One-hop replication serves another very important purpose. We have already mentioned one of the disadvantages of flooding is the generation of duplicate messages due to the number of cycles in the network. In random graphs, where each peer can be connected to any other peer with equal probability, each time a message is sent, the probability of it being a duplicate (that is the destination peer has already received such as message) is equal to the percentage of the peers already reached by the flood. This means that, again in random graphs only, almost all of the duplicates are generated during the last hops of the flood, with the last hop containing the majority of duplicates. One-hop replication enables us to perform a flood, which will reach all of the peers, by not only avoiding to send the message to every last hop peer but also by avoiding to generate most of the duplicates.

This can be better understood with the use of some figures. We shall prove that the percentage of peers we need to reach during all the hops of a flood up to before the last has to be equal to 3/d, if we want to reach 95% of all the peers during the while flood. In the ratio above, d stands for the average degree of the peers in the network. If we assume a value of 30 for d, one can easily see that we need only reach 10% of the graph, in order to reach the rest 85% of the graph during the last hop. We shall prove this, step-by-step. First we shall prove that one needs 3*N messages (when not employing 1-hop replication) to ensure that at least 95% of the peers receive the flood message. As we mentioned before, the Propositions we prove in this section regard random graphs.

The proof is simple and is based on the form of the probability of the false positives of bloom filters we discussed above. In general, if we have an array of n bits, all set to zero, and randomly choose m of them to set to one (regardless of whether the previous value was already one or not), the percentage of ones we end up with is approximately equal to $1 - e^{-n/m}$. We can use this formula if we assume that each peer has a flag with values either one or zero, depending on whether it has received a message of the flood or not. Since the graph is a random one, whenever a message is sent to some peer, each peer in the network has the same probability of receiving it. Thus, if we want to reach 95% of the peers, we need the result of the aforementioned formula to be equal to 0.95. Thus:

Proposition 1: In order to reach 95% of a graph's nodes using naıve flooding we need a minimum of **3 $*$ N** messages.

Proof: Assume that *N* is the total number of peers and *x* is the number of messages transmitted during a flood. Using the above formula, and substituting m for *N* and n for *x*, we get:

$$e^{-x/N} \leq 0.05 \Rightarrow ln(e^{-x/N}) \leq ln(0.05) \Rightarrow$$
$$-x/N \leq -3 \Rightarrow x \geq 3 * N$$

Now we can prove the next step:

Proposition 2: In order to reach 95% of a graph's nodes that employs 1-hop replication using flooding, we need to reach 3/(d − 1) of the graph nodes in all hops except the last one.

Proof: Let n be a function that returns the number of new peers contacted at a given hop. Let f be a function that returns the number of messages generated on a single, given hop. Let d be the average degree of the graph. Initially f (0) = 0 and n(0) = 1. At each hop i it holds:

$$f(i) = n(i-1) * (d-1) \qquad (1)$$

because each one of the nodes that received a message for the first time at hop i − 1, will send it, at hop i, to all of their neighbours except the one it received the message from, thus to d − 1 neighbours. Let H be the hop before the last one. The total number of peers contacted up to hop H is $\sum_{i=0}^{H} n(i)$. Let r be the ratio of peers contacted up to hop H, then: $\sum_{i=0}^{H} n(i) = r * N$ (2) . We want to compute ratio r so that after hop H + 1, we will have reached at least 95% of the graph nodes. We have proven in Proposition 1 that we need a minimum of 3 ∗ N messages to reach 95% of a graph's nodes using naive flooding.

So $\sum_{i=1}^{H+1} f(i) \geq 3 * N$ (3) . If we replace function f from (1) in the above formula:

$$\sum_{i=1}^{H+1} f(i) = \sum_{i=1}^{H+1} [n(i-1) * (d-1)] =$$

$$= (d-1) \sum_{i=1}^{H+1} n(i-1) = (d-1) \sum_{i=0}^{H} n(i)$$

This combined with (1) and (2) gives:

$$(d-1) * r * N \geq 3 * N \Rightarrow r \geq \frac{3}{d-1}$$

Thus, the required result and end of proof.

As we mentioned, Gnutella employs one-hop replication today in the sense that Leaves send their indices to their Ultrapeers. However, one-hop replication is also used in the Ultrapeer level, that is among Ultrapeers. Each

Ultrapeer notifies all its Ultrapeer neighbours of its aggregate (bloom-filtered) index, the index of itself and all its Leaves' combined. This, in a sense actually is 2-hop replication, which is much more difficult to maintain in dynamic P2P networks. It is possible however in the case of Gnutella because Leaves only connect to three Ultrapeers (i.e. have a degree of three) in contrast to the Ultrapeers themselves, which connect to about thirty other Ultrapeers. Thus, the one-hop replication employed between Leaves and Ultrapeers is cheap, leaving Ultrapeers to cope with the one-hop replication between themselves. Of course, in contrast to pure one-hop replication, where indices of a peer do not change, intra-Ultrapeer one-hop replication requires each Ultrapeer inform its neighbours for any change in its aggregate index (i.e. whenever anyone of its Leaves departs).

The fact that each Ultrapeer aggregates all indices to generate one Bloom filter, which it sends to its neighbours, explains the fact why most Ultrapeers' Bloom filters are usually 60% full. This makes it possible that the departure of a Leaf from an Ultrapeer may not significantly alter the structure of its filter, if most of the set bits in the Leaf's filter were already set by other Leaves, in the Ultrapeer's aggregate filter, which, in turn, helps us reduce the number of updates required to maintain the one-hop replication in the Ultrapeer level, at the cost of much reduced precision in the filtering. This is because, in general, for any keyword, the filter of the Ultrapeer will indicate membership with 60% probability, which fortunately is not the case in Gnutella, because most of the queries contain at least two keywords. For the filter to produce a false positive, both keywords will have to be mapped to set bits in the filter. In the case of a 60% full bloom filter, this of course leads to 36% probability.

Dynamic Querying

One should note that not all floods need to have the same TTL value. Even though the TTL value is a maximum limit, it will be reached every time, even if the results the flood was looking for were found on the first hop. This means that the querying peer has no control over the number of peers that will be contacted, other than the TTL. This is not only an issue in the sense that more traffic than required is generated. In addition, the requesting peer may be itself "flooded" by incoming results if its query was a popular one, residing in a large number of peers. On the other side, the reduction in network traffic we could gain if we were able to terminate any flood at any point of its execution, due to an already adequate number of results received, could enable us to increase the cost of flooding for rare items, so as to increase the probability of successfully locating them.

The main problem is that, since the flood progresses in parallel, it is difficult for a branch of the Breadth-First-Search to know whether another branch has located the data. Unless of course, each branch's traversal is not done in parallel. This was the idea that led to Dynamic Querying. According to it, the peer that initiates the flood sends the flood message to only one of its neighbours, instead to all of them according to traditional flooding. That neighbour treats the flood message normally, forwarding it to all of its neighbours. When the results of this partial flood arrive, the peer can decide whether the results are satisfactory. If they are, the process is concluded, otherwise the peer sends another partial flood to a second neighbour and so forth, until enough results have been received or the peer has run out of neighbours.

In addition to that, Dynamic Querying employs another scheme. Instead of sending all partial floods with the same TTL, it sends the first flood with an initial TTL value. It then increases this value with every new partial flood sent. The initial value of the TTL is dependant on the popularity of the requested data, that is the degree of replication of that data in the network. The higher the replication, the lower the initial TTL value. In order to assess that popularity, Dynamic Querying first sends a small partial flood with a TTL of 1. The number of results returned defines the popularity of the data item.

Dynamic Querying is another mechanism of the unstructured systems that benefits from high Ultrapeer degree, since the number of available Dynamic Querying steps (partial floods) is equal to the number of neighbours.

This procedure can be used, of course, only when the TTL values are too small for the flood to reach the whole network. Otherwise, a high TTL value would mean that the first partial flood will have reached all of the peers and initiating another one will only generate duplicates.

2.4 Graph clustering

Two types of graphs have been mainly studied in the context of P2P systems. The first is random graphs which constitute the underlining topology in today's commercial P2P systems [7, 9]. The second type is small-world graphs which emerged in the modelling of social networks [4], introduced by Watts and Strongatz. It has been demonstrated that P2P resource location algorithms could benefit from small-world properties. If the benefit proves to be substantial then the node connection protocol in P2P systems could be modified so that small-world properties are intentionally incorporated in their network topologies.

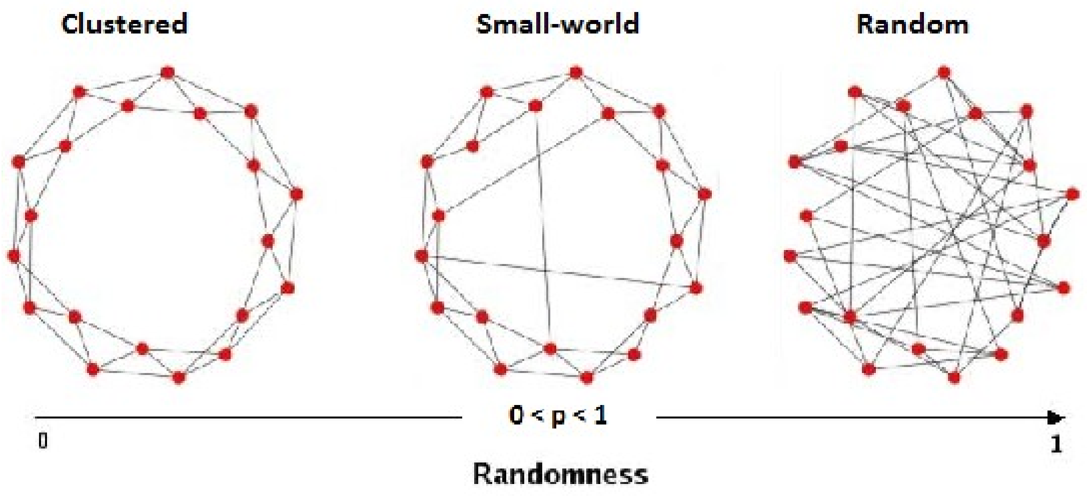

Figure 2.6: Graphs generated with different values of rewiring probabilities.

In random graphs each node is randomly connected to a number of other nodes equal to its degree. Random graphs have small diameter and small average diameter. The diameter of a graph is the length (number of hops for un-weighted graphs) of the longest among the shortest paths that connect any pair of nodes. The average diameter of a graph is the average of all shortest paths from any node to any other node.

A clustered graph is a graph that contains densely connected "neighbourhoods" of nodes, while nodes that lie in different neighbourhoods are more loosely connected. A metric that captures the degree of clustering that graphs exhibit is the clustering coefficient. Given a graph G, the clustering coefficient of a node A in G is defined as the ratio of the number of edges that exist between the neighbours of A over the maximum number of edges that can exist between its neighbours (which equals to k(k − 1) for k neighbours). The clustering coefficient of a graph G is the average of the clustering coefficients of

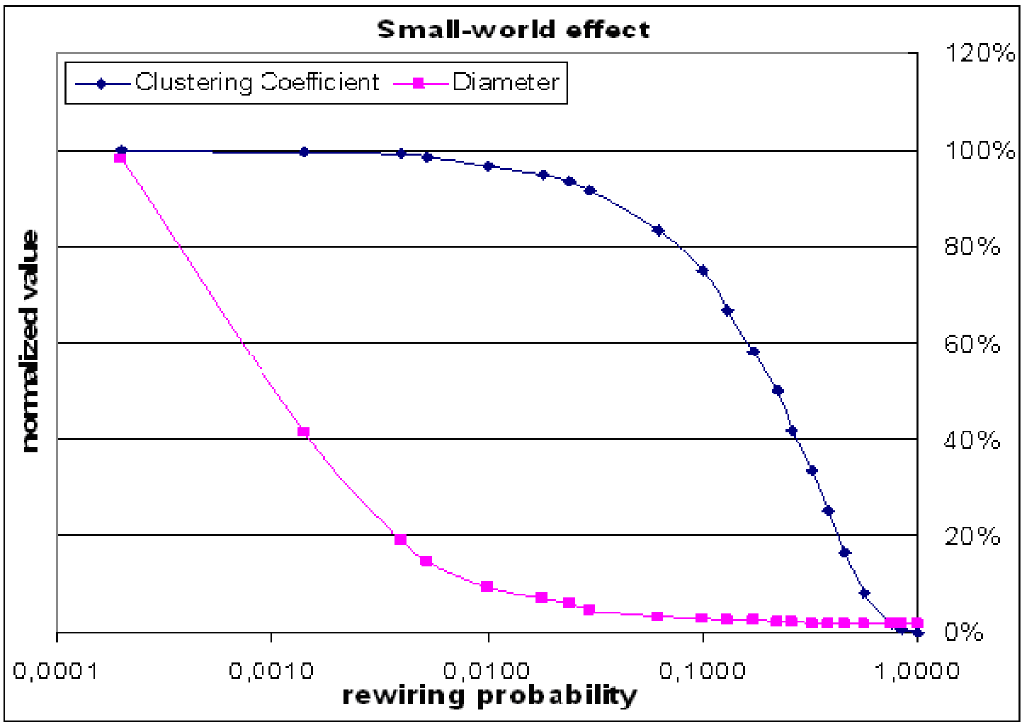

Figure 2.7: By rewiring a few edges of the initial clustered graph to random nodes the average diameter of the graph is greatly reduced, without significantly affecting the clustering coefficient.

all its nodes. Clustered graphs have, in general, higher diameter and higher average diameter than their random counterparts with about the same number of nodes and degree.

A small-world graph is a graph with high clustering coefficient yet low average diameter. The small-world graphs we use in our experiments are constructed according to the Strogatz-Watts model [4]. Initially, a regular, clustered graph of N nodes is constructed as follows: each node is assigned a unique identifier from 0 to N − 1. Two nodes are connected if their identity difference is less than or equal to k (in modN arithmetic). Subsequently, each edge of the graph is rewired to a random node according to a given rewiring probability p. If the rewiring probability of edges is relatively small, a small-world graph is produced (high clustering coefficient and small average diameter). As the rewiring probability increases the graph becomes more random (the

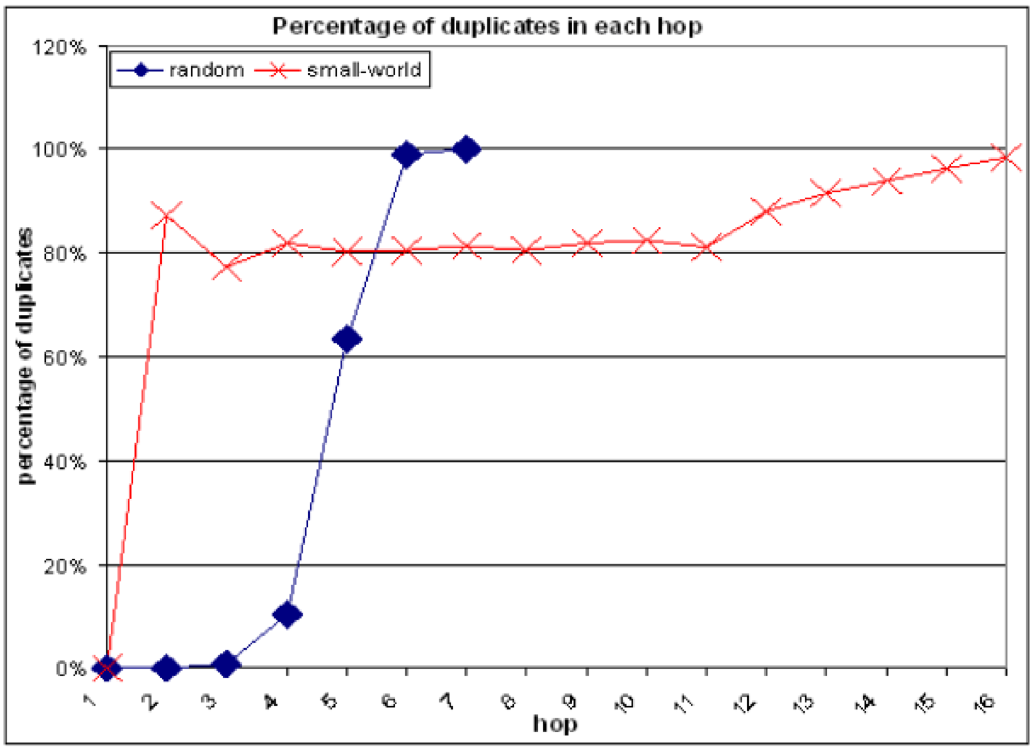

Figure 2.8: Percentage of duplicate messages generated per hop, in random and small-world graphs.

clustering coefficient decreases). For rewiring probability p = 1, all graph edges are rewired to random nodes, and this results in a random graph. In Fig. 2.6 once can see examples of graphs built with different p values. In Fig. 2.7, we can see how the clustering coefficient and the average diameter of graphs vary as the rewiring probability p increases. Small-world graphs are somewhere in the middle of the x axis (p = 0.01).

The clustering coefficient of each graph is normalized with respect to the maximum clustering coefficient of a graph with the same number of nodes and average degree. In what follows, when we refer to the clustering coefficient of a graph with N nodes and average degree d, denoted by CC, we refer to the percentage of its clustering coefficient over the maximum clustering coefficient of a graph with the same number of nodes and average degree. The maximum clustering coefficient of a graph with N nodes and average degree d is the clustering coefficient of the clustered graph defined according to the Strogatz-Watts model, before any edge rewiring takes place. Fig. 2.8 shows the percentage of duplicates messages generated per hop over the messages generated on that hop on a random and on a small-world graph of 2000 nodes and average degree 6. We can see from this figure that in a random graph there are very few duplicate messages in the first few hops (1-4), while almost all messages in the last hops (6-7) are duplicates. On the contrary, in small-world graphs duplicate messages appear from the first hops and their percentage (over the total number of messages per hop) remains almost constant till the last hops.

This effect is very important for P2P systems employing the 1-hop replication technique we mentioned above. As we described, in this case, naïve flooding is only used during the first phases of flooding, where not many duplicate messages are generated in the random graphs used today. The last forwarding of the flood message (which generates the most messages, both duplicate and not) utilizes the information provided by 1-hop replication in order to only forward the flood to appropriate peers and thus avoiding the broadcast-like flood.

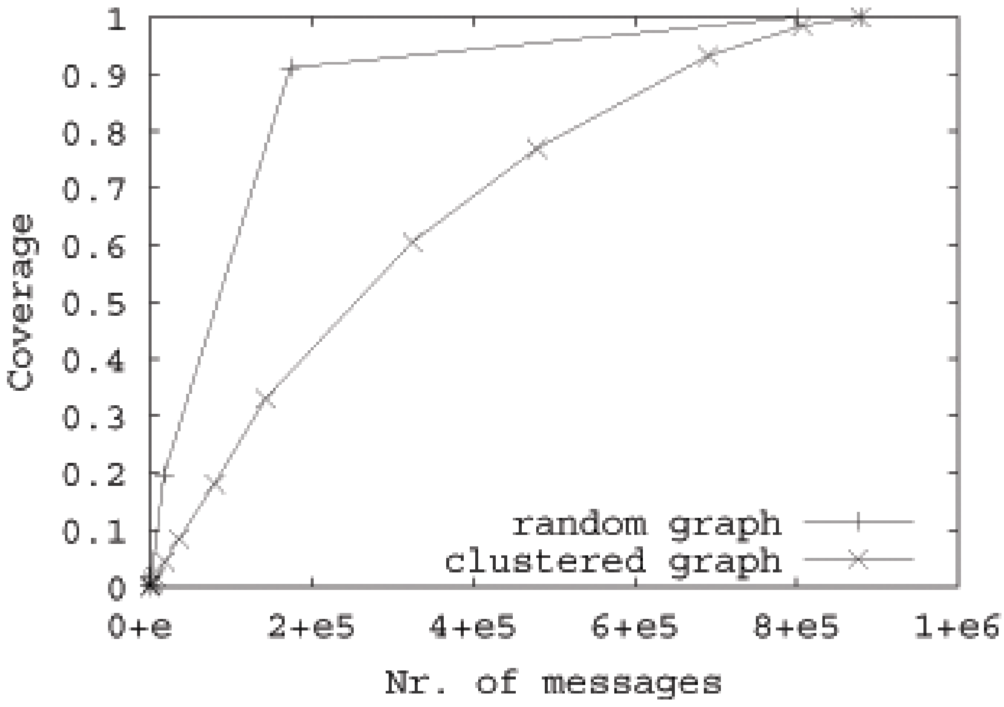

Figure 2.9: Coverage of the graph for given number of messages.

The above is one of the reasons P2P systems today prefer to form a random graph instead of a clustered one. On addition, due to the lack of duplicate messages during the first propagations of the flood, one can reach more peers in a random graph than in a clustered graph, using the same number of messages. This is shown in Figure 2.9.

Chapter 3

A Feedback-based Approach for Reducing Duplicate Messages in Unstructured P2P Systems

3.1 Introduction

As we mentioned before, the flooding process used for locating peers and data in unstructured P2P systems has excellent response time, is very simple to implement, and is very robust in the face of churn. However, it creates a large volume of unnecessary traffic, mainly because each node may receive the same query several times through different paths. In this Chapter we shall describe an innovative technique, the feedback-based approach that aims to improve the scalability of flooding. The main idea behind our algorithm is to monitor the ratio of duplicate messages transmitted over each network connection, and not forward query messages over connections whose ratio exceeds some threshold. Through extensive simulation we show that this algorithm exhibits significant reduction of traffic in random and small-world graphs, the two most common types of graph that have been studied in the context of P2P systems, while conserving network coverage (i.e.: the percentage of all peers that received at least one flood message).

3.2 Related Work

Many algorithms have been proposed in the literature to alleviate the excessive traffic problem and to deal with the traffic/coverage trade-off [11]. One of the first alternatives proposed was random walk. Each node forwards each query it receives to a single neighboring node chosen at random. This propagated message is called a “walker”. In this case the TTL parameter designates the number of hops the walker should propagate. Random walks produce very little traffic, just one query message per visited node, but either reduce considerably network coverage or have long response time. As an alternative multiple random walks have been proposed. The node that originates the query forwards it to k of its neighbors. Each node receiving an incoming query transmits it to a single randomly chosen neighbor. Although compared to the single random walk this method has better behavior, it still suffers from low network coverage and slow response time. Hybrid methods that combine flooding and random walks have been proposed in [5] by Gkantsidis et al.

In another family of proposed algorithms query messages are forwarded not randomly but rather selectively to part of a node’s neighbors based on some criteria or statistical information. For example, each node selects the first k neighbors that returned the most query responses, or the k highest capacity nodes, or the k connections with the smallest latency to forward new queries [6]. A somewhat different approach named forwarding indices proposed by Crespo [2] builds a structure that resembles a routing table at each node. This structure stores the number of responses returned through each neighbor on each one of a pre-selected list of topics. Other techniques include query caching, and the incorporation of semantic information in the network [19, 10, 13].

The specific problem we deal with in this work, namely the problem of duplicate messages, has been identified and some results appear in the literature. In [12] a randomized and a selective approach is adopted by Zhuang et al, and each query message is sent to a portion of a node's neighbors. The algorithm is shown to reduce the number of duplicates and to maintain network coverage. However, the performance of the algorithm is demonstrated on graphs of limited size. In another effort to reduce the excessive traffic in flooding, the authors [5] proposed to direct messages along edges which are parts of shortest paths. They rely on the use of PING and PONG messages to find the edges that lie on shortest paths. However, due to PONG caching this is not a reliable technique. Furthermore, their algorithm degenerates to simple flooding for random graphs, meaning that in this case no duplicate messages are eliminated.

Finally, in [8] Ripenau et al proposed to construct a shortest paths spanning tree rooted at each network node. However, this algorithm is not very scalable since the state each network node has to keep is in the order of O(N*d), where N is the number of network nodes and d its average degree.

3.3 Algorithm Description

The basic idea of the feedback-based algorithm is to identify edges on which an excessive number of duplicates are produced and to avoid forwarding query messages over these edges. In the algorithm's warm-up phase, during which flooding is used, a feedback message is returned to the upstream node for each duplicate message. The objective of the algorithm is to count the number of duplicates produced on each edge during this phase and subsequently, during the

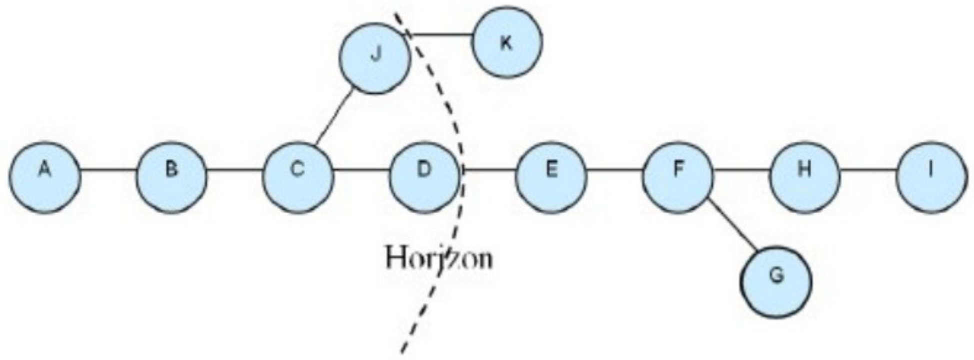

Figure 3.1: Illustration of the horizon criterion for node A and for horizon value 3.

execution phase, to use this count to decide whether to forward a query message over an edge or not.

In a static graph, a query message transmitted over an edge is a duplicate if this edge is not on the shortest path from the origin to the downstream node. One of the key points in the feedback-based algorithm is the following: Each network node A forms groups of the other nodes, and a different count is kept on each one of A's incident edges for duplicate messages originating from nodes of each different group. The objective is for each node A to group together the other nodes so that messages originating from nodes of the same group either produce many duplicates or few duplicates on each one of A’s incident edges. An incident edge of a node A that produces only a few duplicates for messages originating from nodes of a group must belong to many shortest paths connecting nodes of this group to the downstream node. An incident edge of node A that produces many duplicates for messages originating from nodes of a group must belong to few shortest paths connecting nodes of this group to the downstream node. Notice that if all duplicate messages produced on an edge were counted together (independent of their origin), then the algorithm would be inconclusive. In this case the duplicate count on all edges would be almost the same since each node would receive the same query though all of its incident edges. The criteria used by each node to group together the other nodes are critical for the algorithm's performance and the intuition for their choice is explained below.

A sketch of the feedback-based algorithm is the following:

- Each node A groups together the rest of the nodes according to some criteria.
- During the warm-up phase, each node A keeps a count of the number of duplicates on each of its incident edges, originating from nodes of each different group.
- Subsequently, during the execution phase, messages originating from nodes of a group are forwarded over an incident edge e of node A, if the percentage of duplicates for this group on edge e during the warm-up phase is below a predefined threshold value.

Two different grouping criteria, namely, the hops criterion and the horizon criterion, as well as a combination of them, horizon and hops, are used that lead to three variations of the feedback-based algorithm:

- Hops criterion: Each node A keeps a different count on each of its incident edges for duplicates originating k hops away (k ranges from 1 up to the graph diameter). The intuition for this choice is that, as we will see below, in random graphs small hops produce few duplicates and large hops produce mostly duplicates. Thus, messages originating from close by nodes are most probably not duplicates while most messages originating from distant nodes are duplicates. In order for this grouping criterion to work each query message should store the number of hops traversed so far.

Hops	1	2	3	4	5	6	7
Groups of nodes formed by node A	B	C	D,J	E,K	F	G,H	I

Table 1: Groups of the Hops criterion based on the example of example of Fig. 12.

- Horizon criterion: The horizon is a small integer, smaller than the diameter of the graph. A node is in the horizon of some node A if its distance in hops from A is less than the horizon value, while all other nodes are outside A's horizon (Fig. 3.1). For each node inside A's horizon a different count is kept by A on each of its incident edges. Duplicate messages originating from nodes outside A's horizon are added up to the count of their entry node in A's horizon. For example, in Fig. 3.1, duplicates produced by queries originating from node K are added up to the counters kept for node J, while duplicates produced by queries originating from nodes E, F, G, H, I are added up to the counters kept for node D. The intuition for the choice of this criterion is that shortest paths differ in the first hops and when they meet they follow a common route. For this criterion to be effective, a message should store the identities of the last k nodes visited, where k is the horizon value.
- Horizon+Hops criterion: This criterion combines the two previous

Feedback-based algorithm using the Hops criterion

1. Warm-up phase
   a) Each incoming non-duplicate query message is forwarded to all neighbours except the upstream one. For each incoming duplicate query message received, a duplicate feedback message is returned to the upstream node.
   b) Each node A, for each incident edge e, counts the percentage of duplicate feedback messages produced on edge e for all queries messages originating k hops away. Let us denote this count by D(e,k)
2. Execution phase: Each node A forwards an incoming non-duplicate query message that originates k hops away over its incident edges e if the count D(e,k) does not exceed a predefined threshold.

Figure 3.2: The feedback-based algorithm with the Hops criterion

Next, we present three variations of the feedback-based algorithm that are based on the grouping criteria used. The algorithm using the hops criterion is shown in Fig. 3.2. For the hops criterion to work each query message needs to store the number of hops traversed so far. The groups formed by node A in the graph of Fig. 3.1 according to the hops criterion are shown in Table 1.

The algorithm using the horizon criterion is shown in Fig. 3.3. For the horizon criterion to work each query message needs to store the identity of the last k nodes visited. The groups formed by node A in the graph of Fig. 3.1 according to the horizon criterion are shown in Table 2.

Node in A's horizon	B	C	D	J
Groups of nodes formed by node A	B	C	D,E,F,G,H,I	J,K

Table 2: Groups of the Horizon criterion based on the example of example of Fig. 12.

The algorithm using the combination of the two criteria described above, namely the Horizon+Hops, is shown in Fig. 3.4. For this criterion each message should store the number of hops traversed and the identity of the last k nodes visited. The groups formed by node A in Fig. 3.1 for the horizon+hops criterion are shown in Table 3.

We should emphasize that in order to avoid increasing the network traffic due to feedback messages, a single collective message is returned to each upstream node at the end of the warm-up phase.

Feedback-based algorithm using the Hops+Horizon criterion

3. Warm-up phase

   α) Each incoming non-duplicate query message is forwarded to all neighbours except the upstream one. For each incoming duplicate query message received, a duplicate feedback message is returned to the upstream node.

   β) Each node A, for each incident edge e,counts the percentage of duplicate messages produced on edge e for all queries messages originating from a node B inside A's horizon, or which entered A's horizon at node B and originated k hops away. Let us denote this count by D(e,B,k) .

2. Execution phase: Execution phase: Each node A forwards an incoming non-duplicate query message that originates k hops away over its incident edges e if the count D(e,B, k) does not exceed a predefined threshold.

Figure 3.4: The feedback-based algorithm with the Hops criterion

Hops	B	C	D					J	
Groups of nodes formed by node A	1	2	3	4	5	6	7	3	4
	B	C	D	E	F	G,H	I	J	K

Table 3: Groups of the Hops+Horizon criterion based on the example of example of Fig. 12.

3.4 Experimental Results on Static Graphs

Our evaluation study was performed using sP2Ps (simple P2P simulator) developed at our lab. The experiments were conducted on graphs with 2000 nodes and average degree of 6. The clustering coefficient (CC) ranged from 0.0001 to 0.6, which is the maximum clustering coefficient of a graph with N = 2000 and d = 6 (obtained with p=0). We shall refer to CC values from now on, as percentages of that max value. We conducted experiments for different values of the algorithm's parameters. The horizon value varied from 0 (were practically the horizon criterion is not used) up to the diameter of the graph. Furthermore, we used two different threshold values, namely 75% and 100%, to select the connections over which messages are forwarded. For example a threshold of 75% indicates that if the percentage of duplicates on an edge e during the warm up phase exceeds 75% for messages originated at the nodes of a group, in the execution phase no query message from this group is forwarded over edge e. The TTL value is set to the diameter of the graph.

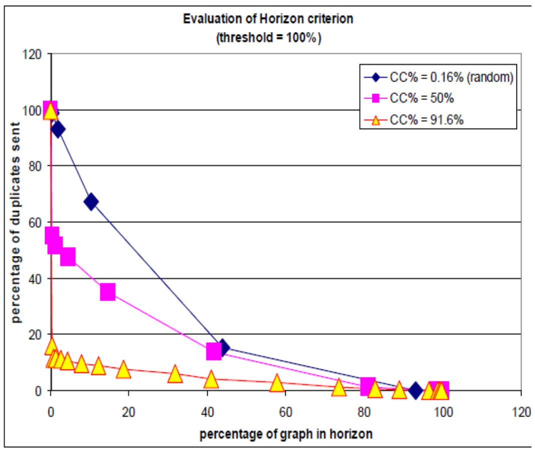

Figure 3.5: Percentage of duplicates as a function of the percentage of graph nodes in the horizon for three graphs with clustering coefficients 0.16, 50, and 91.6, and threshold value 100%.

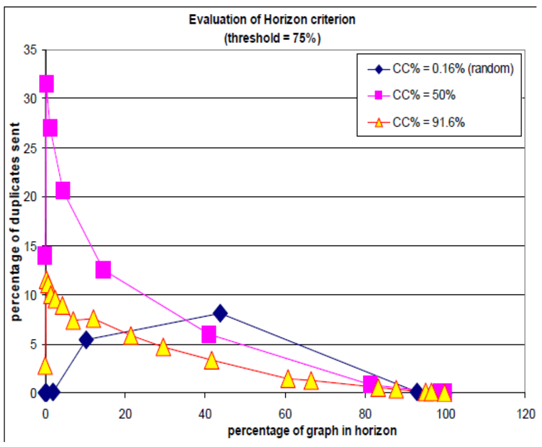

Figure 3.6: Percentage of duplicates as a function of the percentage of graph nodes in the horizon for three graphs with different clustering coefficients (0.16, 50, and 91.6) and threshold value 75%.

The efficiency of our algorithm is evaluated based on two metrics: (1) the percentage of duplicates sent by the algorithm, compared to the naive flooding approach and (2) the network coverage (defined as the percentage of network nodes reached by the query). Thus, the lower the duplicates percentage and the higher the coverage percentage is, the better. Notice that a threshold value of 100% indicates that messages originating from the nodes of a group are not forwarded only over edges that produce exclusively (100%) duplicates for all nodes of that group during the warm-up phase. In this case we do not experience any loss in network coverage, but the efficiency of the algorithm in duplicate elimination could be limited. In all experiments on static graphs, the warm-up phase included one flooding from each node. In the execution phase, during which the feedback-based algorithm is applied, again one flooding is performed from each node in order to gather the results of the simulation experiment.

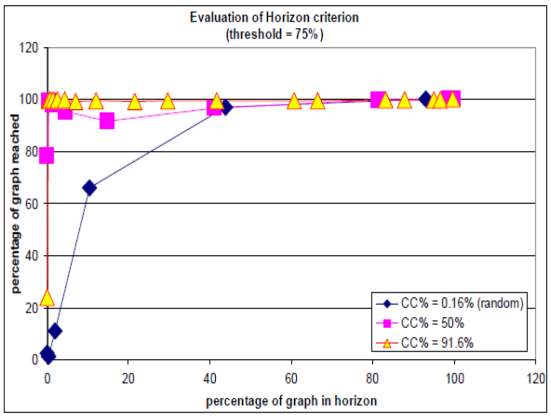

Figure 3.7: Network coverage as a function of the percentage of graph nodes in the horizon for three graphs with clustering coefficients 0.16, 50, and 91.6 and threshold 75%.

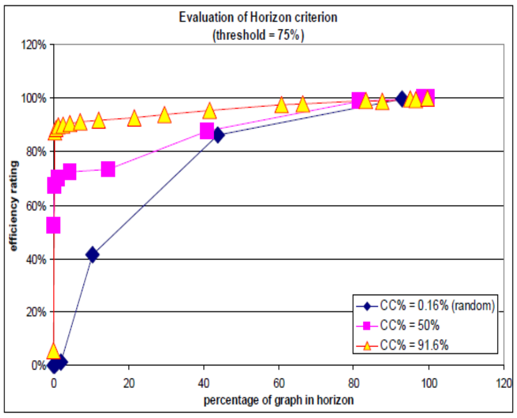

Figure 3.8: Efficiency of the feedback based algorithm as a function of the percentage of graph nodes in the horizon for three graphs with clustering coefficients 0.16, 50, and 91.6 and threshold 75%.

In Figs 3.5 to 3.9 we can see the experimental results for the feedback-based algorithm with the horizon criterion. In Fig. 3.5 we can see the percentage of duplicates produced as a function of the percentage of graph nodes in the horizon for three graphs (random with CC% = 0.16, clustered with CC% = 50, and small-world with CC% = 91.6) and for threshold value 100%, which means that there is no loss in network coverage. We can deduce from this figure that the efficiency of this algorithm is high for clustered graphs and increases with the percentage of graph nodes in the horizon. Notice that in clustered graphs, with a small horizon value a larger percentage of the graph is in the horizon as compared to random graphs. In Fig. 3.9 we plot the percentage of duplicates produced by the algorithm as a function of the clustering coefficient for horizon value 1 and threshold 100%. We can see that even for such a small horizon value the efficiency of the algorithm increases linearly with the clustering coefficient of the graph. We can thus conclude that the feedback-based algorithm with the horizon criterion is efficient for clustered and small-world graphs.

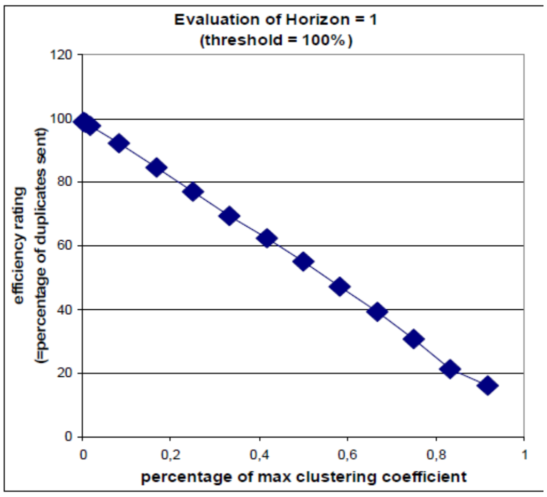

Figure 3.9: Percentage of duplicates as a function of the clustering coefficient for horizon value 1 and threshold value 100%.

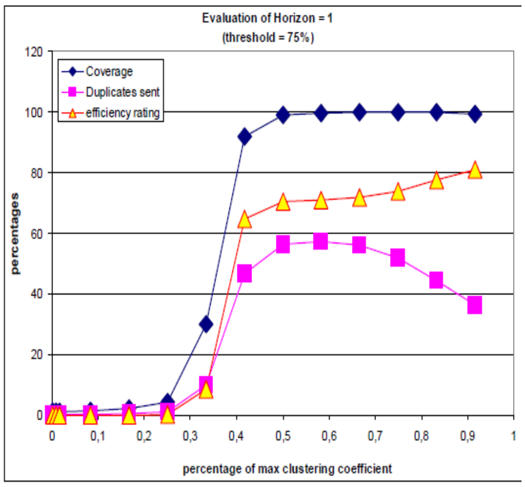

Figure 3.10: Network coverage, percentage of duplicates, and efficiency as a function of the clustering coefficient for horizon value 1 and threshold 75%.

Even if the percentage of graph nodes in the horizon decreases, in case the graph size increases and the horizon value remains constant, the efficiency of the algorithm will remain unchanged, because in clustered graphs the clustering coefficient does not change significantly with the graph size. Thus, the horizon criterion is scalable for clustered graphs. In contrast, in random graphs, in order to maintain the same efficiency as the graph size increases, one would need to increase the horizon value, in order to maintain the same percentage of graph nodes in the horizon. Thus the horizon criterion is not scalable on random graphs.

Figs 3.6 to 3.8 show the efficiency of the algorithm with the horizon criterion in duplicate elimination for threshold 75%. In Fig. 3.6 and 3.7 we can see that the algorithm is very efficient on clustered graphs. From the same figures we can see that with this threshold value in random graphs (CC% = 0.16) most duplicate messages are eliminated but there is loss in network coverage. Thus, even if we lower the threshold value, the horizon criterion does not work well for random graphs. The algorithm's behaviour is summarized in Fig. 3.8, where duplicate elimination, denoted by D, and network coverage, denoted by C, are combined into one simple metric, defined as C^2D, which we will call “efficiency”. We decided to raise the coverage metric to the square to denote its

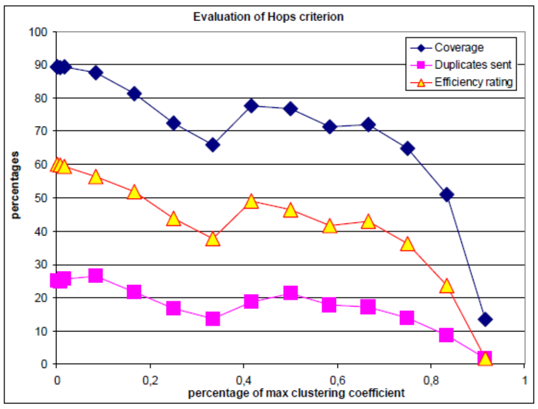

Figure 3.11: Network coverage, percentage of duplicates, and efficiency of the algorithm with the hops criterion as a function of the clustering coefficient.

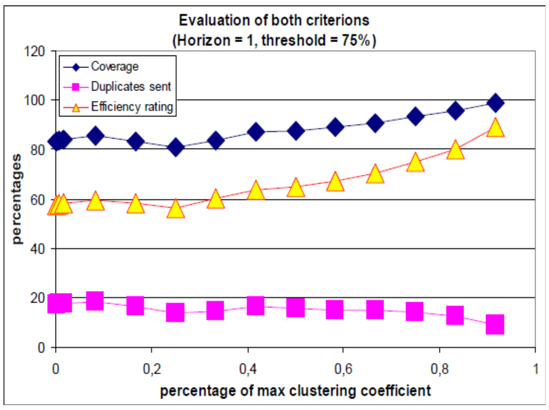

Figure 3.12: Network coverage, percentage of duplicates, and efficiency of the algorithm with the horizon+hops criterion as a function of the clustering coefficient.

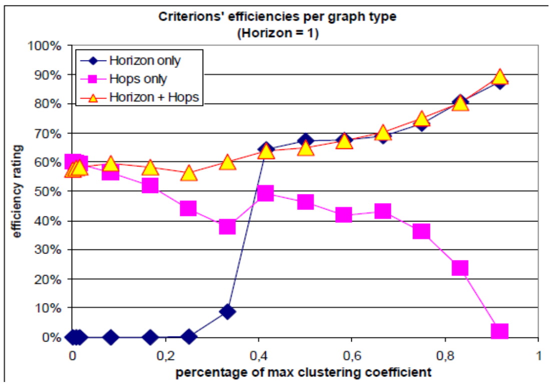

Figure 3.13: Efficiency of algorithms with the horizon, hops, and horizon+hops criteria as a function of the clustering coefficient and for horizon value 1.

increased importance over duplicate elimination.

In Fig. 3.10 we can see again the efficiency of the algorithm for horizon value 1 (as in Fig. 3.9) but for a threshold of 75%. Notice that the algorithm’s efficiency is not linear to the percentage of the clustering coefficient of the graph. This arises because the threshold value of 75% is not necessarily the best choice for any clustering coefficient.

In Fig. 3.11 we can see the experimental results for the algorithm with the hops criterion for a graph with 2000 nodes and average degree 6 while varying the clustering coefficient. We can see in this figure that the hops criterion is very efficient in duplicate elimination, while maintaining high network coverage, for graphs with small clustering coefficient. This means that this criterion exhibits very good behaviour on random graphs. As the clustering coefficient increases, the performance of the algorithm with the hops criterion decreases. This behaviour can be easily explained from Fig. 2.8, where the percentage of duplicates per hop is plotted for random and small-world graphs. We can see from this figure that in random graphs, the small hops produce very few duplicates, while large hops produce too many. Thus, based on the hops criterion only, we were able to eliminate a large percentage of duplicates without greatly sacrificing network coverage.

As mentioned before, the hops criterion works better for random graphs. In case the graph size increases, the number of hops also increases (recall that the diameter of a random graph with N nodes and average degree d is log(N)/log(d)). Thus, the hops criterion is scalable on random graphs.

In Fig. 3.12, we see the efficiency of the algorithm for the horizon+hops criterion. As we can see from this figure this combination of criteria constitutes the feedback based algorithm efficient in graphs with all clustering coefficients, random and small-world. In Fig. 3.12, three different metrics are plotted, the network coverage, the percentage of duplicates, and the efficiency as a function of the clustering coefficient of the graph. We can see that for any clustering coefficient network coverage is always above 80%, while the percentage of duplicate messages not eliminated is always less than 20%. This behaviour is achieved for random and small-world graphs for horizon value of only 1. Thus the horizon+hops criterion is scalable on all types of graphs.

In Fig. 3.13 we compare the efficiencies of the hops, horizon, and horizon+hops and we see that their combination, horizon+hops works better than each criterion separately.

3.5 Experimental Results on Dynamic Graphs

In what follows, we introduce dynamic changes to the graph, meaning that a graph node can leave and some other node can enter the graph, and we monitor how these changes influence the algorithm's efficiency. We introduced a new parameter to our experiments in order to capture the rate of graph change. This parameter measures in query-floods the lifetime of a node in the graph. A graph rate change of r means that each node will initiate, on the average, r query-floods before leaving the network. Insertion of new nodes is performed so as to preserve the clustering coefficient of the graph.

We also introduce a dynamic way to determine when the warm-up phase can terminate, meaning that we have collected enough measurements. The warm-up phase for a group of nodes terminates after the percentage of duplicates seen on an edge for messages originating from nodes of the group stops to oscillate significantly. More specifically, the warm-up phase terminates on an edge for a group of nodes, if in each of the last 20 rounds the change in the count (percentage of the number of duplicates seen on that edge for messages originating from nodes of the that group) was smaller that 2% and the total change over the last 20 rounds was smaller that 1%.

We perform experiments for random graphs and for small-world graphs with clustering coefficient CC% = 33 and CC% = 84. For each of these graphs, the value of the change rate equals 0 (static graph), 1, 50, and 200. A change rate of 200 indicates that each node will make 200 query-floods before leaving the network, which is a reasonable assumption for Gnutella 2 [7]. This is because each Ultrapeer contains, on the average, 30 leaves. A leaf node has in general much smaller average lifetime than an Ultrapeer, which means that each Ultrapeer will "see" more than 30 unique leaves in its lifetime. If we assume that each leaf node will send one query through the Ultrapeer, this explains the fact that real-world measures with an Ultrapeer show that each Ultrapeer sends about

100 queries per hour. For each of these graphs and change rates, we run experiments with the following Horizon values:

- Horizon values = 1—2 for random graphs and for small-world graphs with CC% = 33.
- Horizon values = 1—4 for small-world graphs with CC% = 84.

We performed two experiments with the same horizon value, one using the hops criterion and one without the hops criterion. The threshold value was set to 75%. Each experiment performed 25*2000 floods. The difference between the values "0 wo act. threshold" and "0 with act. threshold" in the x axis in the figures indicates that in both cases the change rate is 0 (static graph), but in the first case, the numbers are taken from the experiments described in the previous section, while in the second case the activation threshold was used to terminate the warm-up phase. This enables us to clearly see the benefit of the activation threshold.

Fig. 3.14 shows how the algorithm performs on dynamic graphs for the horizon criterion. We should first note that the use of the activation threshold increases the efficiency of the algorithm significantly. This happens because nodes gradually start eliminating traffic for certain groups of nodes instead of all of them starting eliminating duplicates for all groups simultaneously.

We can see that the efficiency of the algorithm decreases when the change rate is 1. The main reason for this is not that the measurements for each group quickly become stale, but rather because each node needs some warm-up period to learn the topology of the network. A certain amount of traffic needs to be "seen" by any node, to make the necessary measurements. If that time is a large fraction of the node's lifetime, it means that it will spend most of its time measuring instead of regulating traffic according to the measurements.

Finally and most importantly, we can see that the results for a change rate of 200 are the same as those of a change rate of 0 with activation threshold, which shows that, given that the warm-up phase is shorter than the time during which the nodes use the algorithm (execution phase), the changes of the graph do not affect the algorithm's efficiency.

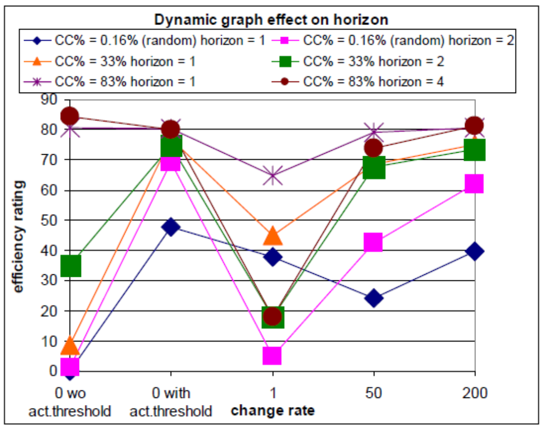

Figure 3.14: Performance (efficiency) of the algorithm on a dynamic graph for the horizon criterion.

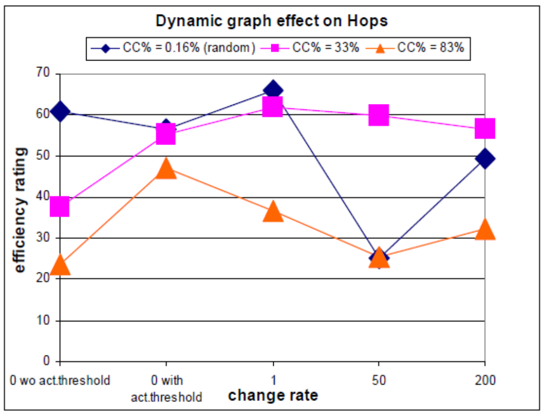

Figure 3.15: Performance (efficiency) of the algorithm on a dynamic graph for the hops criterion.

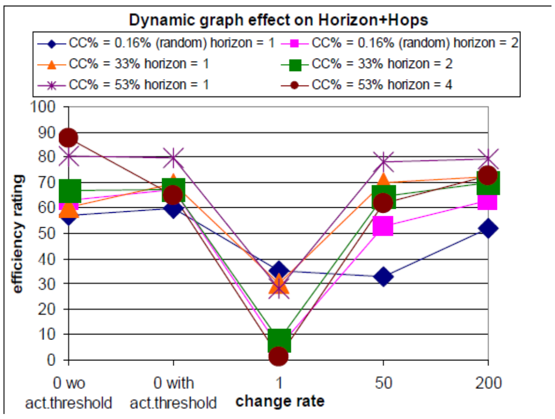

Figure 3.16: Performance (efficiency) of the algorithm on a dynamic graph for the horizon + hops criterion.

In Fig. 3.15 we can see that the activation threshold is beneficial to the algorithm with the hops criterion. Furthermore, from the same figure, it becomes clear that the efficiency of the feedback-based algorithm with the hops criterion is not greatly affected by the dynamic changes in the graph. We should however point out that it seems to lightly affect the efficiency of the algorithm in highly clustered graphs.

In Fig. 3.16 we see the efficiency of the algorithm for the horizon+hops criterion. We should notice again that the use of the activation threshold does not harm the algorithm, except in the case of the graph with high clustering coefficient and for a horizon value greater than 1. However, as we have seen before, there is not reason to use a horizon value larger than 1. Again, the change rate does not affect the measurements for groups of nodes, since the reason for the low efficiency at high change rates is the fact that the nodes spent most of their lifetime in the warm-up phase.

3.6 Summary

We presented the feedback-based algorithm, an innovative method which reduces significantly the number of duplicates produced by flooding while maintaining high network coverage. The algorithm monitors the percentage of duplicates on each connection during a warm-up phase, and directs traffic to connections that do not produce excessive number of duplicates during the execution phase. In order for this approach to work, each network node groups together the rest of the nodes according to some criteria, so that nodes that produce many duplicates on its incident edges are in different groups than those that produce only few duplicates. The efficiency of the algorithm was demonstrated through extensive simulation on random and small-world graphs. The experiments involved graphs of 2000 nodes. The feedback-based algorithm was shown to reduce to less than 20% the number of duplicates of flooding while conserving network coverage above 80%. The memory requirements in each node are much less compared to the algorithm that constructs shortest paths trees from each network node. The efficiency of our algorithm was demonstrated on static and dynamic graphs.

Chapter 4

Partitioning Unstructured P2P Systems to Improve Resource Location

4.1 Introduction

The aim of the work presented in this Chapter is to improve the scalability of flooding by reducing the number of peers that need to be contacted on each request (and thus the traffic cost in messages, be it duplicate or not) without decreasing the probability of query success (accuracy of the search method). The proposed method partitions the Ultrapeer overlay network into distinct subnetworks. Using a simple hash-based categorization of keywords the Ultrapeer overlay network is partitioned into a relatively small number of distinct subnetworks. In general unstructured P2P networks are indirectly supplied with some information about the possible location of each resource. By employing a novel index splitting technique each Leaf peer is effectively connected to each different subnetwork. The search space of each individual flooding is restricted to a single partition, thus the search space is considerably limited. This reduces the overwhelming volume of traffic produced by flooding without affecting at all the accuracy of the search method (network coverage). Experimental results demonstrate the efficiency of the proposed method.

4.2 Related Work

In an effort to alleviate the large volumes of unnecessary traffic produced during flooding several variations have been proposed. Schemes like Directed Breadth First Search (DBFS) [25] forward requests only to those peers that have often provided results to past requests, under the assumption that they will continue to do so. Interest-based schemes, like [23] and [18] aim to cluster together (make neighbourhoods of) peers with similar content, under the assumption that those peers are better suited to provide to each other's needs. Both those systems try to contact peers that have a higher probability of containing the re-quested information. Such schemes usually exhibit small gains over traditional flooding.

Another approach that has been used in the literature to make resource location in unstructured P2P systems more efficient is the partitioning of the overlay network into subnetworks using content categorization methods. A different subnetwork is formed for each content category. Each subnetwork connects all peers that posses files belonging to the corresponding category. Subnetworks are not necessarily distinct. A system that exploits this approach is the Semantic Overlay Networks (SONs) [19]. SONs use a semantic categorization of music files based on the music genre they belong to. The main drawback of this method is the semantic categorization of the content. In file -sharing systems for instance, music files rarely contain information about the genre they belong to and when they do so, each of them probably uses a different categorization of music. In SONs, an already existing, online, music categorization database is used. This database adds a centralized component in the operation of the net-work. Notice that 1-hop replication can be employed in conjunction with this scheme, inside each subnetwork. However, the fact that each peer may belong to more than one subnetwork, reduces the average degree of each subnetwork and thus, the efficiency of the 1-hop replication.

4.3 The Partitions Design

The system we propose in this Chapter allows for the partitioning of any type of content. More specifically, we propose the formation of categories based on easily applicable rules. Such a simple rule is to apply a uniform hash function on each keyword describing the files. This hash function maps each keyword to an integer, from a small set of integers. Each integer defines a different category. We thus categorize the keywords instead of the content (files) itself. Given a small set of integers, it is very likely that each peer will contain at least one keyword from each possible category.

Unstructured P2P systems like Gnutella 2 [14] employ a 2-tier structure. In those systems Ultrapeers form a random overlay network, while Leaf nodes are connected to Ultrapeers only. Each Leaf sends to the Ultrapeers it is connected to, its index in the form of a (compressed) bloom filter. Ultrapeers flood queries to the overlay network on the Leave's behalf. Flooding is only performed at the Ultrapeer level where 1-hop replication is implemented. Whenever an Ultrapeer receives a request this is specifically forwarded only down to those Leaves that contain the desired information (except in the case of false positives). Fig. 2.4 shows a schematic representation of the 2-tier architecture.

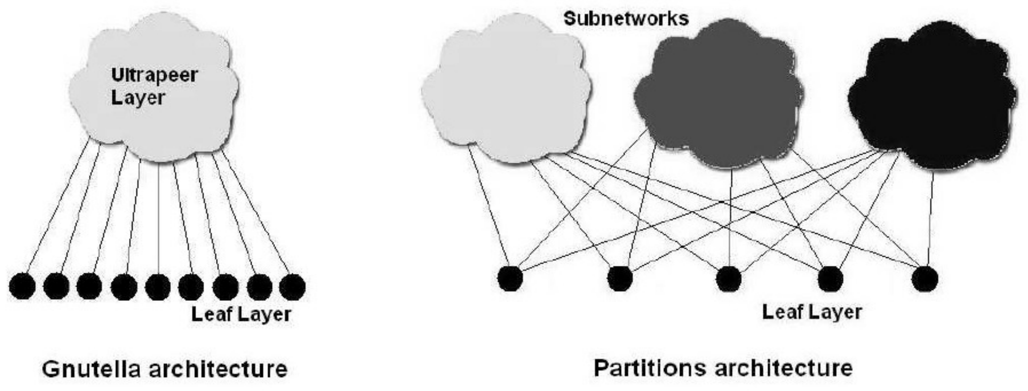

Figure 4.1: Illustration of the Gnutella network and the Partitions design.

The keyword categorization method is used in 2-tier unstructured systems. In the Partitions design, each Ultrapeer in the system is randomly and uniformly assigned responsibility for a single keyword category, by randomly selecting an integer from the range set of the hash function used to categorize the key-words. Ultrapeers responsible for the same category form a distinct subnetwork. Leaves connect to one Ultrapeer per subnetwork and send to it all the keywords belonging to that category. Thus, an innovative index splitting technique is used. Instead of each Leaf sending its entire index (in the form of a bloom filter) to an Ultrapeer, each Leaf splits its index (keywords) based on the defined categories and distributes it to one Ultrapeer per category. Notice that peers operating as Ultrapeers also operate as Leaves at the same time (have a dual role), since they connect as Leaves to Ultrapeers of other subnetworks, in order to publish their content. Even though in this design each Leaf connects to more than one Ultrapeers, the volume of information it transmits is roughly the same since each part of its index is sent to a single Ultrapeer. Each Leaf node sends to the Ultrapeer of a certain category all keywords that belong to the same category (in the form of a bloom filter). Each Ultrapeer sends to its neighbouring Ultrapeers all the aggregate indices of its Leaf nodes to implement 1-hop replication. In Fig. 4.3 we can see a schematic representation of the Partitions design.

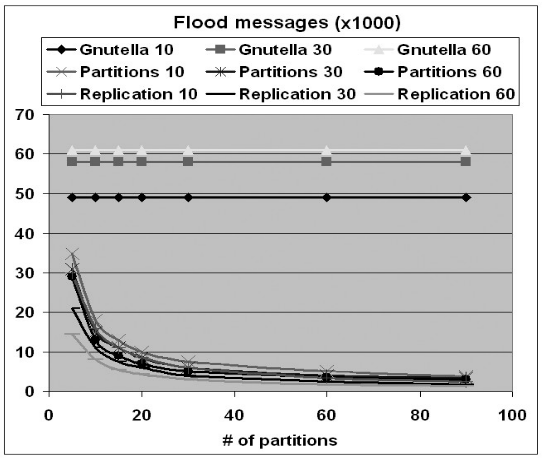

Figure 4.3: Total number of messages per flood for keyword-based searches

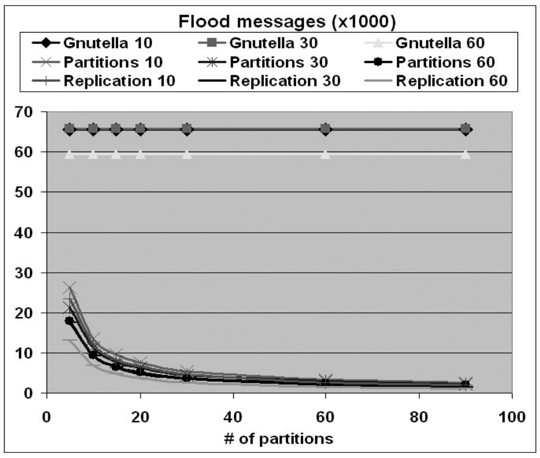

Figure 4.2: Total number of messages per flood for hash-based searches

This separation of Ultrapeers from content has the benefit of allowing them to be responsible for a single keyword category. The benefit of this is two-fold. First, it reduces the size of the subnetworks since they are completely discrete (at least on the overlay level). Secondly, it allows each Ultrapeer to use all its Ultrapeer connections to connect to other Ultrapeers of the same subnetwork, increasing the efficiency of 1-hop replication at the Ultrapeer level.

There are, however, two obvious drawbacks to this design. The first one is due to the fact that each Leaf connects to more than one Ultrapeers, one per content category. Even though each Leaf sends the same amount of index data to the Ultrapeers upon connection as before, albeit distributed, however it requires more keep-alive messages to ensure that its Ultrapeers are still operating. Keep-alive messages however are very small compared to the average Gnutella protocol message. In addition, query traffic is used to indicate liveliness most of the time, thus avoiding sending keep-alive messages. The second drawback arises from the fact that each subnetwork contains information for a specific keyword category. Requests however may contain more than one keywords and each result should match all of them. Since each Ultrapeer is aware of all keywords of its Leaves that belong to a specific category, it may forward a request to some Leaf that contains one of the keywords but not all of them. This

	Ratio
No replication	4.2
Replication	5.5

Table 4: Flooding Efficiencies

fact reduces the efficiency of the 1-hop replication at the Ultrapeer level and at the Ultrapeer to Leaf query propagation. This drawback is balanced in two ways. The first is that even though the filtering is performed using one keyword only, Leaves' bloom filters also contain one type of keywords only, making them more sparse and thus reducing the probability of a false positive. Furthermore, the most rare keyword can be used to direct the search, thus further increasing the effectiveness of the search method. Finally, we also experimented with sending the bloom filters with all keyword types to every Ultrapeer, regardless of category, although Ultrapeers still extract and use only keywords of the same category as their own to form their aggregate bloom filter in order to implement 1-hop replication.

All these schemes have varying degrees of maintenance costs which we explore in the next section using simulations.

4.4 Experimental Results

In this section, we shall present the results from the simulations we conducted, in order to measure both the efficiency of the Partitions scheme in terms of cost of flooding (in messages) and maintenance costs .

We assumed a peer population of 2 million, a number reported by LimeWire Inc [15]. Each Ultrapeer in the Gnutella network serves 30 Leaves, a number obtained from real-world measurements [24]. In addition, each peer

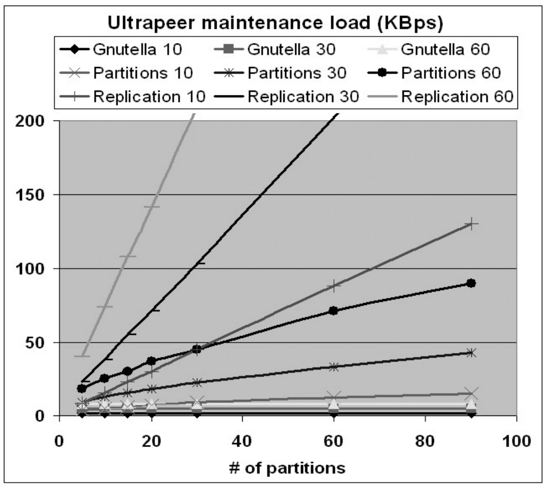

Figure 4.4: Ultrapeer maintenance load for keyword-based searches

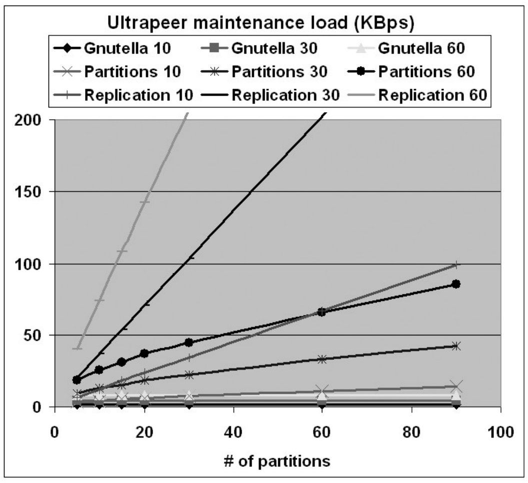

Figure 4.5: Ultrapeer maintenance load for hash-based searches

contains a number of files (and hence keywords) derived from a distribution also obtained from real-world measurements in [22].

Each Ultrapeer in the Partitions design serves 300 Leaves since we assume a number of 10 content categories and thus subnetworks. We perform a large number of floods, each designed to return at least a thousand query results before terminating. Table 4 shows the ratio of the average number of messages per flood for the Partitions design over the average number of messages per flood in Gnutella. Replication means that each Leaf sends all its keywords to all Ultrapeers it is connected to, regardless of category. For example, in the case of replication, flooding in the Partitions design generates 5. 5 times less messages than flooding in Gnutella, in order to return the same number of results per query. We can see that the drawback of filtering using only one keyword is balanced by the fact that the sparser Leaf indices (since they contain only one keyword category) produce less false positives, but mainly outweighed by the message reduction due to the partitioning of the network and therefore the reduction of the search space. We would like to emphasize that each Partitions bloom filter (i.e. containing keywords of a certain category) has the length of a

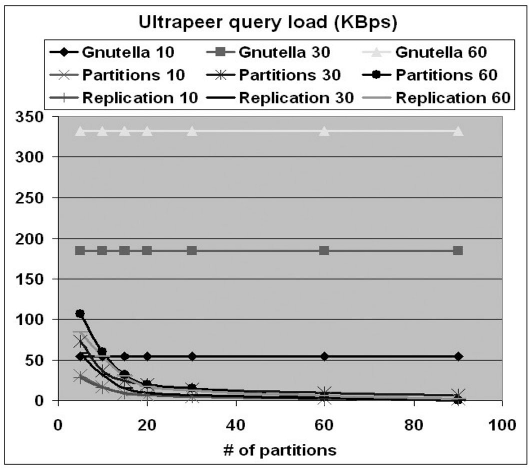

Figure 4.6: # of queries per second for keyword-based searches

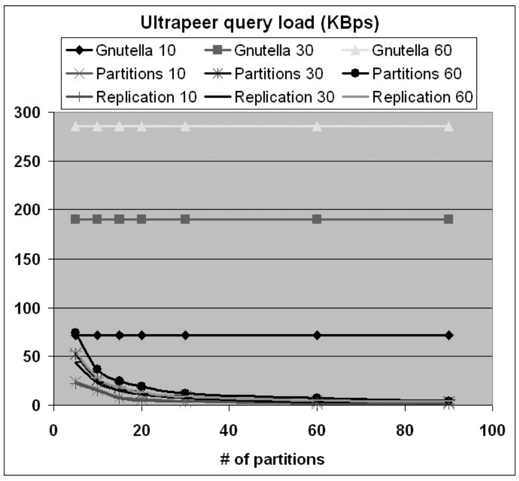

Figure 4.7: # of queries per second for hash-based searches

Gnutella bloom filter. Thus, one can roughly think of all the bloom filters of a single Partitions leaf as a (distributed) Gnutella bloom filter of 10 times the length (due to the 10 category types). However the bandwidth needed to transfer such a bloom filter is not 10 times that of a Gnutella bloom filter, mainly because sparser bloom filters are compressed more efficiently.

In order to measure the maintenance cost of Gnutella and Partitions, we focus on the operation of a single Ultrapeer, because the load of Leaves is negligible in both systems compared to a Ultrapeers load since flooding is performed at the Ultrapeer overlay. In both cases we simulated three hours in the life of a single Ultrapeer, with Leaves coming and going. Each time a Leaf is connecting to the Ultrapeer, it sends its index information, which is propagated by the Ultrapeer to its thirty Ultrapeer neighbours. In addition, we assumed that, periodically , each Ultrapeer receives a small keep-alive message from each Leaf and replies with a similar message to each one of them, unless a query and a reply were exchange during the specified period. For each communication taking place, we measured the incoming or outgoing traffic in bytes, in order to estimate the bandwidth requirements. For each Ultrapeer, we measure query load (traffic containing only flood messages and replies), maintenance load (overlay upkeep traffic) and traffic load (the sum of both).

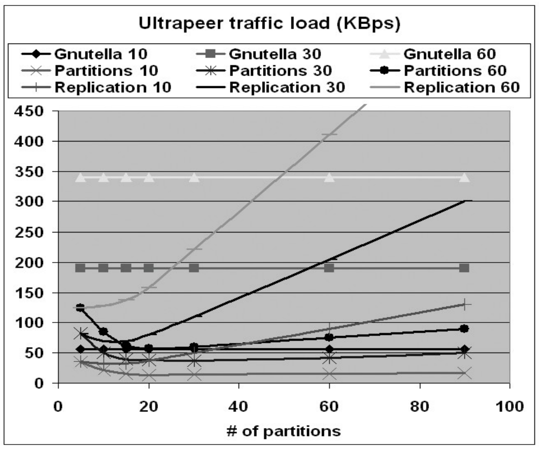

Figure 4.9: Ultrapeer aggregate traffic for keyword-based searches

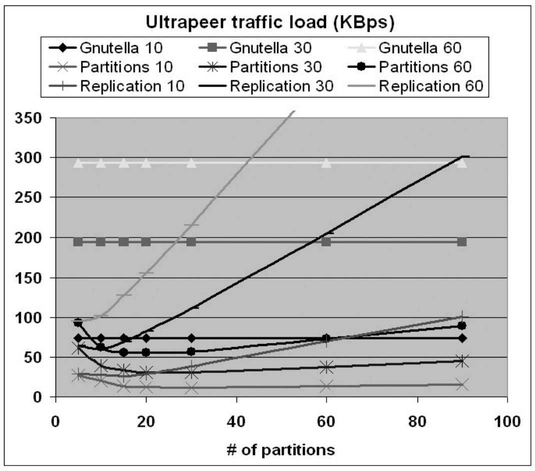

Figure 4.8: Ultrapeer aggregate traffic for hash-based searches

There are two modifications in this scenario, between Gnutella and Partitions. In Partitions, the number of Leaves is 300. In addition, the process of computing the size of the index information sent to the Ultrapeer differs greatly. In the case of Gnutella, we have used the code by LimeWire [15], the most popular Gnutella client, to construct the bloom filter of each Leaf. We first randomly decided on the number of files shared by each Leaf, based on the file sharing distribution per peer presented in [22]. We then extracted this number of files from a list of file-names obtained from the network by a Gnutella crawler developed in our lab. Those file-names were fed to the LimeWire bloom filter generation code, which produced the corresponding bloom filter in compressed form, i.e., the way it is sent over the network by LimeWire servents. Thus we constructed the actual bloom filter, although what we really need in this case is just its size. In the case of Partitions, we likewise computed the number of files to be shared by each Leaf. We extracted again the same number of file-names from the list of available file-names.

We subdivided the Partitions scheme depending on the form of the index information sent by Leaves to Ultrapeers. Two experiments were run with the Partitions scheme using bloom filters. In the first, each bloom filter sent to an Ultrapeer only contained appropriate keywords (of the same category as the corresponding Ultrapeer). In the second experiment, we used replication, i.e. each bloom filter contained all the keywords of the Leaf, regardless of category. In addition, positions of keywords of the corresponding category as the Ultrapeer were set in the bloom filter to the value of two in stead of one. (This bloom filter essentially distinguishes between keywords of the appropriate category and the rest of the categories).

Figs. 4.4 and 4.5 show the results of the simulation for the cost of maintaining the structures of Gnutella and Partitions, without any query (flood) traffic. From this figure it is obvious that, as expected, the maintenance cost of partitions is higher than that of Gnutella. As we will see in the next paragraph the gains incurred during the operational phase of the two systems outweighs the increased maintenance costs.

We then focused our attention to the query traffic load. Measurements showed that, on the average, each Ultrapeer generates 36 queries per hour (i.e., queries initiated by itself or its Leaves). This adds up to approximately 2000 queries per second generated anywhere in the Gnutella network. In addition, we observed a large number of Gnutella queries in order to find the distribution of the number of keywords in each query. Thus, according to those observations, during the simulations we assumed that 20% of the queries contain 1 keyword, 30% contain two, another 20% contain three and finally a 30% contain 4 keywords. In addition, we performed the same experiments, this time assuming hash-based searches. File-sharing applications allow the search of a file, based on the hash value of its contents, in order to locate copies of the same file. This enables multi-source transfer of the file. These searches are, in essence, 1-keyword searches. All types of graphs in this Chapter come in pairs, describing the same experiment, one assuming keyword-based queries and the other assuming hash-based queries.

In our simulation, we assumed that the aim of each flood (both in Gnutella and Partitions) is to reach the entire network, or produce a fixed number of results, whichever comes first. As we mentioned before, such a flood that aims to reach the entire network would need to reach 1/10th of the Gnutella's network (or a Partitions' subnetwork) during all hops of flooding except the last. This means that the Ultrapeer in our simulations has a probability of 0.1 to receiving each query. In addition, every time this does not occur, it has another opportunity to receive the query during the last hop, depending on its bloom filter (in case the searched keywords match in the bloom filter). Should the Ultrapeer receive a query, it is assumed to propagate it to its Leaves, again depending on their bloom filters or index (again depending on a possible keyword match by the bloom filter). Figs. 4.8 and 4.9 show the comparison in the traffic load of Gnutella and Partitions, including maintenance and query traffic. We used a size of 40 bytes for each query. In reality, the size of a query can be up to a few hundred bytes, if XML extensions are used. This means that the performance gains described here are smaller compared to the ones we expect to see in the real world. In addition, for every 1400 bytes for each message sent, we added 40 bytes for the TCP and IP header. From these figures it is evident that Partit ions outperform Gnutella in operational costs, in every case. Finally in Figs. 4.6 and 4.7 one can see the query traffic load alone (without the maintenance traffic) for both the Gnutella and the Partitions Ultrapeer.

4.5 Summary

In this Chapter, we have described a novel approach to reducing the message costs of querying in unstructured networks. The method exploits the partitioning of random overlay networks into a small number of distinct subnetworks based on easily applicable rules. The method allows for the categorization of any type of content. Extensive simulations have been performed and demonstrated that the benefits obtained from our scheme can be as high as an order of magnitude compared to the Gnutella fooding.

Chapter 5

Innocuous Topology Awareness for Unstructured P2P Networks

5.1 Introduction

One of the most critical design aspects of any P2P system is the overlay layer, that is a virtual network of interconnected peers (P2P client-servers) through (and on top of) the underlying IP network. The structure of this overlay network is tightly coupled with the search algorithm, which is usually the main function of a P2P system. This means that the network structure is such as to enable and facilitate this function, which also means that there are rules governing which peers are connected to which peers.This is more apparent in the case of structured P2P systems, where the structure of the overlay network is such as to allow for a binary-tree like search to be performed, which requires a logarithmic (*O(logN)*) search cost in the number of messages.

On the other hand, unstructured systems, by definition, do not impose a specific structure on the overlay. Each peer is free to connect to any other (available) peer. Even though this lack of structure denotes a large degree of freedom in the creation of the overlay, we will show that this is misleading. Most mechanisms used widely in unstructured P2P systems today, actually rely on this random selection of neighbours (the peers to connect to), regardless of their distance and position in the IP network. This leads to a complete lack of correlation between the t wo respective distances (IP network and overlay), wherein lies the problem we aim to rectify.

So, either structured or unstructured, all P2P systems have their own design goals on overlay creation, which do not include taking into consideration the structure of the underlying physical network, the Internet. As a result, most P2P systems make an inefficient use of the IP layer, which has adverse impact not only on their own operation but also on the operation of the other applications, which co-exist on the same medium (the Internet). Some proposals have already been published, which aim to rectify this. Most of them rely on the freedom of peers in unstructured neighbours to connect to any peer they want, in order to create an overlay which better matches the IP layer. However, as we mentioned and will show, this freedom to choose any peer as neighbor is more of a requirement than actual freedom. This means that showing any preference, during neighbourhood selection, on peers depending on their position and distance in the IP network violates this requirement, and thus, we argue, these approaches greatly affect some of the most fundamental characteristics of P2P systems that we mentioned in Chapter 1.

The obliviousness of P2P systems to the underlying network has two main drawbacks. The first is that the average latency between any two neighbors on the P2P overlay is increased since each peer does not actively try to connect to peers which are closer at the IP layer and/or have smaller latency. The second and most important drawback is that the IP path behind each P2P overlay connection contains a large number of routers. This means that even a single, 1-hop, message between neighbors (at the P2P overlay) may travel through many routers and autonomous systems before it reaches its destination. Figure 5.1 illustrates such a simple scenario, where a message from peer A to peer C crosses the Atlantic twice before it reaches peer C on the same continent as peer A. This inefficient routing, is one of the main reasons behind the observed domination of

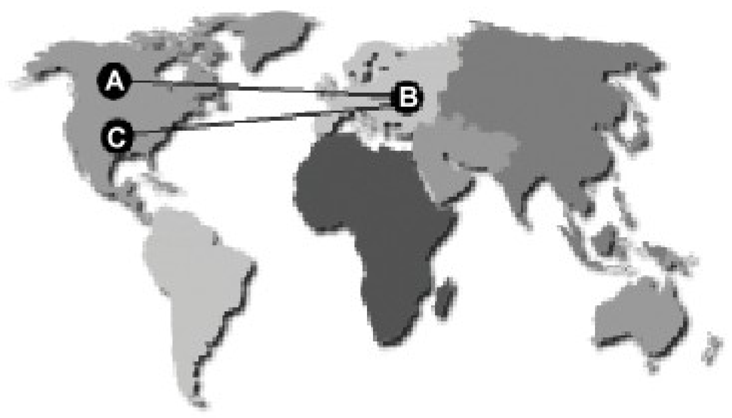

Figure 5.1: Illustration of inefficient routing in today's unstructured P2P systems

P2P traffic in the Internet [50], [51]. An obvious solution to this problem is to have each peer connect to those peers which are closest to itself, in terms of latency, while maintaining a small number of further links to avoid overlay partitioning. However this would create an overlay with a higher degree of structure (clustering), which will have a negative impact on the mechanisms employed in unstructured P2P networks.

In this Chapter, we aim to solve this canandrum. We propose ITA, an algorithm for *Innocuous Topology Aware* construction, which provides unstructured P2P overlay creation with a large degree of topology awareness, while at the same time taking into consideration the impact the proposed changes will have on the rest of the mechanisms employed in unstructured P2P systems. It is able to do so by building a random graph of random graphs, therefore preserving, in a sense, the random nature of the overlay, while at the same time allowing for the existence of ”neighborhoods”, allowing peers to randomly connect to nearby peers. We use a diverse set of metrics to experimentally evaluate out proposal and to give a complete view of its impact on the system's operation. The results we obtain include a 50% reduction in search latency, a 20% reduction in the number of IP messages and a significant (approx also 50%) reduction on the load of the IP network routers. ITA is shown to have no negative impact whatsoever on the 1-hop replication and the dynamic querying mechanisms.

5.2 Related Work

One of the main drawbacks of unstructured P2P systems is the limitation of their scalability due to the large number of messages generated by their search mechanism, called flooding. This is evident in the fact that a large part of the existing literature aims at reducing those messages [46], [53], [36], [32]. However, the vast majority of this work is concerned with reducing the number of the overlay messages, even though a single overlay message usually translates to several IP messages. This abstraction has been shown to be problematic for the network layer.

In the case of structured systems, some work has been carried out aiming to address this problem, even though the possibilities are limited since there are specific requirements for the neighbour selection of each peer. Due to the more rigid structure of those systems, one has less freedom on how to rewire the connections in the system to allow for greater topology awareness. In [33] the authors propose the selection of the closest (latency-wise) neighbour whenever there are more than one choices. This approach can be applied in systems like Pastry [49], Kademlia [44], and Tapestry [56]. However, in systems like Chord [54] and CAN [47], each neighbour is uniquely defined.

Our work focuses on unstructured systems, which are not as sensitive to changes in the overlay creation. Topology awareness algorithms that have been proposed for unstructured systems, such as [39], [41], aim at constructing a generic, topologically aware overlay, and thus do not describe any mechanism for efficiently searching on that overlay. In addition, the constructed graph has a high clustering degree, which predicates that the mechanisms already employed in unstructured P2P systems and which depend on a random overlay to function properly, will experience a high loss in efficiency. In particular, the authors of [39] describe an overlay graph creation method, which is based on having each peer connect to those other peers with which it has the longest common domain suffix. Some random links are also maintained in order to avoid the partitioning of the network. In addition to the drawbacks common to all approaches which increase topology awareness by reducing the randomness of the graph this approach has an additional disadvantage. The graph that is formed is comprised of neighborhoods of diverse sizes, since not all domains have the same peer population. This makes the choice for a universal value for the Time−To−Live (TTL) difficult. The same holds for the systems described in [40], [45], where the neighborhoods are defined by the IP addresses instead of the domain names. In flood-based P2P systems, the TTL value is critical for the efficient operation of the system and is directly connected not only its scalability but also its operational success. A TTL value which is appropriate for some of the neighborhoods can be inefficient for others, leading to either failure to locate content, or to the generation of a large number of duplicate messages. ITA constructs randomly connected “neighborhoods” of roughly equal size, which means that one TTL value “fits all”.

In [35], the authors use synthetic coordinates to create neighbourhoods of close-by, in terms of latency, peers. Their simulations were performed on a network which comprised of 92 IP-layer nodes and included 42 overlay peers. This small network size makes it difficult to reveal the real benefit of the algorithm. In addition, in experiments of this scale it would be difficult to notice the effect of the increased clustering in the flooding mechanisms. In [41], overlay creation is inspired by the *k*-median algorithm, in order to, again, construct neighborhoods of nearby, latency-wise peers and thus reduce the average latency of any path between any two peers in the overlay. This theoretical algorithm appears to be computationally expensive since it requires knowledge of the entire overlay topology to function. Furthermore, as the overlay changes from the departure and arrival of peers, the algorithm needs to continuously adjust the overlay in order to maintain its efficiency. The work described in [52] is a follow-up of [41]. The algorithm still needs to be active all the time to preserve the structure of the network. In addition, the main focus of this work is on the construction of an efficient graph for general use, as is the case for the work described in [42], [43], so there is no descrption on how to search the overlay. We focus on how to efficiently construct an overlay with low clustering that maintains the beneficial properties of random graphs and leads to efficient informati on lookup. Finally, an interesting work is presented in [55]. The method described limits the reorganization of the network to add topology awareness in a 2-hop neighborhood for each peer. ITA constructs the entire overlay from the beginning to allow for the desired topology-awareness.

As we mentioned, any method used to construct and the resulting structure of the overlay is tightly coupled with the other mechanisms at work in a P2P system. In existing P2P systems this is especially true for the mechanisms that comprise the search-lookup function. The works we just mentioned does not take into consideration the impact of the proposed methods on those widely deployed mechanisms such as 1-hop replication and dynamic querying. ITA functions without affecting them in any way, which means that there is no trade-off. Any increase in topology awareness comes at no-cost. In addition, most of the aforemention work requires that each peer continuously executes the topology-awareness algorithm to adopt to changes in the P2P overlay. This is mainly because most of the aforementioned proposed methods try to connect each peer to its closest possible neighbors. This set however changes dynamically in time, due to the churn in the network. ITA only requires a simple and quick bootstrapping process, after which it can continue to function unaffected by the churn of the system. Furthermore, this continuous operation of most of the aforementioned proposed methods requires each peer to continuously probe the network in case some new, closest peer has joined, imposing additional traffic in the network and burden on each peer.

The most recent related work can be found in [81]. In this work, they describe an algorithm for creating an overlay with constant delay between any two peers in the network. They compare their algorithms to two other state-of-the-art algorithms, which they show to out-perform. Their algorithm works in the following fashion: Each peer maintains a number of small random ids. In addition, it samples the network by contacting a number of random peers and initiating a walk from each one, by following peers of decreasing ids, towards the peer with the minimum id in the network. The peers with the lowest latency are chosen as neighbours. We chose this algorithm to compare with ITA latency-wise. Experimental results show that ITA obtains lower latency between peers. In addition, as we shall describe, ITA requires a constant number of samples to create the overlay, whereas Hsiao et al. Algorithm requires a number of samples, which is logarithmic to the number of peers in the system.

Finally, most of the existing literature focuses on reducing the IP latency of queries. We evaluate our work using a variety of metrics including IP latency reduction, IP message reduction, and the traffic load placed on each rout er in the underling IP network. The latter we believe to be a crucial, often neglected, metric in current widely deployed P2P systems.

5.3 ITA Design

This section contains a detailed description of the parts that comprise the design of our ITA algorithm. We then present a discussion and analysis of the advantages which arise from it.

Overlay construction

The ultimate objective of the bootstrapping algorithm is to create for each peer a number of randomly selected *short* connections to *closer* (but not the closest) peers and the same number of randomly selected *long* connections to *distant* peers. The definition of the "short" and "long" connections is based on parameter $\alpha \leq 1$ which constitutes the basic and most fundamental parameter of the algorithm. Let N be the total number of peers in the networks. Each peer A that bootstraps to the network selects its "short" connections randomly among its $\alpha * N$ closer (latency wise) peers, while it selects its "long" connections randomly among the $(1 - \alpha) * N$ more distant (latency wise) peers.

To implement this method, each peer A calculates a (latency related) threshold value x directly dependant on parameter α. Given the value of parameter $\alpha \leq 1$, each peer A that bootstraps to the network approximates a threshold value x so that the number of peers whose latency to A is less than x is $\alpha * N$. In other words, if C is the set of all peers P for which it holds that latency(A, P) ≤ x, peer A calculates its threshold value x so that $|C| = \alpha * N$. Since the latency from each peer to all other peers cannot be measured, the calculation of the threshold value x is approximated by having each peer A make latency measurements to 30/α randomly selected peers. A proof is provided in Proposition 3 below, which shows that this number of latency samples leads to a good threshold approximation.

Proposition 3: Each peer needs 30/α latency measure-ments to other peers in order to approximate threshold x such that $|C| = \alpha * N$ for given $\alpha \leq 1$, with accuracy 95%.

Proof: A peer belongs to C with probability α. To obtain a good threshold approximation, we will select a peer in C that is among the 0.1*|C| peers whose latency is closer to the threshold value. The number of peers which are closer to the threshold according to our choice is $0.1 * \alpha * |C| = \alpha' * |C|$. The probability that a single randomly selected peer belongs to that space is $\alpha' = 0.1 * \alpha$. The probability that neither one of n randomly selected peers belong to that space is $(1 - \alpha')^N \simeq e^{-a' * n}$. To approximate the threshold with accuracy 95% we need

$$e^{-\alpha' * n} <= 0.05 \Rightarrow ln(-\alpha' * n) \leq ln(0.05) \Rightarrow$$
$$-\alpha' * n \leq -3 \Rightarrow n \geq 3/\alpha' \Rightarrow n \geq 30/\alpha$$

which concludes the proof.

So, each peer needs 30/α latency measurement samples to approximate the threshold. In [5], it has been shown that the last peers of k random walks of logN length comprise a uniformly random selection of k peers.

During the sampling measurements, peer A can connect randomly to begin its operation, without having to wait for the end of the of the sampling procedure.

In addition, the Vivaldi coordinate system [34] can be used to facilitate and speed-up the bootstrapping process. Vivaldi is a P2P network coordinate system which can assign a 3-dimensional coordinate to a host. The Euclidian distance between two Vivaldi points (corresponding to two hosts) is an approximation of their latency. Thus, each message broadcast by any peer can contain its Vivaldi coordinates. A bootstrapping peer A can monitor incoming traffic, collect 30/α Vivaldi coordinates and thus compute the threshold value x. Ultrapeers today are reached by at least fifty query messages per second, making the threshold calculation this way a matter of seconds.

It should also be noted that, unless the structure and capacity of the network changes significantly, the threshold value remains unchanged, and so does not need to be recalculated each time the peer joins the overlay. After a threshold value has been obtained, peer A connects to 2/α neighbors in the following fashion

- It connects randomly to 1/α peers, all of which belong to C (i.e.: any peers with a latency lower than the threshold value). These links are called *short* links.
- It also connects randomly to 1/α other peers, which *do not* belong to C. These are called *long* links.

To illustrate, let's assume that parameter α is set to 0.1. This means that C contains approximately 0.1 ∗ N nodes of all the nodes (peers) in the system. Note that the set C is, of course, different for each peer. Each peer A will create 1/α = 10 short links randomly selected among the 10% closer to A peers (i.e. among the peers in A's C set), and the same number of long links randomly selected from the 90% further peers. The number of sample measurements required for the calculation of the threshold, in this case, is 30/α = 300. Not only it take a few seconds to perform this number of RTT measurements, it only takes place once, and not every time a peer (re-)connects in the system.

Search algorithm

Search is conducted in the following fashion:

- The Initiator peer floods its long links with TTL = 1.
- Each of the peers that receives the flood over a long link (and the Initiator peer) initiates a flood with a given TTL = ttl (system parameter) over their short links only.

The long link peers which initiate the localized floods (over their short links) use 1-hop replication as well as dynamic querying the same fashion it is used in Gnutella today. Since short links are randomly connected the efficiency of dynamic querying and 1-hop replication is guaranteed. Alternatively, Dynamic Querying can be used on the long links level by sequentially sending a new flood with increasing TTL, to each long link neighbour.

Analysis

The constructed graph, in conjunction with the described search method, has the following advantages:

- Both the long link-based, system-wide graph and the short link-based, local graphs are random, since each peer selects peers (outside and inside C respectively) randomly for neighbors (i.e. each peer, for instance, in C has the same probability of becoming a short link peer of the same peer A). This enables both 1-hop replication and dynamic querying to operate as if they were executed on a random graph.

- Since any peer in C can serve as short link (instead of opting for the closest ones), the bootstrapping procedure is very fast and lightweight. The same holds for the long links. As a result each peer need only set up its neighbors once, regardless of arrivals and departures elsewhere in the overlay, making ITA as little affected by churn as Gnutella (i.e. a peer only needs to act when a neighbor leaves the system by simply replacing it with another one, as in Gnutella). This simplicity helps preserve almost intact the unstructured nature and the simplicity of construction of the overlay. What is more, if we tried to connect to the closest possible peers, this would require each peer to be on the constant lookout for some closer peer connecting (anywhere) to the network. This constant probing (dependant on churn degree) would increase both the traffic in the network and the computational load of the system. In addition, the threshold value is only affected by changes in the structure of the underlying IP network (which are not very frequent) and not by changes in the P2P overlay, which are rather frequent. So the value is calculated only once and not each time the peer (re-)joins the network.

- $(1 - \alpha) * N$ peers (furthest away at the IP layer) are excluded from becoming short links, which means the proposed system is quite aware of the underlying physical network topology. Increased awareness in the form of a very small α (i.e. trying to connect to the closest possible peers) would help us gain little but lose much, since the small size of the local neighborhoods would lead to high clustering.

- Finally, all local clusters/neighbourhoods have the same size, enabling the use of a single, system-wide TTL = ttl for flooding the short links.

We have conducted experiments using three distinct values for α, namely 0.1, 0.05 and 0.033. These values correspond to a number of 10, 20 and 30 long and the same number of short links. The above discussion justifies the reason for not using smaller values. Values in this range are sufficient for excluding most of the peers from the local "neighbourhood" set C of each peer, while being at the same time large enough to allow large enough neighbourhoods for quick and simple bootstrapping procedure (i.e. being able to quickly locate short-link neighbors). The value of α also dictates the number of the long links, since there are N/|C| = 1/α "neighbourhoods". In addition, the use of 1/α long links is due to the fact that the use of long links should only take place on the first hop, to avoid extra delays in the flood process.

Finally, it is important to note that there is no 1-hop replication between peers connected by long links, so there is no index information exchange. Thus, the maintenance overhead for the additional 1/α long links very low.

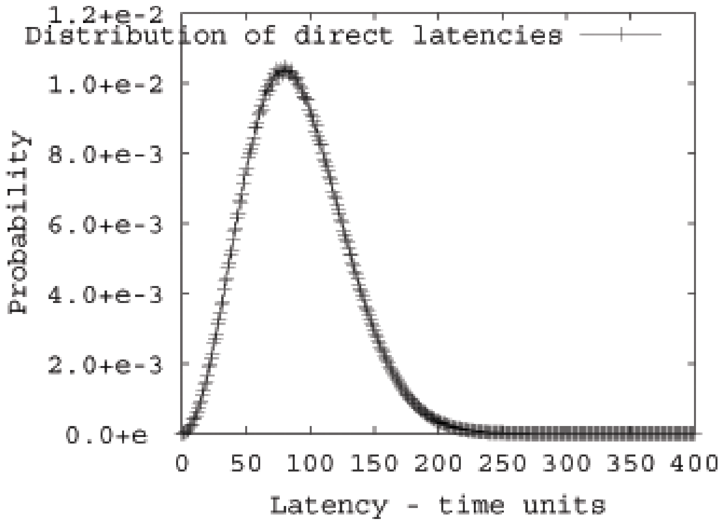

Figure 5.2: Distribution of direct latencies between all pairs of peers

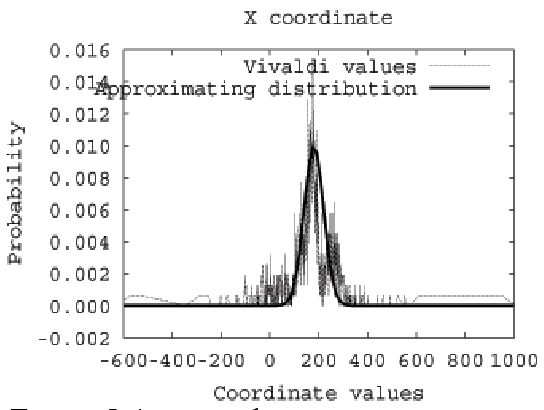

Figure 5.4: x coordinate

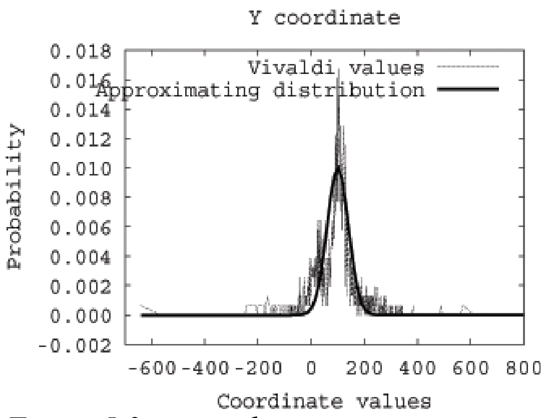

Figure 5.3: y coordinate

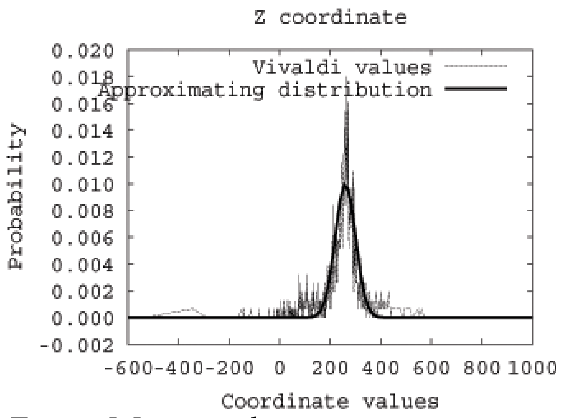

Figure 5.5: z coordinate

Figures 5.4, 5.5, 5.6. Actual values and approximation distribution for the three coordinates

5.4 Experimental results

In order to verify the arguments made in the previous section, we performed several experiments comparing our system with Gnutella, at its peak usage population (approximately 2 million users) [31]. We performed the comparison with Gnutella 0.6, which employs a 2-tier architecture [38], focusing on the Ultrapeer layer where flooding occurs. The metrics upon which our comparison was based were selected to capture the design goals of IT A, namely

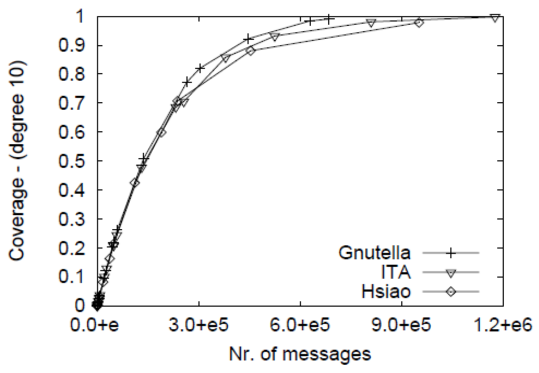

Figure 5.6: Average degree = 10

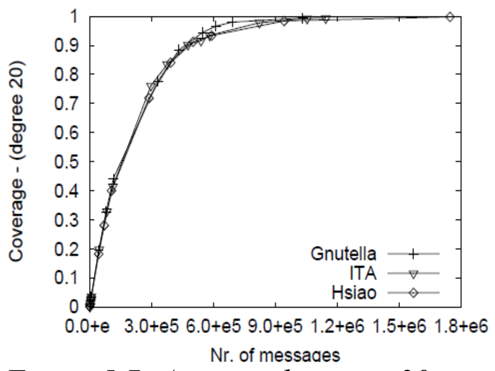

Figure 5.7: Average degree = 20

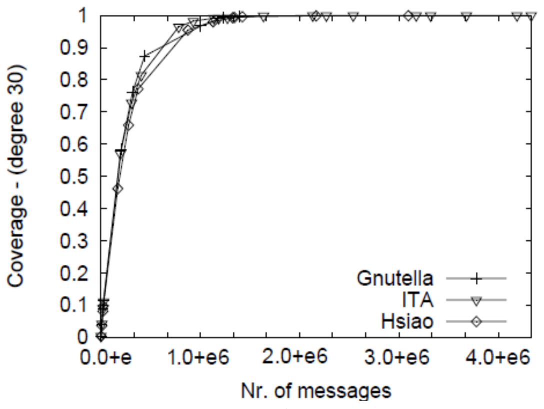

Figure 5.8: Average degree = 30

Figures 5.6, 5.7, 5.8. Flood reach for given number of messages.

to satisfy users by allowing them to get the same number of search query results faster by reducing query response time, and to satisfy ISPs by reducing the load imposed on their routers.

We also compared our system latency-wise with the most recent algorithm we could find in the literature, proposed in [81] by Hsiao et al. Each peer in the proposed algorithm also samples the network, in a different function, to locate peers with lower latency in order to connect to. That number of samples is however relative to the natural logarithm of the total number of peers in the system. In our algorithm, the number of samples is constant, regardless of the size of the network.

We simulated a network of 200,000 peers, which is a realistic number for the size of the Ultrapeer overlay in Gnutella according to LimeWire [30], the company that developed the most popular Gnutella client today [77]. We also used three average degree values for the overlay, namely 30 (which is the average number of connections in a Gnutella Ultrapeer today), 20 and 10. These three numbers correspond to the number of connections per peer in the Gnutella simulations and the number of short and long links in the simulations of the IT A algorithm. Note that since the long links are only used during the first hop of flooding, whereas the short links are used during the second and the remaining hops, the outbound degree during any flood hop is the same both in Gnutella and ITA, even though our algorithm uses double the number of links (short and long).

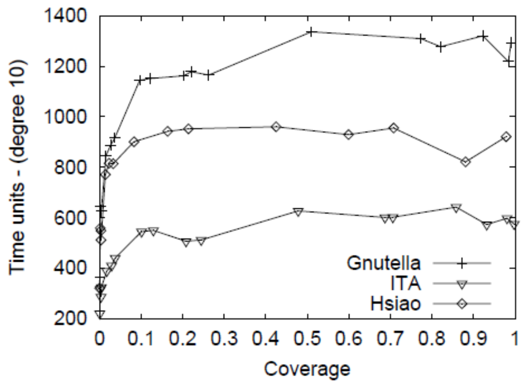

Figure 5.9: Average degree = 10

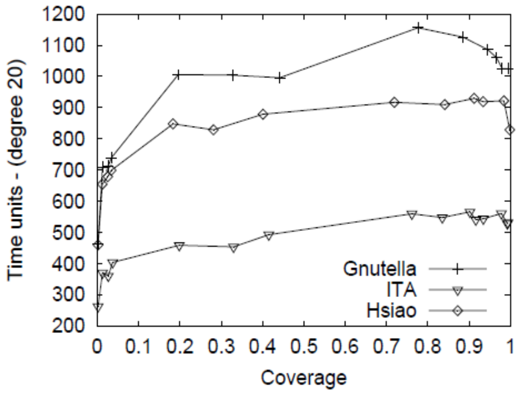

Figure 5.10: Average degree = 20

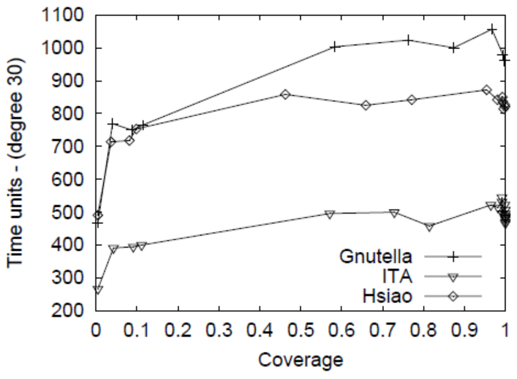

Figure 5.11: Average degree = 30

Figures 5.9, 5.10, 5.11. Time required by a flood versus the percentage of nodes reached.

We performed a large number of floods, in each experiment, with varying TTL values, resulting in a range of the ratio of the peers reached by the flood. For each TTL value we performed 100 floods and averaged the results. We compare ITA and Gnutella using three different metrics. The first metric is the latency of the connections of the peers, which affects the duration of a flood. We measure the average time it takes for a flood to complete, for different TTL values. The second metric is the number of IP messages generated during a single flood. We measure the average number of IP messages generated during floods of increasing TTLs. Finally, the last metric is the standard deviation of the message load imposed on the routers that comprise the IP layer of the Internet. We argue that a reduction in the total number of IP messages in the whole network is of little use if there exist a small number of bottleneck routers whose traffic load remains the same as before. As we mentioned above, the key goals of the IT A algorithm is to benefit *both* the P2P application *and* other applications sharing the same medium, the Internet. First though, we prove that the injection of topology awareness in the overlay construction has not affected the "randomness" of the system.

The random nature of the constructed overlay is indicated by the extent of the reach of a flood for given number of messages. This is because on a clustered graph, as shown in Figure 2.8, duplicate messages appear even from the second hop of the flood. Since duplicate messages, by definition, arrive at a peer which has already received another flood message, they do not add to the reach of the entire flood. Figures 5.6, 5.7 and 5.8 show the similarity between the Gnutella overlay (random graph), the Hsiao and the overlay constructed by ITA with respect to flooding. The close fit of all curves on the the graphs shows that the flood reach is the same using the same number of messages. This means that ITA can provide reduced latency and reduced router load benefits (see below) without affecting 1-hop replication, dynamic querying, and the self-* properties on which Gnutella-like systems depend for their performance. It should be noted here that latencies between neighbours in this experiment were modelled the same way as in the Latency experiments described next.

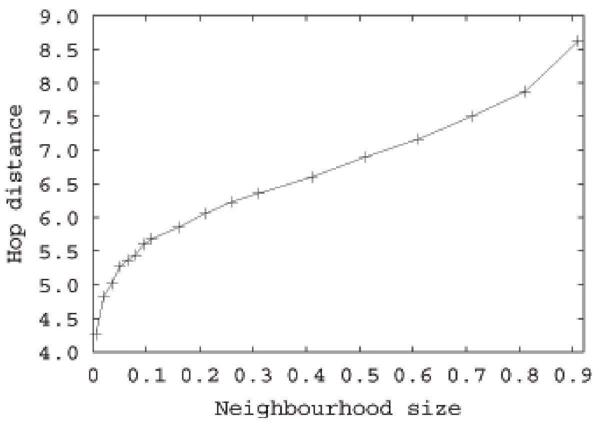

Figure 5.12: Diameter (in hops) of different neighborhood sizes

Latency experiments

In order to model the 200,000 by 200,000 latencies between our simulation peers, we obtained approx. 1000 real-world Vivaldi coordinates. Those 3D coordinates were produced by the Vivaldi project experiments on PlanetLab [34]. We then calculated a distribution which best fits the values observed in those coordinates and we generated 200,000 Vivaldi coordinates using this distribution, thus being able to model the latency between any pair of the 200,000 peers. Figures 5.3, 5.4 and 5.5 show the values of the original Vivaldi coordinates as well as the distributions generated from our approximation distribution. The close fit is an assurance that our randomly generated coordinates closely reflect real-world Vivaldi coordinates. Given the 200,000 x 200,000 latency matrix we generated, Figure 5.2 shows the distribution of the latency for an optimal full mesh graph where each peer has a direct overlay connection to each other peer. The figure shows that the average latency between any two peers is 90 time units.

Figures 5.9, 5.10 and 5.11 provide the experiment results for the first metric the time required for a single flood to conclude. They show the time it takes to flood the network, for given node coverage. We can see that for any desired coverage, the time it takes for our system to reach that number of peers is, on average, at most half the time for Gnutella flooding. Note that the measured time reflects the time from the beginning of the flood until even the last message generated by that particular flood expires. On the other hand, even though the Hsiao et al. algorithm does reduce the time for a flood, compared to Gnutella, it still requires more time than ITA.

There are two reasons for measuring flood duration rather than average response time for a search query. First, a reduction by half in flood duration implies a similar reduction in average query response time. What is more important however, is the fact that it is common for a flood to still be active and being propagated in the network, even though no new results are (and are going

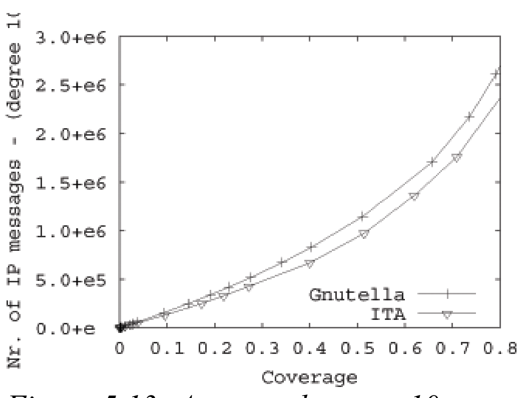

Figure 5.13: Average degree =10

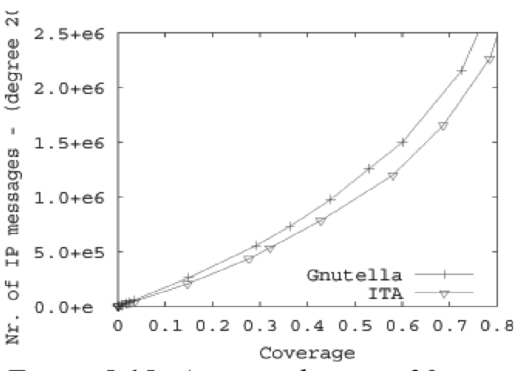

Figure 5.15: Average degree =20

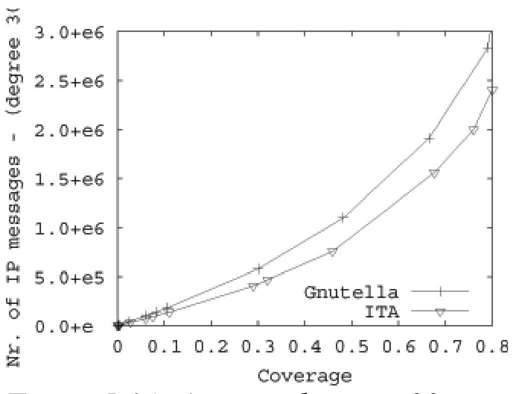

Figure 5.14: Average degree =30

Figures 5.13, 5.14, 5.15. IP messages generated by a flood versus the percentage of nodes reached. Router-level

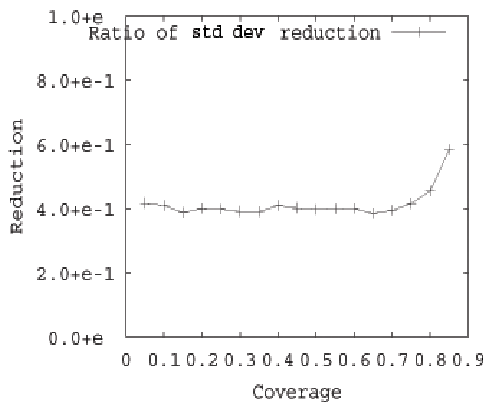

Figure 5.16: Average degree = 10

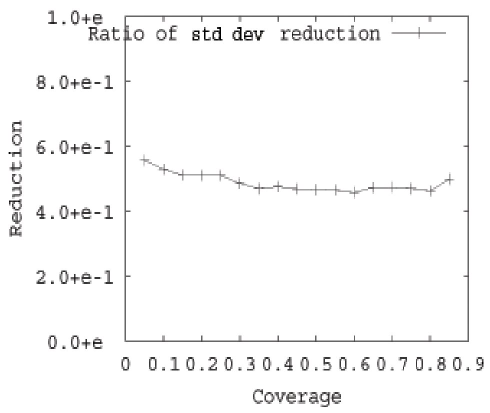

Figure 5.17: Average degree = 20

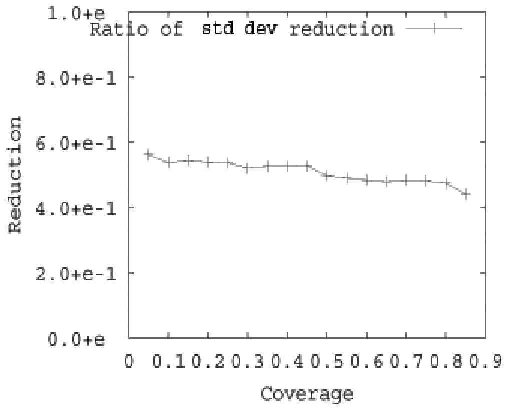

Figure 5.18: Average degree = 30

Figures 5.22, 5.23, 5.24. Ratio of standard deviation of router load reduction. Router-level

to be) provided to the user, so minimizing flood duration when possible is important. Given a constant rate by which new queries enter the network, by measuring the time it takes for a single flood to complete to the last message, we show that IT A doubles the exit rate of floods from the network. This means that ITA doesn't only reduce the number of IP messages per flood and divide traffic load more evenly among routers (as we will show in the next section), but also reduces the build-up of queues in the router buffers.

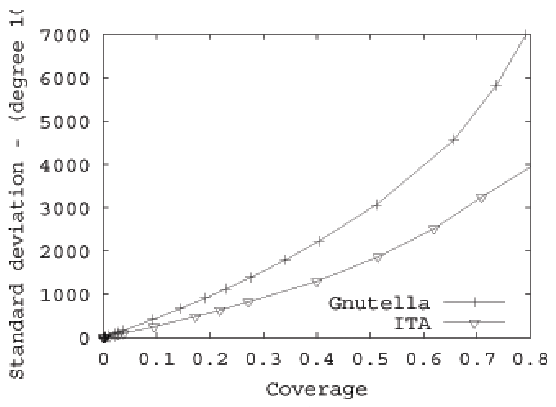

Figure 5.20: Average degree =10

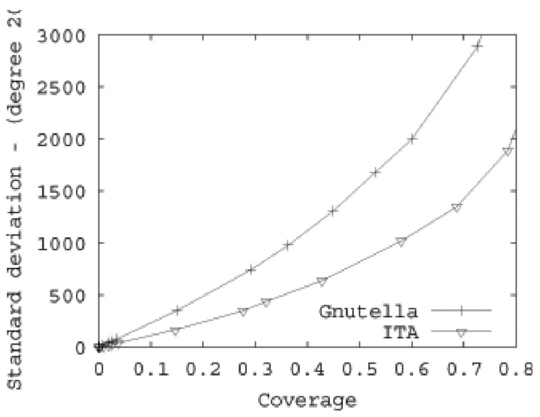

Figure 5.19: Average degree =20

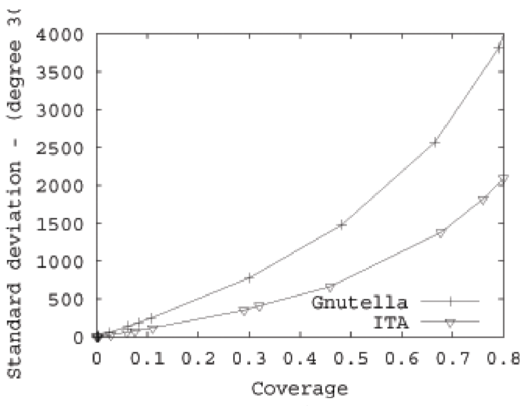

Figure 5.21: Average degree =30

Figures 5.19, 5.20, 5.21. Standard deviation of router traffic loads versus the percentage of nodes reached. Router-level

IP layer experiments

In this section we focus on the impact the ITA algorithm has on the IP layer. In order to perform simulations including the IP layer we obtained the latest trace of the router-level topology of the Internet from CAIDA [28]. This trace was publicly released in 2010 and it contais a much larger number of routers than the previous one. This trace initially contained approximately 33 million routers. However, we decided to remove the 1-degree routers (leaf routers) for two reasons. The first is the fact that performing simulations with this numbe r of routers was time (and probably memory) prohibiting. In addition, the existence of leaf routers in the IP topology would not add to the accuracy of the simulation results. By pruning those routers, we ended up with the much more managable dataset of 1.2 million routers. This dataset, in addition to making simulations feasible, still retains the structure of the Internet intact. In addition, it still is about six times larger than the previous CAIDA dataset and hundrends times larger than most of the router graphs used in similar simulations in the literature we have described.

In order to be more thorough in the evaluation of our algorithm, we also performed the same number of experiments at the AS layer. We also obtained an AS-level graph from CAIDA. This dataset contained approximately 30.000 Autonomous Systems. By obtaining the number of subnets for each AS from the

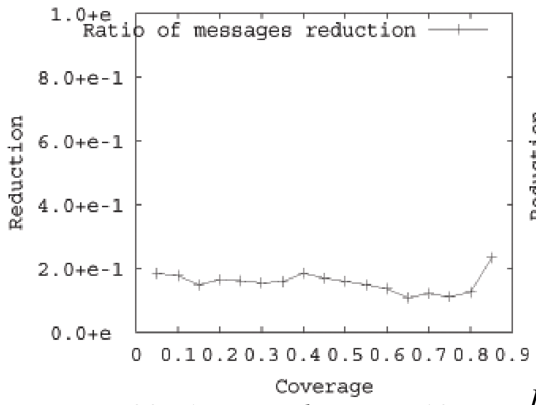

Figure 5.22: Average degree = 10

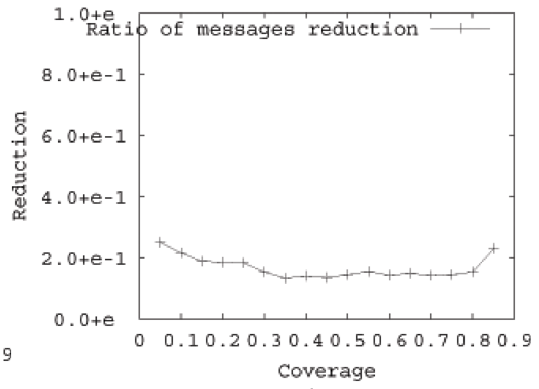

Figure 5.23: Average degree = 20

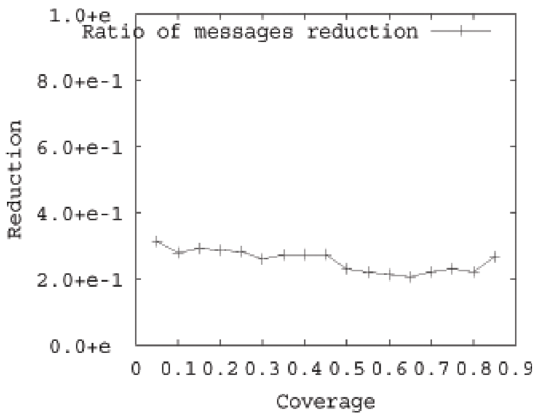

Figure 5.24: Average degree = 30

Figures 5.16, 5.17, 5.18. Ratio of IP message reduction. Router-level

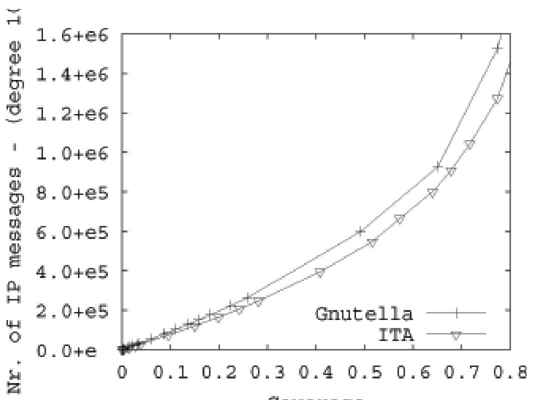

Figure 5.25: Average degree = 10

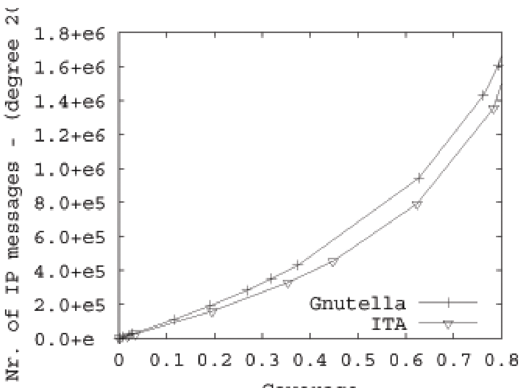

Figure 5.26: Average degree = 20

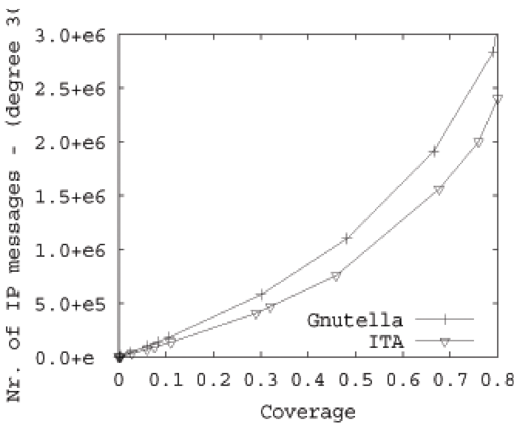

Figure 5.27: Average degree = 30

Figures 5.25, 5.26, 5.27. IP messages generated by a flood versus the percentage of nodes reached. AS-level

Internet Assigned Numbers Authority (IANA) [29], we were able to extract an AS population distribution, which we used to assign peers to each AS in our simulation.

In the IP layer experiments, both on the router and AS level, we used again 200,000 peers, each of which was randomly assigned to a router in the router-level graph, or AS in the AS-level graph. Since the CAIDA datasets do not contain latencies, we approximated the latencies with the number of IP hops between any two peers. Thus, each peer tries to form short links with those other peers whose routers are clocse to its own at the IP layer. Again, we do that by

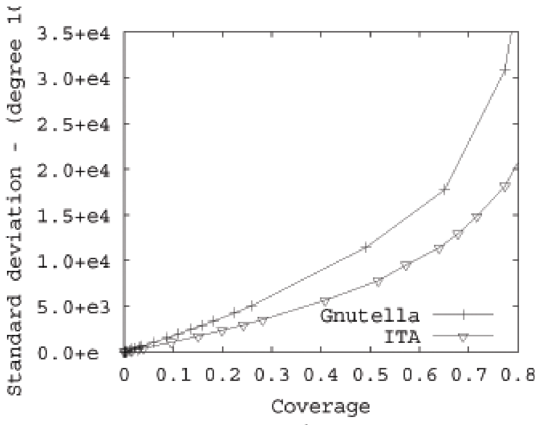

Figure 5.28: Average degree = 10

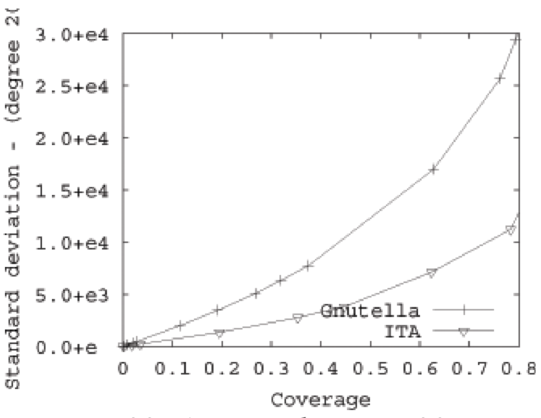

Figure 5.29: Average degree = 20

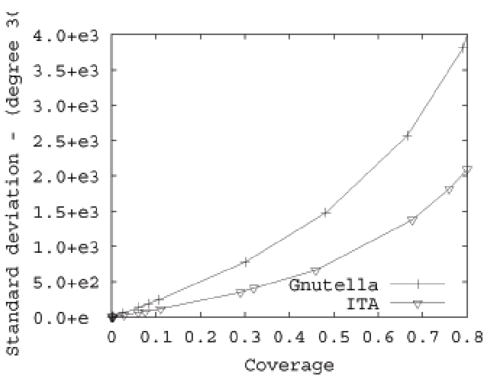

Figure 5.30: Average degree = 30

Standard deviation of router traffic loads versus the percentage of nodes reached. AS-level

obtaining the α ∗ 100% of all routers which are closest to our own router. Long links are again formed randomly, as are short links in a given neighborhood. Some measurements on the formed overlay show that the average number of routers in a Gnutella direct link between two peers is 6.9. In contrast, the same number for ITA's long links is 7 and for the short links it is 5.5. As we shall see below, we can expect a reduction of messages on the order of 15% to 25% ($\simeq (7 - 5.5)/7$). Given the percentage of the routers which can be reached for a single TTL value, which is shown in Figure 5.12, the average values we mentioned make a lot of sense. This figure shows the ratio of all peers that can be reached for a given hop distance. This shows that the vast majority of routers need to traverse a chain of at least 3 hops before they start encountering more than one per hop routers. This means that 4 is, more or less, a minimum value for a short link, imposing a lower bound on the reduction of IP messages that we can accomplish.

After running the simulations, which included performing several floods from random peers, with several TTL values to obtain coverage percentages ranging from 0 to 1, we obtained the following results on the router level: Figures 5.13, 5.14 and 5.15 illustrate the reduction in the number of IP messages for floods of various lengths. As one can see, the expected reduction that is observed is in the range of 15% to 25%. Figures 5.16, 5.17 and 5.18 show the reduction of the IP messages generated by ITA, compared to Gnutella. They show that, on average, 20% of the Gnutella IP messages, on the router graph experiments, are absent from the ITA experiments. Figures 5.25, 5.26 and 5.27 display the results of the similar experiments, albeit conducted in the AS level. The similarity of the results provide a good argument for their consistency.

Another important metric for the efficiency of any topology-aware overlay construction algorithm is the traffic load dis tribution across the routers in the system. Any reduction in the total number of IP messages is of little use if the number of messages forwarded by a small number of (possibly core) routers remains unchanged. For this reason, we plotted the standard deviation in the traffic load of all routers, again for floods of different sizes. Figures 5.19, 5.20 and 5.21 show that ITA reduces the standard deviation approximately by 40% to 50% on the router level graph. Similarly, Figures 5.22 through 5.24 show the relative reduction in the standard deviation of router loads. This means that there is a reduction in the effect of bottle-necks in the network. The same experimental results on the AS level are presented in figures 5.28, 5.29 and 5.30. In addition, we measured the traffic load for the most heavily used router, which ITA also cuts down by half, reducing the bottleneck effect on the routers.

Finally, we would like to note the stability of the aforementioned reductions in all floods performed, regardless of TTL value, meaning that even for floods of smaller reach, the algorithm is still beneficial. In addition, it should be noted that, in order to implement this algorithm, there is no need to change the Gnutella communication protocol itself, but rather only new versions of servents are only required.

5.5 Summary

In this Chapter, we presented ITA algorithm, a novel approach for injecting topology awareness into unstructured Gnutella-like P2P systems, while maintaining the self-* properties of the overlay topologies that are highly desirable in these systems. ITA reduces to half the time required for a search query to achieve a particular network coverage compared to the latest version of the widely deployed Gnutella. It also outperforms a recently proposed, state-of-the-art algorithm for topology awareness. Moreover, ITA reduces the number of IP messages generated during a search query flood by as much as 25%, which is a significant reduction for ISPs who care about the load imposed on their routers and its effect on the performance of other applications. Finally, there is an additional reduction by approximately by half on the standard deviation of router loads.

Chapter 6

Conclusions – Future Directions

Unstructured P2P systems present a strong design paradigm for the development of global-scale distributed applications and systems. Their self-organizing and self-healing capabilities can provide a system with increased scalability and robustness, characteristics important to any design. Their main drawback has been the inefficient routing of messages, both on the overlay and on the underlying network layers. In this dissertation, we aimed at tackling several faces of this problem.

Messages exchanged between peers can follow several alternate paths to reach the same destination peer. This ensures the receipt of the message and increases system robustness. It also however generates a large number of redundant messages due to the same peer receiving the same message from alternate paths. We developed a distributed algorithm which detects the paths followed by those redundant messages and chokes those paths, reducing the number of redundant messages by 90% while experiencing a loss of only 10% in system coverage. We also showed this algorithm to be efficient even in the face of constant changes in the system, due to the dynamic nature of the P2P systems.

In addition, we proposed a new overlay creation and search method, which tries to reduce the amount of overlay traffic required per query. By dividing the participants of the system into sub-groups and assigning each group responsibility for indexing a part of the total system content, we have made it possible to achieve the same query satisfaction success with approximately 8 times less traffic, while at the same time maintaining the unstructured, loose nature of the system, which provides it with its excellent self-healing capabilities.

Finally, we focused on the cooperation of the system with its underlying network medium, namely the IP layer. Given the fact that a single P2P overlay message between two peers can be translated into many IP messages, we presented an algorithm for overlay creation and search (complementary to the previous one), which attempts to add topology awareness to the overlay of the P2P system, without harming its random nature. As a result, we achieved a more than half reduction in the time required for the execution of a query. In addition, we achieved an IP messages reduction of about 20% and at the same time, a more than half reduction of the standard deviation of router loads among all Internet routers.

We believe that all these proposals combined together present a significant reduction in the traffic generated by unstructured P2P systems in the world today.

All in all, we strongly believe Peer-to-Peer systems to continue to be an important paradigm in the design of future large-scale distributed systems. Even though the industry seems to favour centralized designs for the time being, for reasons such as security and control, we believe that in the future more applications will look forward to exploiting the large power and resources contained in the sheer number of end-users. This fact will reshape the nature of Peer-to-Peer systems, as new applications with different needs appear, leading to the emergence of new problems requiring solutions in the field.

Already P2P systems are used today for more than file-sharing. BitTorrent and Skype, two of the most widely used P2P systems, both enable the use of bandwidth sharing. Future systems have already been proposed that will allow users to share storage space with each other, such as PAST [82] and OceanStore [83]. The P2P paradigm will expand in the future to include such diverse applications, as Video Conferencing, Location-based services in Mobile Ad Hoc Networks (MANET), as well as Context-Aware Services and anonymous yet secure e-commerce.

Several research issues can be solved first, in order to enable P2P Systems to spread to a wide range of applications. These topics include more research in overlay optimization and resource allocation to guarantee Quality of Services in P2P Systems.

More incentives need to be injected into future systems to reduce the number of free-riding peers.

Another interesting topic is the development of semantic searches, which will increase the richness in query formulation and enable more precise and meaningful searches.

A decentralized reputation mechanism has been researched since the advent of P2P systems, however an effective, accurate and deployable mechanism has yet to be proposed.

Another very interesting topic would be the interoperability of similar P2P based applications, such as different file-sharing systems.

Thesis publications

1. Charis Papadakis, Paraskevi Fragopoulou, Elias Athanasopoulos, Evangelos Markatos, Marios Dikaiakos and Alexandros Labrinidis: A Feedback Based Approach to Reduce Duplicate Messages in Unstructured Peer-to-Peer Systems. In Integrated Research in GRID Computing, pages 103-118, Springer, 2007. ISBN: 978-0-387-47656-8
   (Editors: Sergei Gorlatch and Marco Danelutto)
2. Harris Papadakis, Paraskevi Fragopoulou, Evangelos P. Markatos, Marios D. Dikaiakos, Alexandros Labrinidis. Hash-Based Overlay Partitioning in Unstructured Peer-to-Peer Systems. In Parallel Processing Letters, 19(1), pp. 57-71, 2009.
3. Harris Papadakis, Mema Roussopoulos, Paraskevi Fragopoulou and Evangelos P. Markatos. Imbuing unstructured P2P systems with non-intrusive topology awareness. In Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing. September 2009, Seattle.
4. Harris Papadakis, Paolo Trunfio, Domenico Talia and Paraskevi Fragopoulou. An Experimental Evaluation of the DQ-DHT Algorithm in a Grid Information Service In Proceedings of the CoreGRID ERCIM Working Group Workshop on Grids, P2P and Service Computing. Held in conjunction With EuroPAR 2009, August 2009.

5. Agostino Forestiero, Carlo Mastroianni, Harris Papadakis, Paraskevi Fragopoulou, Alberto Troisi, Eugenio Zimeo, A Scalable Architecture for Discovery and Composition in P2P Service Networks, 2008 CoreGRID Integration Workshop, Heraklion-Crete, Greece.
6. Harris Papadakis, Paolo Trunfio, Domenico Talia and Paraskevi Fragopoulou. Design and Implementation of a Hybrid P2P-based Grid Resource Discovery System. In Proceedings of the CoreGRID Workshop on Grid Programming Models, Grid and P2P System Architecture, Grid Systems, Tools and Environments. Heraklion, Crete, Greece June 12-13, 2007. Also appered in Making Grids Work, Springer, USA, 2008, ISBN: 978-0-387-78447-2. (Editors: Marco Danelutto, Paraskevi Fragopoulou and Vladimir Getov)
7. Harris Papadakis, Paraskevi Fragopoulou, Marios Dikaiakos, Alexandros Labrinidis and Evangelos Markatos. Divide Et Impera: Partitioning Unstructured Peer-to-Peer Systems to Improve Resource Location. In Proceedings of the 2nd CoreGRID Integration Workshop, October 2006. Also appeared in Achievements in European Research on Grid Systems, pages 1-12. Springer, 2008. ISBN: 978-0-387-72811-7
8. Peer-to-Peer resource discovery in Grids: Models and systems, Trunfio, P., Talia, D., Papadakis, H., Fragopoulou, P., Mordacchini, M., Pennanen, M., Popov, K., Vlassov, V., Haridi, S. Future Generation Computer Systems, volume 23, issue 7, year 2007, pp. 864 – 878
9. Demetrios Zeinalipour-Yazti, Harris Papadakis, Chryssis Georgiou, Marios D. Dikaiakos: Metadata Ranking and Pruning for Failure Detection in Grids. Parallel Processing Letters 18(3): 371-390 (2008)

Bibliography

[1] Y. Chawathe, S. Ratnasamy, and L. Breslau. Making Gnutella-like P2P Systems Scalable. ACM SIGCOMM, 2003.

[2] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. Int. Conf. Distributed Comp. Systems, 2002.

[3] Garrett Hardin,"The Tragedy of the Commons,"Science, Vol. 162, No. 3859 (December 13, 1968), pp. 1243-1248.

[4] Duncan, J. Watts, and S. H. Strongatz. Collective Dynamics of Small-world Networks. Nature, 393:440-442, 1998.

[5] C. Gkantsidis, M. Mihail, and A.Saberi. Hybrid Search Schemes for Unstructured Peer-to-Peer Networks. IEEE INFOCOM, 2005.

[6] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. Int. ACM Conf. Supercomputing, 2002.

[7] R. Manfredi and T. Klingberg. Gnutella 0.6 Specification, http://rfc-gnutella.sourceforge.net/src/rfc-0 6-draft.html

[8] M. Ripenau, I. Foster, A. Iamnitchi, and A. Rogers. UMM: A Dynamically Adaptive, Unstructured, Multicast Overlay. In Service Management and Self-Organization in IP-based Networks, Dagstuhl Seminar Proceedings, 2005.

[9] Sharman Industries. Kazaa, http://www.kazaa.com

[10] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location using Interest-Based Locality in Peer-to-Peer Systems. IEEE INFOCOM, 2003.

[11] D. Tsoumakos and N. Roussopoulos. A Comparison of Peer-to-Peer Search Methods. Int. Workshop on the Web and Databases, 2003.

[12] Z. Zhuang, Y. Liu, L. Xiao, and L.M. Ni. Hybrid Periodical Flooding in Unstructured Peer-to-Peer Networks. Int.l Conf. Parallel Computing, 2003.

[13] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos. Exploiting Locality for Scalable Information Retrieval in Peer-to-Peer Systems. Information Systems Journal, 30(4):277-298, 2005.

[14] Gnutella 0.6 protocol specification.

http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[15] Limewire Inc. http://www.limewire.com

[16] B.H.Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, 1970.

[17] Y.Chawathe, S.Ratnasamy, L.Breslau,N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. Proc. ACM SIGCOMM 2003 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 407-418, 2003.

[18] V.Cholvi P. Felber, and E.Biersack. Efficient search in unstructured peer-to-peer networks. Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures, 2004.

[19] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, 2002.

[20] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. Proc. of INFOCOM, 2005.

[21] C. Papadakis P. Fragopoulou E. Athanasopoulos M. Dikaiakos, A. Labrinidis, and E. Markatos. A feedback-based approach to reduce duplicate messages in unstructured peer-to-peer networks. Proc. of the CoreGRID Integration Workshop, 2005.

[22] R. Rejaie, Shanyu Zhao, and D. Stutzbach. Characterizing files in the modern Gnutella network:Ameasurement study. Proc. SPIE/ACM Multimedia Computing and Networking, 2006.

[23] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest based locality in peer-to-peer systems. Proc. of INFOCOM, 2003.

[23] D. Stutzbach and R. Rejaie. Characterizing the two-tier gnutella topology. Proc. of the ACM SIGMETRICS, Poster Session, 2005.

[25] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS02), 2002.

[26] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. Proc. Int. Conference on Data Engineering (ICDE 2003), pp. 49-60, 2003.

[27] Fisk, A. Gnutella Ultrapeer Query Routing, v. 0.1. LimeWire Inc. 2003

[28] Cooperative association for internet data analysis,

http://www.caida.org/home.

[29] The internet assigned numbers authority (iana), http://www.iana.org/.

[30] Limewire inc, http://www.limewire.com.

[31] E. Bangeman. Study: Bittorrent sees big growth, limewire still nr.1 p2p app. ars technica, 2008.

[32] M. M. C. Gkantsidis and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. INFOCOM, 2005.

[33] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Proximity neighbor

selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Harvard, 2003.

[34] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. SIGCOMM, 2004.

[35] A. Dufour and L. Trajkovi´c. Improving gnutella network performance using synthetic coordinates. In QShine '06: Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks, page 31, New York, NY, USA, 2006. ACM.

[36] V. C. P. Felber and E. Biersack. Efficient search in unstructured peer-to-peer networks. Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures, 2004.

[37] A. Fisk. Gnutella ultrapeer query routing, v. 0.1. 2003.

[38] T. G. D. Forum. Gnutella 0.6 protocol specification.

[39] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. Proceedings of the 11th international conference on Information and knowledge management, (CIKM02), page 300307, 2002.

[40] B. Krishnamurthy and J. Wang. Topology modeling via cluster graphs. SIGCOMM Internet Measurement Workshop, 2001.

[41] N. Laoutaris, G. Smaragdakis, A. Bestavros, and J. Byers. Implications

of selfish neighbor selection in overlay networks. IEEE INFOCOM, 2007.

[42] Z. Li and P. Mohapatra. Impact of topology on overlay routing service. IEEE INFOCOM, 2004.

[42] Y. Liu, H. Zhang, W. Gong, and D. F. Towsley. On the interaction between overlay routing and underlay routing. IEEE INFOCOM, 2005.

[43] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. IPTPS02, 2002.

[44] V. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. ACM SIGCOMM, 2001.

[45] H. Papadakis, P. Fragopoulou, M. Dikaiakos, A. Labrinidis, and E. Markatos. Divide et impera: Partitioning unstructured peer-to-peer systems to improve resource location. Achievements in European Research on Grid Systems CoreGRID Integration Workshop 2006 (Selected Papers), 2007.

[46] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pages 161–172, New York, NY, USA, 2001. ACM.

[47] C. Rohrs. Query routing for the gnutella network. 2001.

[48] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pages 329–350, 2001.

[49] S. Saroiu, K. P.Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An analysis of internet content delivery systems. 5th Symposium on Operating Systems Design and Implementation, 2002.

[50] S. Sen and J.Wang. Analyzing peer-to-peer traffic across large networks. ACM SIGCOMM Internet Measurement Workshop, 2002.

[51] G. Smaragdakis, N. Laoutaris, A. Bestavros, J. W. Byers, and M. Roussopoulos.

Egoist: Overlay routing using selfish neighbor selection. BUCS-TR-2007-013, 2007.

[52] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. INFOCOM, 2003.

[53] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. Proceedings of the 2001 ACM SIGCOMM Conference, pages 149–160, 2001.

[54] L. Yunhao, X. Li, L. Xiaomei, N. L. M., and Z. Xiaodong. Location awareness in unstructured peer-to-peer systems. Parallel and Distributed Systems, IEEE Transactions on, 16(2):163–174, 2005.

[55] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

[56] Garrett Hardin,"The Tragedy of the Commons,"Science, Vol. 162, No. 3859 (December 13, 1968), pp. 1243-1248.

[57] M. Cai, M. Frank, J. Chen, P. Szekely, MAAN: A multi-attribute addressable network for Grid information services, in: Proc. 4th Int. 3 Workshop on Grid Computing, GRID 2003, 2003, pp. 184–191.

[58] M. Cannataro, D. Talia, Semantics and knowledge Grids: Building the next-generation Grid, IEEE Intelligent Systems 19 (1) (2004) 56–63.

[59] H. Chen, H. Jin, X. Ning, Semantic Peer-to-Peer overlay for efficient content locating, in: Proc. Int. Work. on Advanced Web and Network Technologies, APWeb 2006, 2006, pp. 545–554.

[60] A. Crespo, H. Garcia-Molina, Routing indices for Peer-to-Peer systems, in: Proc. 22nd Int. Conf. on Distributed Computing Systems, ICDCS'02, 2002, pp. 23–30.

[61] A. Crespo, H. Garcia-Molina, Semantic overlay networks for P2P systems, Technical Report, Stanford University, 2003.

[62] F. Dabek, E. Brunskill, M. Frans Kaashoek, D.R. Karger, R. Morris, I. Stoica, H. Balakrishnan, Building Peer-to-Peer systems with chord, a distributed lookup

service, in: Proc. 8th Workshop on Hot Topics in Operating Systems, HotOS-VIII, 2001, pp. 81–86.

[63] M. Frans Kaashoek, D.R. Karger, Koorde: A simple degree-optimal distributed hash table, in: Proc. Second Int. Workshop on Peer-to-Peer Systems, IPTPS 2003, 2003, pp. 98–107.

[64] D. Heimbigner, Adapting publish/subscribe middleware to achieve Gnutella-like functionality, in: Proc. 2001 ACM Symposium on Applied Computing, SAC, 2001, pp. 176–181.

[65] A. Iamnitchi, I.T. Foster, A Peer-to-Peer approach to resource location in Grid environments, in: J. Weglarz, J. Nabrzyski, J. Schopf, M. Stroinski (Eds.), Grid Resource Management, Kluwer, 2003.

[66] F.B. Kashani, C.C. Chen, C. Shahabi,WSPDS:Web services Peer-to-Peer discovery service, in: Proc. Int. Conf. on Internet Computing, IC’04, 2004.

[67] Simon G.M. Koo, Karthik Kannan, C.S. George Lee, On neighbor-selection strategy in hybrid Peer-to-Peer networks, Future Generation Computer Systems (2006) 732–741.

[68] F.B. Kashani, C. Shahabi, Searchable querical data networks, in: Proc. First Int. Workshop on Databases, Information Systems, and Peer-to-Peer Computing, DBISP2P, in: LNCS, vol. 2944, Springer, 2003, pp. 17–32.

[69] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Looser, Super-peer-based routing and clustering strategies for RDF-based Peer-to-Peer networks, in: Proc. 12th Int. Conf. World Wide Web, WWW ’03, 2003, pp. 536–543. 91

[70] D. Puppin, S. Moncelli, R. Baraglia, N. Tonelotto, F. Silvestri, A Grid information service based on Peer-to-Peer, in: Proc. 11th Euro-Par Conf., Euro-Par 2005, in: LNCS, vol. 3648, Springer, 2005, pp. 454–464. 103

[71] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location and routing for large scale Peer-to-Peer systems, in: Proc. IFIP/ACM Int. Conf. on Distributed Systems Platforms,Middleware 2001, in: LNCS, vol. 2218, Springer, 2001, pp. 329–350. 113

[72] C. Schmidt, M. Parashar, Flexible information discovery in decentralized distributed systems, in: Proc. 12th Int. Symp. on High-Performance Distributed Computing, HPDC-12 2003, 2003, pp. 226–235.

[73] A. Singla, C. Rohrs, Ultrapeers: Another step towards Gnutella scalability. http://rfc-gnutella.sourceforge.net/src/Ultrapeers 1.0.html. 125

[74] John R. Douceur, The Sybil Attack. In Proceedings of IPTPS '01 Revised Papers from the First International Workshop on Peer-to-Peer Systems. Springer-Verlag London, UK, ISBN:3-540-44179-4

[75] Akamai Technologies, http://www.akamai.com/

[76] *On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing*. Ian Foster and Adriana Iamnitchi, 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), February 2003, Berkeley, CA

[77] Gnutella, http://en.wikipedia.org/wiki/Gnutella

[78] The Second Coming of Gnutella, http://www.xml.com/pub/r/1005

[79] Fan Li, Cao Pei, Almeida Jussara, Broder, Andrei (2000), "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", *IEEE/ACM Transactions on Networking* 8(3): 281–293, doi:10.1109/90.851975. A preliminary version appeared at SIGCOMM '98.

[80] Bloom filiters wikipedia entry. http://en.wikipedia.org/wiki/Bloom_filter

[81] H.-C. Hsiao, H. Liao, and C.-C. Huang. Resolving the topology mismatch problem in unstructured peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 20:1668–1681, 2009.

[82] A. Rowstron and P. Druschel. *Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility*. 18th ACM SOSP01. 2001

[83] J. Krubiatowicz, D. Bindel, Y. Chen et al. *OceanStore: An Architecture for Global Scale Persistent Storage*. Proceedings of the 9th International Conference on Architecture Support for Programming Languages and Operating Systems. 2000.