Parallel Processing Letters, Vol. 19, No. 1 (2009) 57–71
© World Scientific Publishing Company

HASH-BASED OVERLAY PARTITIONING IN UNSTRUCTURED PEER-TO-PEER SYSTEMS*

HARRIS PAPADAKIS, PARASKEVI FRAGOPOULOU[†], EVANGELOS P. MARKATOS
Foundation for Research and Technology-Hellas, Institute of Computer Science
N. Plastira 100, Vassilika Vouton, GR-70013 Heraklion, Crete, Greece
{adanar, fragopou, markatos}@ics.forth.gr

MARIOS D. DIKAIAKOS
Department of Computer Science, University of Cyprus
1678 Nicosia, Cyprus
{mdd}@cs.ucy.ac.cy

ALEXANDROS LABRINIDIS
Department of Computer Science, University of Pittsburgh
Pittsburgh, PA 15260, USA
{labrinid}@cs.pitt.edu

Received February 2008
Revised July 2008
Communicated by S.G. Akl

ABSTRACT

Unstructured peer-to-peer (P2P) networks suffer from the increased volume of traffic produced by flooding. Methods such as random walks or dynamic querying managed to limit the traffic at the cost of reduced network coverage. In this paper, we propose a partitioning method of the unstructured overlay network into a relative small number of distinct subnetworks. The partitioning is driven by the categorization of keywords based on a uniform hash function. The method proposed in this paper is easy to implement and results in significant benefit for the blind flood method. Each search is restricted to a certain partition of the initial overlay network and as a result it is much more targeted. Last but not least, the search accuracy is not sacrificed to the least since all related content is searched. The benefit of the proposed method is demonstrated with extensive simulation results, which show that the overhead for the implementation and maintenance of this system is minimal compared to the resulted benefit in traffic reduction.

Keywords: Peer-to-peer, resource location, flooding, overlay network

*This research work was carried out under the FP6 NoE CoreGRID funded by the EC (IST-2002-004265).
[†]With the Department of Applied Informatics and Multimedia, Technological Educational Institute of Crete, Greece.

1. Introduction

In modern peer-to-peer (P2P) systems, different entities, under different authoritative control, are interconnected and cooperate on equal terms, in order to offer services to each other, acting both as servers and as clients, thus the term *peers* for the participating entities. This "equality" of participants role gives P2P systems an inherent scalability and robustness, justifying the popularity of the P2P paradigm, as an attractive tool which promises to enable the development of global-scale, cooperative, distributed applications. The techniques proposed in this paper apply to both file sharing and content distribution P2P systems.

P2P systems are distinguished in two main categories. In *structured* P2P systems all information stored in the system is indexed by employing a *Distributed Hash Table* (DHT), thus enabling efficient resource location in time (and number of messages) logarithmic in the number of participating peers. In structured P2P systems, the DHT imposes a certain order on the connectivity of the participating peers which is reflected in the structure of the overall network. The first and most famous structured P2P system was Chord. Others systems, such as the Pastry, Tapesty, BitTorrent followed. The main drawback of these systems is the maintenance cost of their rigid structure which limits their self-healing properties in case of failures, thus rendering them less robust, albeit more scalable. On the other side of the spectrum, we have *unstructured* P2P systems which employ a random overlay network to interconnect the participating peers. Those systems are aptly named unstructured since each peer is directly and randomly connected to a small set of other peers, called *neighbours*, making the network more ad-hoc in nature. The absence of a structure makes such systems much more robust and highly self-healing compared to structured systems, however, at the cost of reduced scalability.

Unstructured P2P systems employ a broadcast-like process called *flooding* for resource location. A peer looking for a file issues a query to its neighbors. The neighbours check whether they can satisfy the query, and, at the same time, forward it to their own neighbors. Since this mechanism may generate a tremendous number of messages, a limit has been imposed on the number of times a query may be forwarded, called TTL (Time-To-Live hops). Even though this addition increases the scalability of the method, it can greatly reduce the chances that a query is satisfied. In addition, another drawback of flooding is the fact that due to the completely decentralized nature of flooding, each peer may receive the same request through a number of different neighbours.

Overlooking the definition of P2P systems, modern unstructured systems employ a 2-tier architecture to reduce the cost of flooding. Most unstructured P2P systems like Gnutella 2 [1] also employ a 2-tier structure [20, 4]. 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 a compressed form

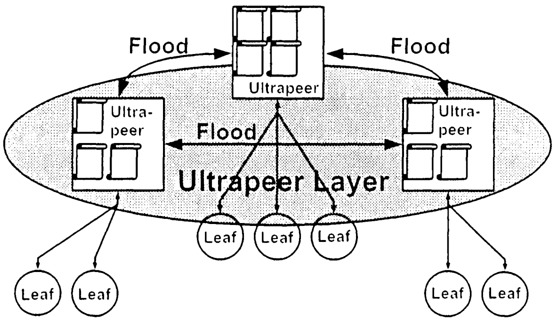

Fig. 1. The Gnutella 2-tier architecture.

(bloom filter). Ultrapeers flood queries to the overlay network on the Leaves' behalf. Flooding is only performed at the Ultrapeer level.

Another technique widely used in unstructured P2P systems today, is 1-hop replication. One-hop replication dictates that each peer should inform all of its immediate neighbors of the files it contains. Using this information during the last hop propagation of a request at the Ultrapeer level, the request is forwarded exclusively to those last hop Ultrapeers that contain the requested file. One-hop replication reduces the number of messages generated only during the last hop of flooding [9]. However, the traffic generated during that last hop constitutes the overwhelming majority of the traffic generated during the entire flooding. Simple calculations show that 1-hop replication requires d times fewer messages to spread to the whole network compared to naive flooding, where d is the average degree of the network (average number of connections for each Ultrapeer). It is easy to prove that in order to flood an entire, randomly constructed, network that employs 1-hop replication, one need only reach $3/d$ of the peers during all hops but the last. In today's Gnutella, where the average degree is 30, one would need to reach 10% of the peers and then use 1-hop replication to forward the query to the appropriate last hop peers, in order to reach the entire network.

Most of todays unstructured P2P systems implement 1-hop replication by having peers exchange bloom filters of their indices. A Bloom filter [3] is a space efficient way to represent a set of objects (keys). They employ one or more uniform hash functions to map each key to a position (or more than one) in an N-sized binary array, whose bits are initially set to 0. Each key is mapped through each hash function to an array position which is set to 1. To check for the participation of some key in the set, the key is hashed to get its array position. If that array position is set to 1, the bloom filter indicates key membership. Bloom filters require much less space than the actual set, there is thus some loss of precision translated in the possibility of *false positives.* This means that a bloom filter may indicate membership for some key that does not belong to the set (more than one keys mapped to the same position). It cannot however indicate absence of a key which is in the set (false negative).

In 2-tier unstructured P2P systems 1-hop replication is implemented among Ultrapeers by exchanging Bloom filters. This greatly reduces the number of messages since the number of Ultrapeers is much smaller than the number of Leaves. Whenever an Ultrapeer receives a request this is forwarded only down to those Leaves that contain the desired information (except in the case of false positives). Fig. 1 shows a schematic representation of the 2-tier architecture. The 2-tier architecture dictates that the Bloom filter exchanged by Ultrapeers for 1-hop replication is actually the OR of all the bloom filters it receives from its Leaves as well as its own.

The aim of the work presented in this paper is to improve the scalability of flooding by reducing the number of peers that need to be contacted on each request, without decreasing the probability of query success (accuracy of the search method) to the slightest extent. This can be accomplished due to the complete "unstructureness" of today's systems. We can therefore inject a small degree of structure in the unstructured system, one that is small enough to avoid sacrificing any of the self-healing capabilities of the system but at the same time will enable a more clever search which will provide great message cost reduction. 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, each containing only keywords of a specific type. In this manner, instead of providing a system with a way of knowing where the information is (the approach of structured P2P systems), we instead provide the system with a way of knowing where the information is NOT. This approach is much more preferable in the case of unstructured systems (Bloom filters are also based on the same philosophy). 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.

This paper is organized as follows: Following the "Related work" Section, the hashed-keyword method used to partition the overlay network is presented in Section 3. In Section 4 the simulation results are presented. We conclude in Section 5 with some directions for future research.

2. Related work

P2P-based resource discovery systems allow nodes participating in the system to share both the storage load and the query load [10]. In addition, they provide a robust communication overlay.

Flooding is supported by those P2P systems that follow the unstructured approach. Flooding, however, can generate a large volume of traffic if not carefully deployed, due to the duplicate messages generated during this process. Several P2P resource discovery algorithms appear in the literature, trying to alleviate the excessive volume of traffic produced during flooding [11]. One of the first alternatives proposed was random walks. Each node forwards each query it receives to a single neighboring node chosen at random, a method that generates very little traffic but suffers from reduced network coverage and long response time. As an alternative, multiple random walks have been proposed, where the querying node starts simultaneously k parallel random walkers. Although compared to a single random walk this method has better behavior, it still suffers from low network coverage and long response time compared to flooding.

Hybrid methods that combine flooding with random walks have been proposed in [9]. Schemes like Directed Breadth First Search (DBFS) forward queries only to those peers that have provided results to past requests, under the assumption that they will continue to do so. Interest-based schemes aim to cluster together peers with similar content, under the assumption that those peers are better suited to serve each other's needs. In another family of algorithms, query messages are forwarded selectively to part of a node neighbors based on predefined criteria or statistical information. For example, each node selects the first k highest capacity nodes or the k connections with the smallest latency to forward new queries [19]. A somewhat different approach named forwarding indices builds a structure that resembles a routing table at each node [6]. This structure stores the number of responses returned through each neighbor on each one of a preselected list of topics. Other techniques include query caching, or the incorporation of semantic information in the network [7, 15].

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 possess files belonging to the corresponding category. Subnetworks are not necessarily distinct.

A system that exploits this approach is the Semantic Overlay Networks (SONs) [7]. 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. An approach that overcomes this semantic categorization method has been proposed in [15]. In SONs, an already existing, online, music categorization database is used. This database adds a centralized component in the operation of the network. 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.

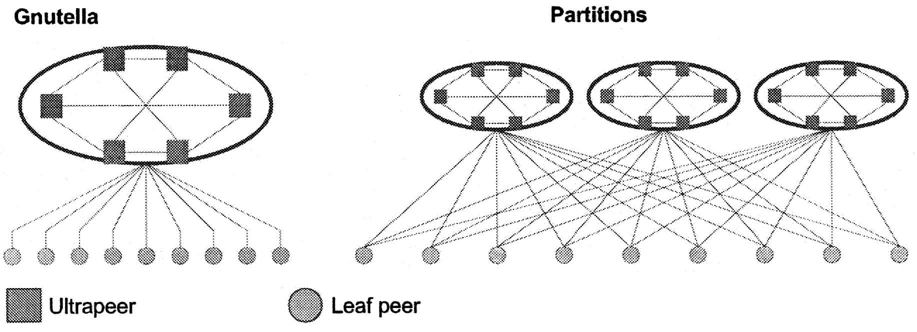

Fig. 2. Illustration of the Gnutella network and the Partitions design.

An interesting survey on P2P resource discovery methods which emphasis to Grid systems is presented in [18].

3. Hashed-keyword overlay partitioning

One way to reduce the cost of flooding is to partition the overlay network into a small number of *distinct subnetworks* and to restrict the search for individual request to one network partition. The *Partitions* scheme, proposed in this section, enriches unstructured P2P systems with appropriate data location information in order to enable more scalable resource discovery, while not affecting at all the self-healing properties and the inherent robustness of these systems. More specifically, Partitions employ a uniform hash function to map each keyword to an integer, from a small set of integers. Each integer defines a different category. Thus, keywords are categorized instead of content (files names). This method is more generic compared to the SONs method since it can be applied to any type of content (not only music files) and does not require semantic categorization of content.

The keyword categories are exploited in a 2-tier architecture, where nodes operate as Ultrapeers and/or Leaves. The partitioning of the network is performed as follows (see Fig. 2):

- Each Ultrapeer is randomly and uniformly assigned responsibility for a single keyword category. Ultrapeers responsible for the same category form a random subnetwork. As a consequence, the network overlay is partitioned into a small number of distinct subnetworks, equal to the number of available categories.
- Leaves randomly connect to one Ultrapeer per subnetwork. Each Leaf sends to each Ultrapeer it is connected to all its keywords, in the form of a bloom filter, that belong to the Ultrapeer's category. Thus, an innovative index splitting technique is used. Instead of each Leaf sending its entire index (keywords) to each Ultrapeer it is connected to, each Leaf splits its index

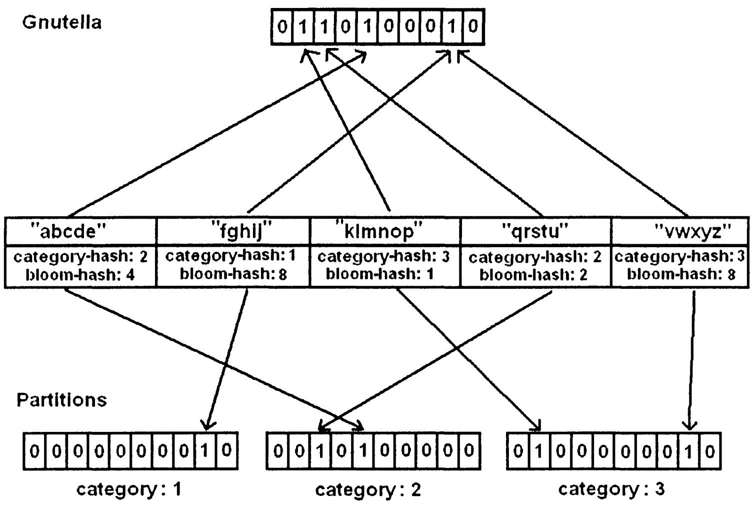

Fig. 3. Gnutella and Partitions bloom filters.

based on the defined categories and constructs a different bloom filter for each keyword category. Each bloom filter is then sent to the appropriate Ultrapeer. An illustration of this technique can be found in Fig. 3. It is shown in this figure that while in Gnutella there is one bloom filter for all keywords, in Partitions there is one bloom filter for each one of the three keyword categories. For example, keywords "abcde", "fghij", "klmnop", "qrstu", and "vwxyz" are mapped to positions 4, 8, 1, 2, and 8 of the Gnutella bloom filter. These same keywords are mapped to the same positions but in different bloom filters in the Partitions design. Keywords "abcde" and "qrstu" are mapped to the category 2 bloom filter (positions 4 and 2 respectively), keyword "fghij" is mapped to the category 1 bloom filter (position 8) while keywords "klmnop" and "vwxyz" are mapped to the category 3 bloom filter (positions 1 and 8 respectively).

Fig. 2 illustrates schematically the Partitions design and its main structural difference to the Gnutella architecture. It can be easily seen in this figure that the Ultapeer layer in Partitions is separated into a certain number of distinct subnetworks and that each Leaf node is connect to one Ultrapeer per subnetwork.

We should emphasize that in this design Ultrapeers are de-coupled from their content, meaning that peers operating as Ultrapeers will have to also operate as Leaves at the same time in order to share their own content, which spans several categories. Furthermore, even though in this design each Leaf connects to more than one Ultrapeers, the volume of information it collectively transmits to all of them is roughly the same since each part of its index is sent to a single Ultrapeer.

The Partitions scheme is demonstrated in Fig. 2. The unstructured overlay network is partitioned into distinct subnetworks, one per defined category. A search for a keyword of a certain category will only flood the appropriate subnetwork and avoid contacting Ultrapeers in any other network partition. The benefit of this is two-fold. First, it reduces the size of the search for each individual request. Secondly, it allows each Ultrapeer to use all its Ultrapeer connections to connect to other Ultrapeers in the same network partition, increasing the efficiency of 1-hop replication at the Ultrapeer level. One-hop replication dictates that each Ultrapeer contains an index of the contents of its neighbouring Ultrapeers (including the contents of their Leaves).

There are, however, two 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 collectively upon connection as before, it requires more keepalive messages to ensure that its Ultrapeer connections are still active. Keepalive 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 the need for keepalive messages. Experimental results presented in the next section demonstrate that the maintenance cost of this design due to the extra keepalive messages is minimal and is outweighted by the benefit in message reduction.

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 search results 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 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 contain keywords of one category, which makes them more sparse, thus reducing the probability of a false positive. Furthermore, the most rare keyword can be used to direct the search, further increasing the effectiveness of the search method.

4. Simulation results

In this section, we 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 cost. The first metric measures the traffic load of the flooding process in the entire network, while the second metric focuses on the load experienced by a(ny) single Ultrapeer. We performed several simulations, varying the number of Leaves per Ultrapeer and the number of categories/subnetworks. In addition, apart from the Gnutella and the Partitions scheme, we also run simulations on a modified version of the Partitions scheme, called *Replication*. The only difference between Partitions and Replication is the fact that Leaves send a complete (containing all keywords, regardless of category) bloom filter to all the Ultrapeers they connect to. In both cases, Partitions and Replications, we measured the operational (messages for flooding) and the maintenance (keepalive messages) costs.

In all simulations, we assumed a Leaf population of 2 million, a number reported by LimeWire Inc [2]. Each peer contains a number of files (and hence keywords) derived from a distribution also obtained from real-world measurements [13]. Each Ultrapeer in the Gnutella network serves, on average, 30 Leaves, a number obtained from real-world measurements [17]. In addition, we also performed simulations for 10 and 60 Leaves per Ultrapeer. Each Ultrapeer in the Partitions design serves a number of Leaves equal to the number of categories/subnetworks multiplied by the number of Leaves per Gnutella Ultrapeer. Since each Leaf sends each Ultrapeer a fraction of its keywords (namely, a fraction equal to the number of categories), this results in a roughly similar number of keywords in a Partitions Ultrapeer as in a Gnutella Ultrapeer. The number appearing next to each scheme name in the legend of the graphs denotes the number of full indices stored in each Ultrapeer. For instance, Gnutella 30 is the classic Gnutella algorithm (each Ultrapeer serves 30 Leaves). In Partitions 10, each Ultrapeer serves 100 Leaves, receiving one-tenth of each one's index, thus adding up to 10 full indices. In all schemes, each Ultrapeer has 30 Ultrapeer-neighbours. Finally, we have used a Zipf-like distribution for the popularity of keywords, both in the peers' filenames and in the queries. The end-result is the one reported in [17], with Leaves having, on average, a 3% full bloom filter and a Gnutella Ultrapeer with 30 Leaves having a 65% full bloom filter. The size of the bloom filter array is the same as the one used in Gnutella today, which is 2 to the power of 16 (65536). The percentage of fullness of the bloom filter is the probability that an one-keyword query will be forwarded to a Leaf or a last-hop Ultrapeer. Figs 4(a) and 5(a) show the efficiency of the Partitions and Replication scheme, respectively, in reducing the cost of the flooding.

4.1. *Keyword-based queries*

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. In addition, it follows from the results that the benefits of being able to filter using all keywords in a query when forwarding to the Leaf layer (Replication scheme) is small compared to the increased maintenance cost. We would like to emphasize that each Partitions bloom filter (i.e. containing keywords of a certain category) has the length of a 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

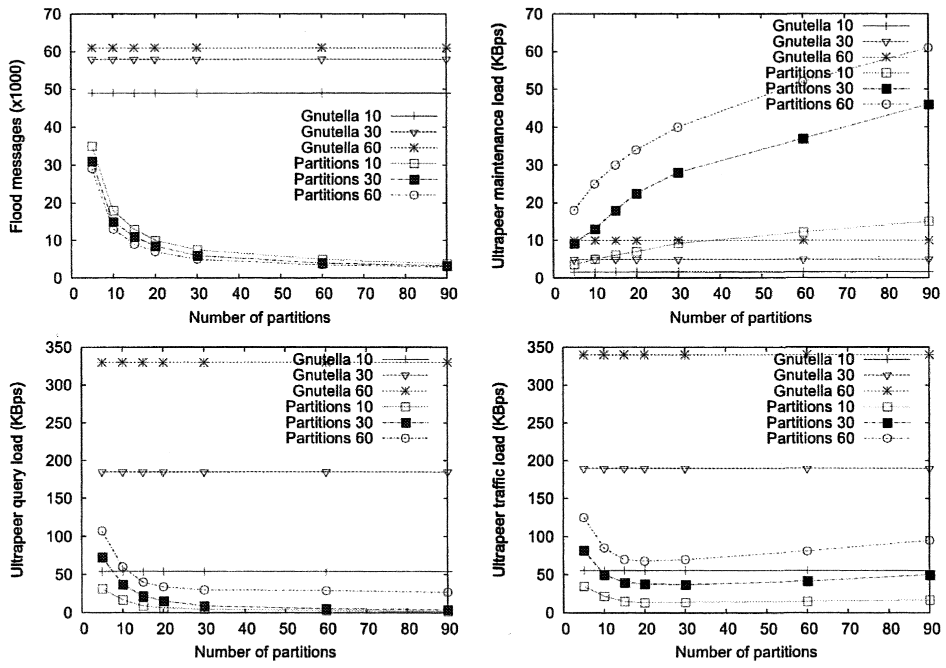

Fig. 4. Keyword-based queries, Gnutella vs Partitions. (a) Number of messages generated in one flood. (b) Ultrapeer maintenance load. (c) Ultrapeer query load. (d) Total Ultrapeer traffic load.

because sparser bloom filters are compressed more efficiently.

Another interesting observation is the fact that Gnutella 10 outperforms Gnutella 30, even though the Ultrapeer layer is 3 times bigger. This is easily explained by the fact that the Ultrapeers' bloom filters are more empty (since they contain the aggregate bloom filters of only 10 Leaves instead of 30). This means that the filtering during the last hop of flooding is more accurate, increasing the efficiency of 1-hop replication.

We then focus on the traffic load experienced by a single Ultrapeer. In all 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 30 Ultrapeer neighbors. In addition, we assumed that, periodically (every 10 seconds), 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. We used a keep-alive message size of 50 bytes and a query message size of 80 bytes, as indicated by the Gnutella Protocol Specification. In addition, for every 1400 bytes for each message sent, we added 40 bytes for the TCP and IP header.

We have used the code by LimeWire [2], the most popular Gnutella client,

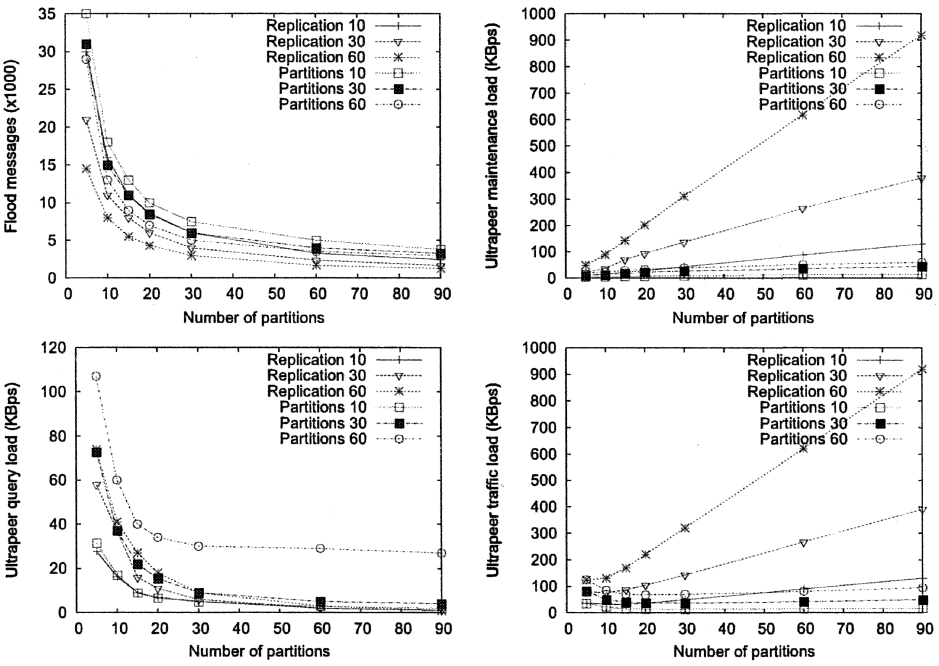

Fig. 5. Keyword-based queries, Partitions vs Replication. (a) Number of messages generated in one flood. (b) Ultrapeer maintenance load. (c) Ultrapeer query load. (d) Total Ultrapeer traffic load.

to construct the bloom filter of each Leaf. For each peer, we then extracted a number of files (equal to the number of files assigned to that peer) from a list of filenames obtained from the network by a Gnutella crawler developed in our lab. Those filenames 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 needed was just its size (and the ratio of "fullness").

We run three types of simulations measuring (1) the maintenance traffic (bloom filter and keep-alive messages exchange), (2) the query traffic and (3) the total traffic. In the last case, we observed the nearly total lack of keep-alive messages, due to the implicit use of query traffic for indication of liveness. In all the graphs presented, the x axis corresponds to the number of partitions (keyword categories).

Figs 4(b) and 5(b) show the increase in the maintenance load when employing our scheme. As expected, the maintenance load in the Replications scheme increases linearly with the number of available partitions. This is because, the number of Leaves each Ultrapeer has to serve also increases linearly with the number of partitions. We can see that this is not the case with the Partitions scheme, where, even though the number of Leaves per Ultrapeer also increases, the size of the index submitted by each Leaf decreases.

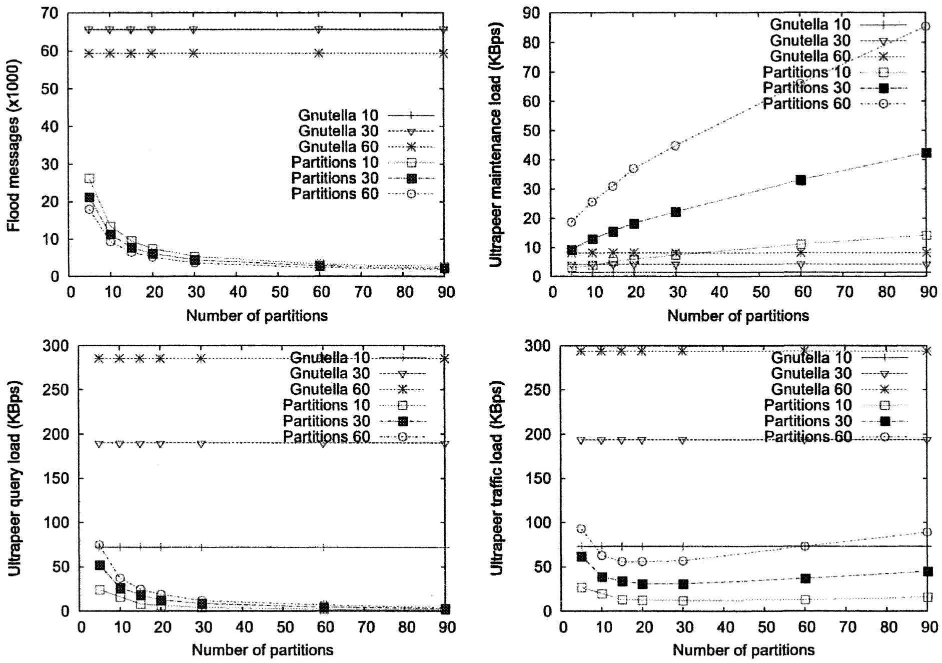

Fig. 6. Hash-based queries, Gnutella vs Partitions. ((a) Number of messages generated in one flood. (b) Ultrapeer maintenance load. (c) Ultrapeer query load. (d) Total Ultrapeer traffic load.

We then focused our attention to the query traffic load. Measurements conducted in our lab 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 one keyword, 30% contain two, another 20% contain three and finally a 30% contain four keywords. 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 is achieved first. As we mentioned before, a flood that aims to reach the entire network would need to reach $\frac{1}{10}$th of the Gnutella's network (or a Partitions' subnetwork) during all hops of flooding except the last thanks to the 1-hop replication. 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. Figs 4(c) and 5(c) show the comparison in the traffic load of Gnutella, Partitions and Replication.

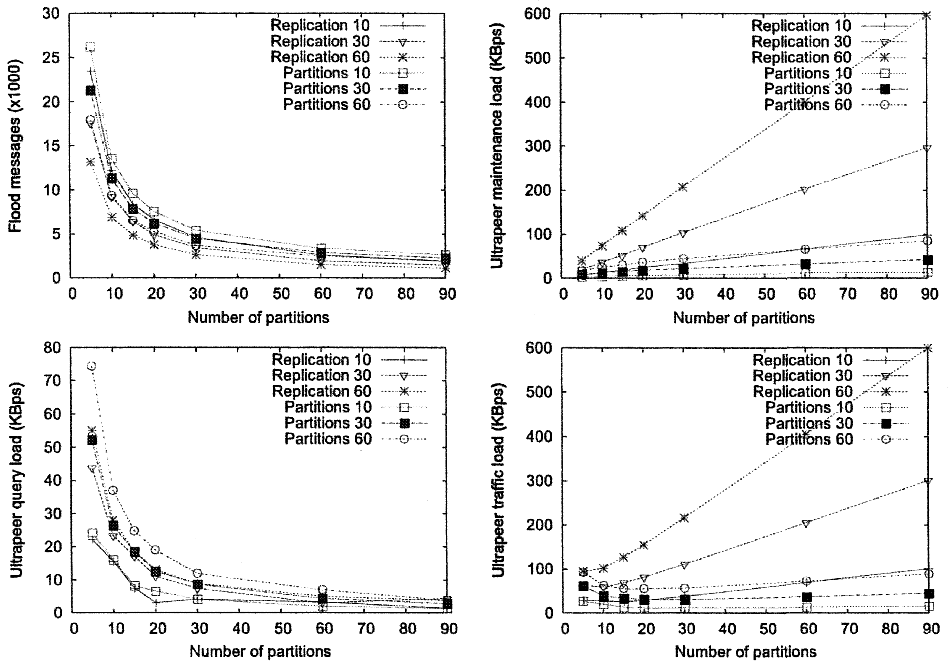

Fig. 7. Hash-based queries, Partitions vs Replication. (a) Number of messages generated in one flood. (b) Ultrapeer maintenance load. (c) Ultrapeer query load. (d) Total Ultrapeer traffic load.

Finally, Figs 4(d) and 5(d) show the total traffic load experienced by a single Ultrapeer. From these figures it is evident that Partitions outperform Gnutella in operational costs, in all cases.

4.2. *Hash-based queries*

All the results presented above regard keyword-based queries. However, there is a large number of queries in P2P networks in general which are not keyword-based. These are the hash-based queries. Those queries are looking for a file with a specific MD5 or SHA-1 hash value instead of one that contains some keywords in its filename. The main use of such queries is the location of alternate sources for download for a file that is already being transferred (and was probably located using keyword-based queries). Those alternate sources can be used for swarm downloading, that is downloading different parts of the same file from multiple sources, in parallel. Hash-based queries can also have other uses, such as locating illegal content (copyrighted material, illegal pornography, etc.) which usually exists in the system under misleading filenames. Another important use of hash-based queries is the detection of false results, to avoid downloading some content only to realize that it was something completely different than what its name indicated.

Depending on the uses of hash-based queries of interest to the system, the frequency of such queries may vary from low up to consisting the majority of the queries. Those queries can be thought of as one-keyword queries (the single keyword being the hash value in the query). Since our system especially excels in one-keyword queries, we conducted the same measurements as before, including hash-based queries in the experiments. Since the previous experiments contained a percentage of hash-based queries equal to 0, in the results presented below we have assumed a frequency equal to 50% (one hash-based query per normal query). This will give us the behavior of the system on the other side of the spectrum, since as we said the actual frequency of the hash-based queries depends on the importance of their uses for each system. Figs 6 to 7 show the results of the new experiments.

5. Conclusions

In this paper, 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 flooding.

References

[1] Gnutella 0.6 protocol specification.
http://rfc-gnutella.sourceforge.net/developer/stable/index.html
[2] Limewire Inc. http://www.limewire.com
[3] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, 1970.
[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, 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.
[5] V. Cholvi P. Felber, E. Biersack. Efficient search in unstructured peer-to-peer networks. Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures, 2004.
[6] A. Crespo, H. Garcia-Molina. Routing indices for peer-to-peer systems. Int. Conf. on Distributed Computing Systems (ICDCS'02), Vienna, Austria, 2002.
[7] A. Crespo, H. Garcia Molina. Semantic overlay networks for P2P Systems. Int. Conf. on Agents and Peer-to-Peer Computing (AP2PC 2004), New York, USA, 2004.
[8] A. Fisk. Gnutella Ultrapeer Query Routing, v. 0.1. LimeWire Inc. 2003.
[9] C. Gkantsidis, M. Mihail, A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. IEEE INFOCOM 2005, Miami, USA, 2005.
[10] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker. Search and replication in unstructured peer-to-peer networks. Int. Conf. on Supercomputing (SC 2002), Baltimore, USA, 2002.
[11] C. Papadakis P. Fragopoulou E. Athanasopoulos M. Dikaiakos, A. Labrinidis, E. Markatos. A feedback-based approach to reduce duplicate messages in unstructured peer-to-peer networks. Proc. of the CoreGRID Integration Workshop, 2005.
[12] A. Ratnasamy, P. Francis, M. Handley, R.M. Karp, S. Shenker. A scalable content-addressable network. ACM SIGCOMM 2001, pp. 161-172.

[13] R. Rejaie, S. Zhao, D. Stutzbach. Characterizing files in the modern Gnutella network: A measurement study. Proc. SPIE/ACM Multimedia Computing and Networking, 2006.

[14] A. Rowstron, P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, Nov. 2001.

[15] K. Sripanidkulchai, B. Maggs, H. Zhang. Efficient content location using interest-based locality in peer-to-peer Systems. IEEE INFOCOM 2003, San Franciso, USA, 2003.

[16] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. SIGCOMM'01, San Diego, California, USA, August 2001.

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

[18] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, S. Haridi. Peer-to-Peer Resource Discovery in Grids: Models and Systems. Future Generation Computer Systems 23, 864-878, 2007.

[19] D. Tsoumakos, N. Roussopoulos. A Comparison of peer-to-peer search methods. Int. Workshop on the Web and Databases (WebDB 2003), San Diego, USA, 2003.

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

[21] B. Yang, H. Garcia-Molina. Designing a super-peer network. Proc. Int. Conference on Data Engineering (ICDE 2003), pp. 49-60, 2003.

[22] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J.D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications, vol. 22(1), Jan. 2004.