The Design and Evaluation of a Self-Organizing Superpeer Network

Paweł Garbacki, Dick H.J. Epema, and Maarten van Steen

Abstract—Superpeer architectures exploit the heterogeneity of nodes in a peer-to-peer (P2P) network by assigning additional responsibilities to higher capacity nodes. In the design of a superpeer network for file sharing, several issues have to be addressed: how client peers are related to superpeers, how superpeers locate files, how the load is balanced among the superpeers, and how the system deals with node failures. In this paper, we introduce a self-organizing superpeer network architecture (SOSPNET) that solves these issues in a fully decentralized manner. SOSPNET maintains a superpeer network topology that reflects the semantic similarity of peers sharing content interests. Superpeers maintain semantic caches of pointers to files, which are requested by peers with similar interests. Client peers, on the other hand, dynamically select superpeers offering the best search performance. We show how this simple approach can be employed not only to optimize searching, but also to solve generally difficult problems encountered in P2P architectures such as load balancing and fault tolerance. We evaluate SOSPNET using a model of the semantic structure derived from eight-month traces of two large file-sharing communities. The obtained results indicate that SOSPNET achieves close-to-optimal file search performance, quickly adjusts to changes in the environment (node joins and leaves), survives even catastrophic node failures, and efficiently distributes the system load taking into account superpeer capacities.

Index Terms—Peer to peer, superpeer architectures, semantic clustering, self-organizing systems.

1 Introduction

A significant amount of work has been done in the field of optimizing the performance and reliability of content sharing peer-to-peer (P2P) networks [37], [47]. Among the proposed optimizations, the concept of leveraging the heterogeneity of peers by exploiting high-capacity nodes in the system design has proved to have great potential [58]. The resulting architectures break the symmetry of pure P2P systems by assigning additional responsibilities to high-capacity nodes called *superpeers*. In a superpeer network, a superpeer acts as a server to *client (ordinary, weak) peers*. Weak peers submit queries to their superpeers and receive results from them. Superpeers are connected to each other by an overlay network of their own, submitting and answering requests on behalf of the weak peers.

Several protocols have been proposed to exploit superpeers [39], [41], [57], [58]. We add to this work the design of a superpeer network capable of optimizing the relationships between peers taking into account their content interests as deduced from their (possibly changing) behavior. We call our architecture the *Self-Organizing Superpeer Network* (SOSPNET) because the relationships between peers are discovered, maintained, and exploited automatically, without any need for user intervention or explicit mechanisms.

- *P. Garbacki is with Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043. E-mail: pawelg@gmail.com.*
- *D.H.J. Epema is with the Department of Computer Science, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands. E-mail: d.h.j.epema@ewi.tudelft.nl.*
- *M. van Steen is with the Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. E-mail: steen@cs.vu.nl.*

Manuscript received 5 Dec. 2007; revised 10 Apr. 2009; accepted 19 Aug. 2009; published online 13 Oct. 2009.
Recommended for acceptance by C.-L. Wang.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-12-0623.
Digital Object Identifier no. 10.1109/TC.2009.157.

While some researchers have focused on exploiting static properties of shared data [45], [51], [59], also the possibility of utilizing patterns in dynamic peer behavior has attracted the attention of the research community [11], [14], [52], [53], [55]. Such patterns in peer behavior have been reported by several measurement studies [24], [25], [26], [28], which have revealed correlations between the search requests made by users of popular P2P systems. It was observed that the performance of locating content can be greatly improved [20], [33] by grouping peers interested in similar files and routing their search requests within these groups. The semantic relationships between peers and files can be discovered relatively easily [11], [55]. The biggest challenge is, thus, to build an architecture that maintains and exploits the discovered *semantic structure* existing in all these semantic relationships. In this paper, we present the design and evaluation of a P2P architecture that combines the homogeneity of peer interests with the heterogeneity of peer capacities to solve the problem of efficient peer relationship management.

The design of our self-organizing superpeer network is guided by the following requirements: First, SOSPNET should be self-organizing in that it is able to discover and exploit the semantic structure present in the network, no matter what the initial topology is. Second, a new peer joining the network does not need to have any knowledge about the system; the longer a peer stays in the system, the more information it can collect and exploit for improving the performance of its searches. Third, the time it takes a new peer to achieve its optimal performance should be minimized.

SOSPNET uses two-level semantic caches deployed at both the superpeer and the weak peer level to maintain relationships between related peers and files. The cache maintained by a superpeer contains references to those files

that were recently requested by its weak peers, while the cache of a weak peer stores references to those superpeers that satisfied most of its requests. We propose a novel *mixed* caching policy that combines the advantages of the traditional least frequently used (LFU) and least recently used (LRU) policies to improve the cache hit rates for less popular files. Furthermore, SOSPNET incorporates in its design a mechanism for balancing the load among superpeers. Load balancing is fully integrated with the content search algorithm and does not require any additional information exchange between superpeers nor a separate, external control component. The load balancing decisions are made independently by individual superpeers based on local information.

We also introduce a general performance model of a P2P system with semantic relations between peers and files based on two 8-month-long measurements of a large P2P network. From the model, we derive a bound on the search performance of a superpeer network using semantic caches. In a series of simulations, we show that the performance of SOSPNET is very close to the theoretical bound. In addition, we evaluate in our simulations the fault tolerance, the clustering properties, and the load balancing capabilities of SOSPNET. Finally, we compare SOSPNET with alternative architectures, assess its responsiveness to peer joins and leaves, and measure the time needed to find an optimal set of connections between peers, which all help in understanding how the system would perform in a real environment.

The rest of the paper is organized as follows: In Section 2, we specify the problem domain and scope of the presented system. Section 3 describes in detail the architecture of our self-organizing superpeer network. Section 4 introduces a model of P2P networks with semantic relationships between peers and files based on real-world traces. This model is used in Section 5 to evaluate the performance of our architecture. Section 6 summarizes the related work. The paper concludes in Section 7 by exploring some opportunities for future work.

2 Organizing Peer Relationships

The vast majority of mechanisms for optimizing different performance aspects of P2P networks rely in one way or another on organizing the relations between peers. The relationships are organized by defining for each peer the set of other peers, called its *neighbors*, it interacts with.

In symmetric P2P networks such as Gnutella [2] and Freenet [16], any two peers are potential neighbors. In hybrid approaches such as Napster [6], all peers have a single neighbor—a central server that keeps information on all peers and responds to requests for that information. In superpeer networks [58] such as Kazaa [4], Gnutella ultrapeers [49], and Chord superpeers [38], neighbors are selected from the set of high-capacity peers called superpeers; low-capacity peers—the client peers—cannot become neighbors.

In this paper, we aim at solving the problems of the existing superpeer networks related to the issue of establishing relationships between peers. Before presenting our approach, we identify the weak points of existing superpeer architectures. Each of the popular superpeer protocols proposed in the literature, including Kazaa, Gnutella ultrapeers, and Chord superpeers, makes at least one of the following three assumptions:

1. Every peer is assigned to a fixed, very small number (usually one) of superpeers. Consequently, superpeers become bottlenecks in terms of fault tolerance. Restoring the system structures such as routing tables back to a consistent state after a superpeer crash requires a considerable effort.
2. Peers are assigned to superpeers randomly and statically. The randomness of the assignment is explicit (as in Gnutella) or implicit (as in Chord, where the superpeer selection is based on peer identifiers, which are selected randomly). This static assignment does not adapt to changes in the network structure or in peer characteristics (e.g., content interests).
3. The peer-to-superpeer assignment has the so-called *all-or-nothing* property. When a peer connects to a superpeer, the latter takes responsibility for all the content stored at the peer. Such an assignment does not take into account the possible diversity of the peer's interests, and makes balancing the load among the superpeers difficult.

In the rest of the paper, we show how to overcome all these limitations by introducing our self-organizing superpeer architecture SOSPNET.

3 The Architecture of the Self-Organizing Superpeer Network

In this section, we present the SOSPNET system design. After a general overview of the SOSPNET architecture, we discuss in detail the employed data structures and protocols.

3.1 Architecture Overview

The basic idea behind the system architecture we propose is simple and intuitive. Weak peers with similar interests are connected to the same superpeers. As a consequence, superpeers get many requests for the same files. The request locality suggests the usage of caches that store the results of recent searches. But not only superpeers are responsible for discovering semantic structure in the network. We also allow weak peers to collect statistics about the content indexed by the superpeers. Having this information, weak peers can make local decisions about which superpeers to connect to.

In our architecture, superpeers store the information about the location of the content recently requested by their weak peers. Weak peers, on the other hand, sort the superpeers known to them according to the number of positive responses to their queries, and prefer to connect to superpeers that have satisfied most of their requests.

To accelerate the process of grouping peers with similar interests under the same superpeers, we allow weak peers to exchange their lists of superpeers. More precisely, if a search succeeds, the requesting peer asks the peer that has the requested file for its list of top-ranked superpeers. This list is then merged with the list of superpeers known to the requesting peer. The intuition here is that if both peers were interested in the same file, then it is highly probable that they will share interest for more files in the future.

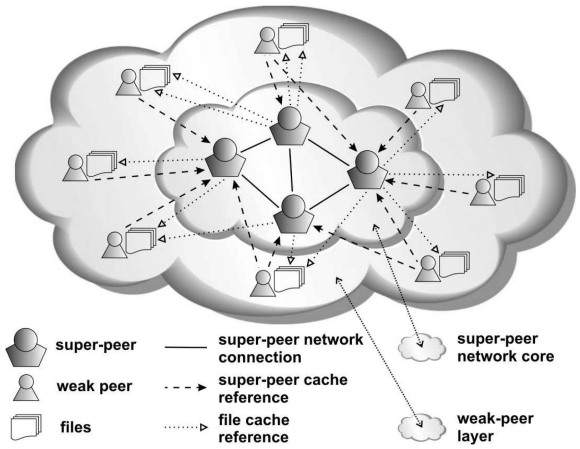


Fig. 1. The structure of SOSPNET.

3.2 System Model

The information stored at a node in our system depends on the type of this node. Each weak peer maintains a *superpeer cache*, which contains the identities of superpeers (e.g., their IP addresses and port numbers). Each superpeer has a *file cache* of pointers to files stored at the weak peers. The relationships between SOSPNET peers are presented in Fig. 1.

All items in the superpeer and file caches are assigned *priorities*, which are nonnegative integer numbers. The priority determines the importance of a particular item, the higher the better. The initial priority assigned to a data item when it is added to the cache and the way the priority is modified upon a cache hit are determined by the *caching policy*. There are two situations when the priorities are taken into account. First, when the cache capacity is exceeded, the item with the lowest priority is removed. Second, the priorities are used for optimizing query routing. Details are presented in Section 3.4.

The last element of Fig. 1 that has not been mentioned until now is the network interconnecting the superpeers. We do not specify precisely which P2P protocol should be used here. We assume, however, that this protocol can efficiently deal with frequent changes of the information stored at the superpeers. Additionally, we require that the probability that a search succeeds is high when the requested information is present at least at one of the superpeers. Examples of protocols satisfying these criteria are Gnutella and epidemic-based approaches such as SCAMP [21].

The load balancing mechanisms of SOSPNET require introducing some specific terminology. We assume that each superpeer specifies its *capacity* as a value in the interval $(0, 1]$, with higher values assigned to more capable peers. We do not make any further assumptions about the superpeer capacities, which may either reflect static node properties (e.g., CPU speed) or change dynamically based on the current situation in the system (e.g., available bandwidth). The particular method of computing the capacity values falls outside the scope of this paper. The *current load* of a superpeer is computed by counting the number of requests processed by the superpeer in a certain time frame called the *request history window*. The size of the request history window is the same for all superpeers, thus making the current-load values consistent across all superpeers in the system. However, the values of the current load of the superpeers cannot be compared directly, as different superpeers may have different capacities. Instead, we compute for each superpeer the *effective load* by dividing the current load by the capacity of the superpeer. A superpeer controls its load simply by dropping some of the search requests it receives. The *accepted load* is defined as the fraction of accepted search requests of those sent to the superpeer.

3.3 Two-Level Caching

The two-level caching architecture represented by superpeer and file caches allows us to separate caching policies that can be optimized for a peer role. In SOSPNET, the superpeer caches of the weak peers and the file caches of the superpeers are controlled according to different caching policies.

The priority of a superpeer in a superpeer cache is increased by one after every positive feedback provided by this superpeer. This leads to the *in-cache LFU* [10] policy. The benefit of LFU is its inherent memory property—the priority of a superpeer is determined by the number of successful feedbacks it has provided in the past. The priority changes slowly, so one positive response from an unknown superpeer will not discredit a well-proven superpeer that satisfied many requests in the past, which would be the case if one used a memoryless policy [32] such as LRU.

The caching policy employed for file caches should meet some specific requirements. First, similar to LRU, the file caches of the superpeers have to adapt fast to the changing needs of the weak peers. This is important particularly in the initial stage of the superpeer lifetime, when it is contacted by random peers. Second, like LFU, the file caching policy should keep track of long-term file popularity. Addressing the specific requirements of file caches, we propose a *mixed* caching policy that combines the desired properties of LRU and LFU. According to the mixed policy, if the file pointer is not yet present in the cache, then it is added to the cache with its priority one higher than the highest priority of all other cached items as in LRU. Otherwise, the priority corresponding to the file pointer is increased by one as in LFU. The high initial priority of the inserted item and the slow alteration of the priorities of items in the cache result in a better caching performance for less popular files as we show in Section 5.

3.4 Search Protocol

Peers use the information collected during past searches to improve the performance of future requests. The contents of the superpeer and file caches are reorganized depending on the feedback provided by peers involved in the search process.

The pseudocode of the search algorithm employed in our self-organizing superpeer network presented in Fig. 2 is divided into four subroutines. The superpeer cache of peer p is denoted by $p.S$, while the file cache of superpeer s is represented by $s.F$.

The main search algorithm is the function **peer_search**. When a weak peer p looks for a file f, it first checks the file caches of the superpeers known to it (line 2). Note that p starts with the superpeers with the highest priorities. When

```
1  peer_search(p : peer, f : file_name):
2      for s in p.S ordered according to decreasing priorities do
3          q ← super-peer_local_search(s,f)
4          if super-peer_local_search succeeded then
5              t ← s
6              break
7      if f was not found until now then
8          s ← super-peer in p.S selected randomly with
                probability proportional to its priority in p.S
9          < q, t > ← super-peer_search(s,f)
10         if super-peer_search did not succeed then
11             return ERROR "File f not found"
12     if p.S contains t then
13         increase the priority of t in p.S
14     else
15         insert t into p.S
16     merge_super-peer_caches(p, q)
17     return q

18 super-peer_local_search(s : super-peer, f : file_name):
19     if an entry < f, q > exists in cache s.F then
20         increase the priority of < f, q > in s.F
21         return q
22     else
23         return ERROR "File f not found"

24 super-peer_search(s, f):
25     perform a search in the super-peer network to locate a
       super-peer t which has an entry < f, q > in its cache
26     if search succeeded then
27         insert < f, q > into s.F
28         return < q, t >
29     else
30         return ERROR "File f not found"

31 merge_super-peer_caches(p : peer, q : peer):
32     for s in q.S do
33         if p.S contains s then
34             increase the priority of s in p.S
35         else
36             insert s into p.S
```

Fig. 2. Pseudocode of the search protocol in SOSPNET.

the file is found (line 4), a pointer to superpeer s that knows the location of f is stored for future reference (line 5). However, if the file was not found with this method (line 7), the search request is forwarded to one of the superpeers in p's superpeer cache selected according to a random distribution biased toward superpeers with higher priority (line 8). This superpeer is further responsible for locating file f. If the search succeeds, a pair $<q, t>$, where q is a peer that has f and t is a superpeer that has a pointer $<f, q>$ in its file cache, is returned to p (line 9). At this point, the self-(re)organization process begins. This process is performed in two stages. First, peer p increases the priority of the superpeer t that satisfied the search request (lines 12-15). As a consequence, in the future, p will direct more of its requests to t. Second, p integrates the list of superpeers kept by the weak peer q with its own superpeer cache (line 16). We exploit here a simple, yet powerful principle called *interest-based locality* [50], which postulates that if p and q are interested in the same file, it is very likely that more of their requests will overlap. It is thus beneficial for both p and q to use the same set of superpeers.

The algorithm of the **superpeer_local_search** is straightforward. The search succeeds only if a pointer to file f is present in the file cache of superpeer s (line 19). Before returning the peer q that possesses file f (line 21), the priority of the corresponding cache item is increased (line 20).

The function **superpeer_search** performs the search in the superpeer network (line 25). Upon receipt of the search results, a pointer to the requested file f and to the peer q holding file f is added to the file cache of s (line 27). The return value of the function (line 28) contains not only the peer q, but also the superpeer t that has a pointer to f in its file cache.

The last function presented in Fig. 2, **merge_super-peer_caches**, takes two parameters representing two peers p and q. The superpeer cache of peer p is updated with the content of q's superpeer cache (lines 32 and 33). The functionality of merging the superpeer caches is not crucial for the system operation, but it accelerates the process of grouping weak peers under the same superpeers, which improves the search performance.

3.5 Insert Protocol

The file-insert protocol deployed by SOSPNET is very simple. Once in a while, each weak peer sends information on the files which it possesses to one of the superpeers in its superpeer cache. This superpeer is selected randomly with a probability proportional to its priority in the superpeer cache of the weak peer.

3.6 Balancing the Load among Superpeers

Load balancing is critical to the availability, accessibility, scalability, and throughput of a P2P system. Poor load balancing may gradually transform the superpeer network into a backbone network as was observed for Gnutella [13]. The idea here is to avoid overloading individual superpeers, which is the case when some superpeers are getting significantly more queries than others.

Before describing the load balancing mechanism of SOSPNET, we first define the requirements of load balancing for a superpeer network in general. A minimal requirement is to prevent situations in which the load imposed on a superpeer exceeds its capacities. A more advanced load balancing solution can further guarantee that the load assigned to each superpeer is proportional to its capacity. Finally, the performance overhead and implementation burden incurred by adding the load balancing extensions should be low. In the remainder of this section, we show how the above goals can be easily achieved by exploiting the properties of the self-organizing superpeer network.

At first sight, the load balancing problem that we face in the SOSPNET design seems to be more difficult than in other superpeer networks because the SOSPNET superpeers do not explicitly know their weak peers. Furthermore, in the SOSPNET architecture, the assignment of weak peers to superpeers is not fixed. As a consequence, the superpeers cannot transfer weak peers between each other without the active cooperation of the weak peer layer. Being aware of these limitations, we have built into the search protocol a mechanism that indirectly influences the set of superpeers contacted by the weak peers by discouraging directing requests to overloaded superpeers.

The basic idea behind the load balancing mechanism of SOSPNET relies on the observation that a superpeer may control the number of received requests by affecting its

```
18   super-peer_local_search(s : super-peer, f : file_name):
18.1     r ← random value from range (0, 1)
18.2     if r > s.accepted_load then
18.3         return ERROR "Super-peer s overloaded"
18.4     add request timestamp to request history window s.W
19       if an entry < f, q > exists in cache s.F then
         ...

24   super-peer_search(s, f):
         ...
26       if search succeeded then
26.1         update_accepted_load(s, t)
27           insert < f, q > into s.F
         ...

37   update_accepted_load(s : super-peer, t : super-peer):
38       s.requests ← number of requests in window s.W
39       t.requests ← number of requests in window t.W
40       s.effective_load ← s.requests/s.capacity
41       t.effective_load ← t.requests/t.capacity
42       Δ ← (t.effective_load − s.effective_load)/
             (t.effective_load + s.effective_load)
43       new_accepted_load ← s.accepted_load + Δ
44       if new_accepted_load > 1 then
45           new_accepted_load ← 1
46       if new_accepted_load < 0 then
47           new_accepted_load ← 0
48       s.accepted_load ← β · s.accepted_load+
                           (1 − β) · new_accepted_load
```

Fig. 3. Pseudocode of the superpeer load balancing protocol in SOSPNET.

priority in the superpeer caches of weak peers. An overloaded superpeer can simply start dropping some of the requests, effectively decreasing its priority in the superpeer caches of the requesting peers. As the priority of a superpeer has a direct impact on the probability of that superpeer being selected as a request target, the load imposed on the overloaded superpeer will gradually decrease. Note that if a superpeer s refuses to service a request, then eventually, the client peer will ask another superpeer t to search for the file and to subsequently store a reference in its file cache. In other words, t will eventually take over some of the file references that were cached by s.

The requirement that the load experienced by a superpeer is proportional to its capacity involves relating the effective load of that superpeer to the effective loads of other superpeers in the system. To avoid introducing an independent load-information exchange protocol, we let superpeers gather load values of other nodes while performing searches.

The integration of the SOSPNET load balancing functionality with the search protocol is presented in Fig. 3. The function **superpeer_local_search** of Fig. 2 is extended with lines 18.1-18.4, which control the fraction of requests that are handled by superpeer s. Only a fraction of $s.accepted\_load$ randomly selected requests are accepted and processed as described in Section 3.4. The remaining requests are dropped, forcing the requesters to decrease the priority of s. If a request is accepted, its time stamp is saved in the request history window denoted by $s.W$ (line 18.4). Request time stamps are used later for computing the current load of the superpeer.

The value of the accepted load of superpeer s is updated every time s discovers another superpeer t during the invocation of **superpeer_search** (line 26.1) by taking into account the load of t in the **update_accepted_load** function. The values of the effective loads of s and t, denoted by $s.effective\_load$ and $t.effective\_load$, respectively, are computed by dividing the numbers of requests in the request history windows of the two peers by their capacities (lines 38-41). The imbalance between the loads of s and t is then quantified by computing the relative difference Δ between the effective loads (line 42), which is then used to compute the value of the parameter $new\_accepted\_load$ of s (lines 43-47). Finally, the accepted load of s is updated by applying exponential smoothing [12] with weighting factor $\beta \in (0, 1)$ to the current value of the accepted load and $new\_accepted\_load$ (line 48). We use exponential smoothing instead of just replacing the accepted loads with the new values to avoid drastic changes in the accepted loads, giving the system time to adapt to the new settings [19].

In one specific case, the behavior of the load balancing algorithm can be confusing. Let's assume that superpeer s is overloaded and that it has in its cache the pointer $<f, q>$ to file f requested by p. The request will be forwarded to another superpeer, say t. Superpeer t will then perform a superpeer search, find s, store a pointer to f in its own cache, and return $<q, s>$ to p. As a consequence, peer p will increase the priority of s in its superpeer cache. This behavior is counterintuitive as p should be discouraged to contact s in the near future. However, the increase of the priority of s should be interpreted as a one-time trade-off. If a different peer subsequently sends a request for file f to t, superpeer t will satisfy the request from its local file cache. Our load balancing algorithm has, thus, the highly desired property of replicating file pointers cached by the overloaded superpeers at lighter loaded peers.

The load balancing scheme that we presented here is simple yet powerful and extremely flexible. While many state-of-the-art load balancing algorithms assume that all peers have equal capacities [30], [31], our self-organizing architecture can deal with arbitrary capacity values and even allows these values to be changed during system operation. The load imbalance caused by a change of the parameters of the superpeers is automatically taken into account, and the system gradually adapts to the new circumstances. Because neither the weak peers nor the file pointers have to be explicitly reassigned from one superpeer to another, no complex overlay infrastructure such as *virtual servers* [44], [60] or *buckets* of file identifiers [9] needs to be introduced.

3.7 Discussion

The flexibility built into SOSPNET eliminates all three weak points of existing superpeer designs mentioned in Section 2. First, by manipulating the size of its superpeer cache, a weak peer may decide to how many superpeers it is connected. The more connections maintained by the peer, the better is the resilience to crashes of multiple nodes.

Second, the problem of static peer-to-superpeer assignment is solved by the policy used for the superpeer caches. This policy prefers superpeers indexing content that is close to a user's interests. Possible changes in user interests or in

TABLE 1
Details of the Four Data Sets

Dataset name	Data collecting period	Number of semantic types	Number of files
suprnova.org	February 2004 – April 2004	198	24,081
piratebay.org	November 2005 – May 2006	40	164,821
suprnova_syn	—	198	24,081
piratebay_syn	—	40	164,821

the type of files cached at the superpeers result in restructuring the connections between peers.

Third, the all-or-nothing property is replaced with a property that we refer to as *partial responsibility*. The superpeers in our system index individually selected files rather than the entire set of files stored at their weak peers. This type of architecture can deal with a situation in which a single weak peer has files of different semantic types. Pointers to these files can then be cached by multiple superpeers.

4 Performance Model

In this section, we introduce a performance model of P2P systems with a semantic structure in the popularities of the files that are being shared and in the interests of the peers that are present in such systems. The semantic structure is described by a file access pattern and can be defined mathematically by the file request probabilities of the peers in the system. Below, we first show how the analytical model allows us to extract the semantic structure from real-world traces and how to generate it in synthetic data sets that can serve as inputs to simulations. We then apply this model to the two traces that we use in our performance evaluation in Section 5. Second, we show that there is an optimal arrangement of the items in the caches in the two-level caching scheme of SOSPNET that reflects the semantic structure of file popularities and peer interests, and we derive a theoretical bound on the performance of this arrangement. We use this bound when assessing the caching performance of SOSPNET in Section 5. We start this section with introducing some definitions and notation.

4.1 Definitions and Notation

It has been observed [20] that user content interests as well as file popularities in file sharing P2P networks are not independent. The similarities in user request patterns can be modeled by assigning *semantic types* to both files and peers.

We use the symbols D, U, and N to denote the total numbers of files (data items), peers (users), and semantic types in the system. The numbers of files and peers of (semantic) type n, $n = 1, \ldots, N$, are denoted by d_n and u_n, respectively. Each peer periodically generates a request for a file, which is selected according to a distribution that depends on the peer's type only. We denote by $p(m)$ the *type popularity* of semantic type m, $m = 1, \ldots, N$, which is defined as the overall probability that a random peer requests a file of type m. Similarly, we denote by $p(m, k)$ the *file popularity* of file k, $k = 1, \ldots, d_m$, of type m, which is the overall probability that a random peer asks for the kth file of type m. We assume that the types and the files per type are numbered in decreasing order of popularity. The *rank* of a file is its number in the ordering of all files if we list the semantic types, and within each semantic type the corresponding files, in decreasing order of popularity. The distribution of the type popularities can be computed from the file popularities in the following way:

$$p(m) = \sum_{k=1}^{d_m} p(m, k). \quad (1)$$

Further, we use the symbol $p_n(m, k)$ to denote the probability that a peer of type n requests the kth file of type m. We will define this probability in the next section.

4.2 Models of the Semantic Structure

In our experiments, we use four data sets to model file popularities. The first two of them are based on real-world traces, while the second two are created synthetically. The properties of the data sets are presented in Table 1. There are two reasons for using both models. First, a broader spectrum of the simulation data increases the credibility of our results. Second, we use the opportunity to assess the usefulness of synthetic data sets in the evaluation of system designs based on the semantic paradigm. The validation is performed by comparing the behavior of SOSPNET with real and synthetic data sets.

4.2.1 Model Based on Real Traces

Before presenting the method for computing the type popularities $p(m)$ and the file popularities $p(m, k)$ from the actual data traces, we describe how we have obtained these traces. For a period of eight months, we have collected the download statistics provided by the suprnova.org [7] and piratebay.org [8] Websites, which at the time of gathering the data were the most popular [29], [43] Websites used for searching files in the BitTorrent [17] network. BitTorrent is currently the largest P2P network with over one-third of the world's P2P traffic [42]. Each file registered at suprnova.org or piratebay.org is categorized by human volunteers called moderators. We treat the categories defined by the moderators as the semantic types.

For each file registered at suprnova.org or piratebay.org, we were able to obtain the number of peers downloading this file. The fraction of downloaders for a file can be interpreted as the file popularity $p(m, k)$. In order to reduce the influence of temporal interest localities such as flashcrowds [43] on the value of $p(m, k)$, we compute for each file the average number of downloads observed during the whole measuring period. The average is obtained by dividing the total number of downloads of the file by the duration of the period in which the file was

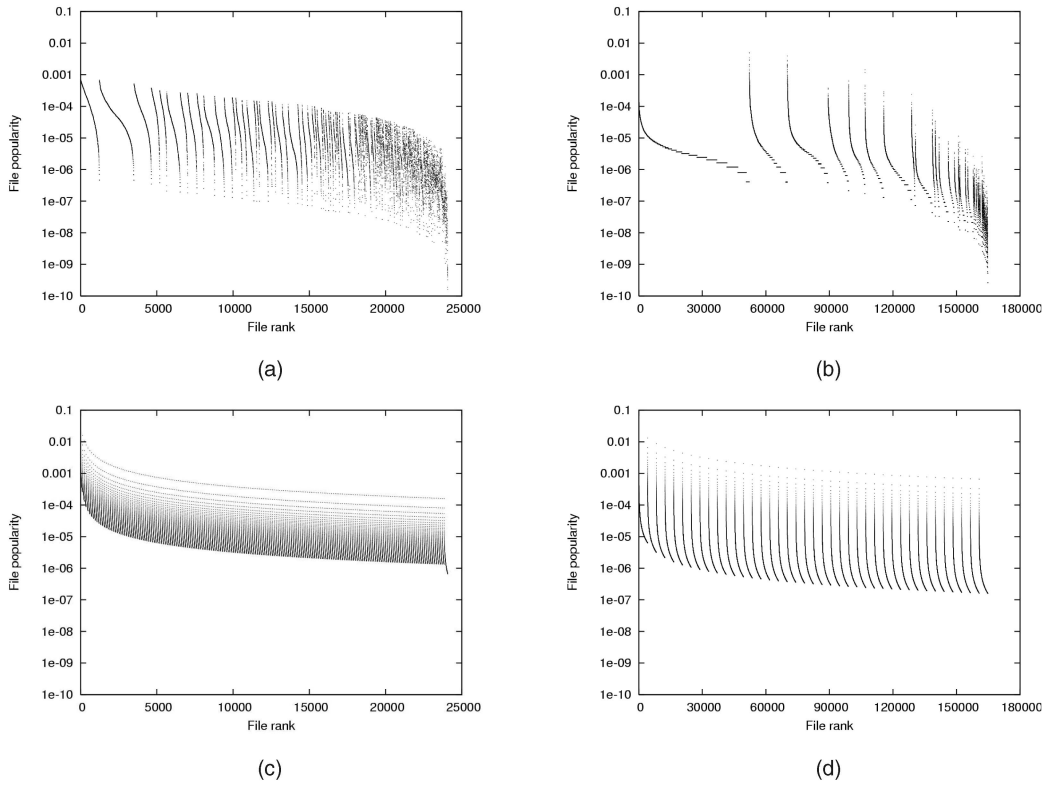


Fig. 4. The file popularity distribution $p(m,k)$ for the four data sets. The visible "tail" of the `suprnova_syn` synthetic distribution is caused by the fact that the last semantic type is represented by more files than the other types in the data set. (a) `suprnova.org`, (b) `piratebay.org`, (c) `suprnova_syn`, and (d) `piratebay_syn`.

accessible for download. The distributions of the file popularities $p(m,k)$ extracted from the `suprnova.org` and `piratebay.org` traces are presented in Figs. 4a and 4b, respectively, with the files ranked as explained above.

Although collecting the access patterns for a particular file is possible, obtaining complete statistics about the content downloaded by a specific peer is infeasible. First, many users are behind NAT boxes, which prevents us from discovering their peer IP addresses. Second, we cannot guarantee that a user is not using other Websites to look for the files he is interested in. Consequently, the behavior of users needs to be modeled synthetically, taking, however, file popularities into account.

We propose the following formulas for u_n and $p_n(m,k)$:

$$u_n = p(n) \cdot U, \tag{2}$$

$$p_n(m,k) = \begin{cases} (1-\alpha) \cdot p(m,k), & m \neq n, \\ \left[(1-\alpha) + \dfrac{\alpha}{p(n)}\right] \cdot p(n,k), & m = n. \end{cases} \tag{3}$$

Equation (2) explains that the number of users of a certain semantic type is proportional to the popularity of this type. The parameter $\alpha \in [0,1]$ in (3) characterizes how strong the interest of users is for files of their own type. When α equals 0, peers of all types behave indifferently, while at the other extreme with α equal to 1, peers of type n request only files of type n.

Clearly, the values $p_n(m,k)$ define valid probability distributions because the sum $\sum_{m,k} p_n(m,k)$ equals 1 for every $n = 1, \ldots, N$:

$$\begin{aligned} \sum_{m,k} p_n(m,k) &= \sum_k p_n(n,k) + \sum_{m \neq n,k} p_n(m,k) \\ &= \sum_k \left[(1-\alpha) + \frac{\alpha}{p(n)}\right] p(n,k) + \sum_{m \neq n,k} (1-\alpha) p(m,k) \\ &= \frac{\alpha}{p(n)} \sum_k p(n,k) + (1-\alpha) \sum_{m,k} p(m,k) = 1. \end{aligned}$$

In the above formulas, we use (1) and the fact that $p(m,k)$ is a probability distribution, which implies that $\sum_{m,k} p(m,k)$ equals 1.

A very important property of our model of the semantic structure is that the frequency of queries to files generated by all the peers in the system follows the distribution $p(m,k)$. To prove this fact, we compute the probability that a randomly and uniformly chosen peer selects the kth file of type m:

$$\begin{aligned} \sum_n \frac{u_n}{U} p_n(m,k) &= \sum_n p(n) \cdot p_n(m,k) \\ &= p(m) \left[(1-\alpha) + \frac{\alpha}{p(m)}\right] p(m,k) + \sum_{n \neq m} p(n)(1-\alpha) p(m,k) \\ &= \alpha p(m,k) + (1-\alpha) p(m,k) \sum_n p(n) = p(m,k). \end{aligned}$$

In the above reasoning, we make use of the fact that $p(n)$ is a valid probability distribution and so $\sum_n p(n)$ equals 1.

4.2.2 Synthetic Model

The synthetic model of the semantic structure that we use in our experiments was previously introduced in [22]. This model assumes that the numbers of files of each semantic type is the same, and that the distribution of the file

popularity within one type, the file popularities without type partitioning, and the numbers of peers of each type follow Zipf's law. We note that most related studies have assumed a Zipf distribution (e.g., [15], [28], [36], [48]), with the notable exception of an evaluation of Kazaa [24] that tends to indicate that content popularity follows a different type of distribution.

The number of files and peers of each semantic type, and the request characteristics in the synthetic model are given by the following formulas:

$$u_n = \frac{U}{n \cdot H_N}, \tag{4}$$

$$d_n = \frac{D}{n \cdot H_N}, \tag{5}$$

$$p(m) = \frac{1}{m \cdot H_N}, \tag{6}$$

$$p(m,k) = \frac{1}{m \cdot H_N} \cdot \frac{1}{k \cdot H_{d_m}}, \tag{7}$$

$$p_n(m,k) = \begin{cases} \frac{1}{k \cdot H_{d_m}} \cdot \frac{1-\alpha}{m} \cdot \frac{1}{Z}, & m \neq n, \\ \frac{1}{k \cdot H_{d_n}} \cdot \left(\alpha + \frac{1-\alpha}{n}\right) \cdot \frac{1}{Z}, & m = n, \end{cases} \tag{8}$$

where $H_i = \sum_{j=1}^{i} 1/j$ is the ith harmonic number and Z is a normalizing constant chosen such that $\sum_{m,k} p_n(m,k)$ equals 1 for $n = 1, \ldots, N$. It can be shown that Z equals $(1-\alpha) \cdot H_N + \alpha$, independent of n.

For the sake of comparison with the trace-based data sets, we generate two synthetic data sets which we shall further call `suprnova_syn` and `piratebay_syn`. The numbers of files and semantic types in the synthetic data sets are the same as in the corresponding trace-based data sets (see Table 1). Figs. 4c and 4d show the request frequencies per file in the `suprnova_syn` and `piratebay_syn` data sets, respectively.

4.3 Optimal Caching Performance

Having the formal description of the model of the semantic structure, we can compute the *optimal caching performance* of SOSPNET, which is defined as the expected cache hit rate when the arrangement of items in the caches is optimal. It is generally not obvious how to define the *optimality* of a particular arrangement of cached items. Here, we describe the optimality in terms of performance and fairness by considering *Pareto optimality* [27]. An arrangement of items in caches is Pareto optimal if it is not possible to modify the contents of the cache of one peer in such a way that the fraction of requests produced by this peer that can be satisfied by the superpeers in its superpeer cache increases, while for all other peers, this fraction does not decrease. Note that in our model, the optimality is determined by the arrangement of the items in the superpeer caches, as the contents of the file caches are fully determined by the arrangements of items in the superpeer caches. To support this claim, note first that the content of a file cache of a superpeer depends only on the set of weak peers that sends requests to this superpeer, and second that a weak peer contacts a specific superpeer only if a pointer to this superpeer is in its superpeer cache.

4.3.1 Existence of the Optimum

We will now prove that there always exists an arrangement of items in the superpeer caches that is optimal. Consider the file cache $s.F$ of superpeer s, and denote by $r_n(s)$ the fraction of all requests submitted to s that are issued by peers of type n. Assuming the Independent Reference Model [54] which generalizes the properties of demand-driven caching policies such as LFU, LRU, or mixed, the cache hit ratio of a peer of type n is nondecreasing function of $r_n(s)$—the more requests are produced by peers of type n, the better cache $s.F$ adapts to the needs of peers of this type.

We say that an arrangement of items in the superpeer caches of the weak peers is *structured* if the caches of all peers of the same semantic type are the same. Consider a nonoptimal structured arrangement of items in the superpeer caches. Nonoptimality of an arrangement implies that it is possible to modify a superpeer cache of one of the peers, say p, in such a way that two conditions hold. First, the fraction of requests produced by p that can be satisfied by the superpeers in p's superpeer cache increases. Second, for all other peers, this fraction does not decrease.

A cache modification for a peer p of type n influences the values of $r_n(s)$ of some of the file caches at the superpeers. Removing any superpeer s from p's cache decreases (or does not influence) $r_n(s)$ and increases (or does not influence) $r_m(s)$, where m is different from n. Subsequently adding any superpeer s to p's cache has the opposite effect—$r_n(s)$ increases (or does not change) and $r_m(s)$ decreases (or does not change). Furthermore, if for some l, $r_l(s)$ increased (decreased) after modifying p's cache, then it will also increase (decrease) when we perform the same modification to the cache in p', where p' is a peer of the same type as p. We explained above that the cache hit ratio of a peer of type l at superpeer s is a monotonic function of $r_l(s)$. We conclude that if a certain modification in a peer's superpeer cache improves the hit ratios at some peers and does not decrease the hit ratios at all other peers, then applying the same modification to the cache in another peer of the same type will improve the same hit ratios, and will not decrease the others.

We have just shown that if a certain modification of the superpeer cache of peer p improves the nonoptimal, structured arrangement of superpeer caches, then by applying the same modification to the superpeer caches of all peers of the same type as p, the arrangement can be improved by at least the same amount. Such a modification results in an arrangement that is again structured. There is, however, a finite number of (structured) arrangements, which means that we cannot endlessly improve. At some point, we will end up with a structured arrangement which is optimal. Note that in the optimal arrangement, the superpeer caches of all weak peers of the same semantic type are the same.

4.3.2 Upper Bound

After proving the existence of the optimal caching performance, we will provide an upper bound on its value. In an ideal situation, each weak peer p has its own "private" set of superpeers that caches pointers to files which are most likely

to be requested by p. Let's assume for simplicity that the sizes of all superpeer caches in the system are equal to σ, and that the sizes of all file caches are equal to ϕ. Consequently, the superpeers of p can index in total at most $\sigma \cdot \phi$ unique files. Now we only have to find the set of $\sigma \cdot \phi$ files, which is most likely to be requested by p. According to (3), the probability that the kth file of type m is requested by p is $p_n(m,k)$, where n is the semantic type of p. We sort all values $p_n(m,k)$ in a descending order: $p_n(m_1,k_1) \geq p_n(m_2,k_2) \geq \cdots \geq p_n(m_D,k_D)$, $(m_i,k_i) \neq (m_j,k_j)$ for $i \neq j$. The probability that a file request of p is satisfied by one of its superpeers, denoted by $ocp(n)$ (optimal caching performance of a peer of type n), can be expressed as

$$ocp(n) = \frac{\sum_{i=1}^{\sigma\cdot\phi} p_n(m_i,k_i)}{\sum_{i=1}^{D} p_n(m_i,k_i)} = \sum_{i=1}^{\sigma\cdot\phi} p_n(m_i,k_i). \quad (9)$$

The optimal caching performance of the whole system, ocp, is the weighted average of the values $ocp(n)$, with the weights equal to the fractions of peers of type n:

$$ocp = \sum_n \frac{u_n}{U} \cdot ocp(n) = \sum_n p(n) \cdot ocp(n). \quad (10)$$

The optimal caching performance in an ideal situation quantified in (10) provides an upper bound on the optimal caching performance value in any (nonideal) situation.

5 Performance Evaluation

For the purpose of performance evaluation, we have built a discrete time simulator of SOSPNET and some alternative system architectures that provide the reference points. The simulator uses the model of the semantic structure introduced in Section 4. We investigate a variety of performance aspects of the simulated systems including the cache hit ratios, the efficiency of the caching policies, the dynamics of peer joins and leaves, and the properties of the load balancing mechanisms.

We believe that the performance of SOSPNET, and in particular, its performance relative to the systems to which we compare SOSPNET, in a real environment will be close to its performance in our simulations for two reasons. First, SOSPNET operates at a level that is no way dependent on such elements as the actual network with its particular latency and connectivity characteristics in which it would run. Second, we use workloads that are based on the traces of real systems.

5.1 Experimental Setup

The simulated system consists of 100,000 peers and 1,000 superpeers. The selection of the number of peers relative to the number of superpeers is guided by what was learned from the study of Kazaa [34], the most popular superpeer network ever deployed, in which the peer-to-superpeer ratio is around 100. The sets of files and semantic types are obtained with the method described in Sections 4.2.1 and 4.2.2. The numbers of peers of a particular semantic type follow the distribution defined by (2). The value of the parameter α in (3) is set to 0.8. The size of the superpeer cache in any weak peer is 10 while the size of the file cache in any superpeer is 1,000. Before the simulation starts, all the superpeer caches have been filled with the identities of superpeers selected randomly and uniformly from the set of all superpeers. The file caches are initially empty. Each peer initially stores 10 files selected randomly, taking into account the file popularities and peer type. The superpeers are organized into a Gnutella-like network. If a superpeer cannot answer a file request, it employs request flooding in the superpeer network.

The simulation is executed in phases. In each phase, every weak peer requests one file. The target of the request is determined by the distributions $p_n(m,k)$. Although in SOSPNET searching and load balancing are integrated in one protocol, in the experiments, we evaluate these two mechanisms separately by disabling load balancing during all experiments but the last one.

5.2 Results

This section presents the results of the experimental evaluation of SOSPNET.

5.2.1 Caching Performance

In the first series of experiments, we compare the performance of searching in SOSPNET with three other systems that exploit a semantic structure in the P2P network.

The first of these systems, the *symmetric peer-to-peer network*, does not make use of superpeers. Similarly as in [55], each peer in this network maintains a cache of nodes that answered their requests in the past. The caching policy used here is the same as the policy of the superpeer caches in SOSPNET. The size of the peer caches is set to 40. The total number of items cached in the symmetric network is 4,000,000 (100,000 peers with caches of size 40 each), which is twice as high as the total number of items cached in SOSPNET, which is 2,000,000 (100,000 superpeer caches of size 10 each, and 1,000 file caches of size 1,000 each).

The second system used for comparison, the *fixed superpeer network*, exploits superpeers, but assumes that the set of superpeers assigned to a weak peer is fixed. Similar to SOSPNET, weak peers are initially assigned a list of 10 randomly and uniformly selected superpeers, but this list stays unmodified during the whole simulation. The fixed superpeer architecture is comparable with the Gnutella network with ultrapeers [49], with the addition that the responses to peer requests are cached. The size of the file cache at each superpeer is the same as in our self-organizing network, and equals 1,000.

Finally, the third reference system, the *superpeer network with two-level caching*, follows the approach described in [22], exploiting two-level caching of semantic information, without weak peers exchanging information about their superpeers.

Fig. 5 presents a comparison of the performance of SOSPNET and the three reference systems. The results are shown separately for all four data sets. For each execution phase, we present the fraction of search requests that are satisfied by one of the peer's direct neighbors (cache hit ratio). The direct neighbors in the symmetric system are the nodes stored in the peer's cache. In the superpeer architectures, the direct neighbors are the superpeers contained in the superpeer cache. To improve the clarity, we only plot every fifth point. Also, we organize the labels in the legends to reflect the order of the lines in the plots. The solid line represents the value of the optimal caching performance given by (10).

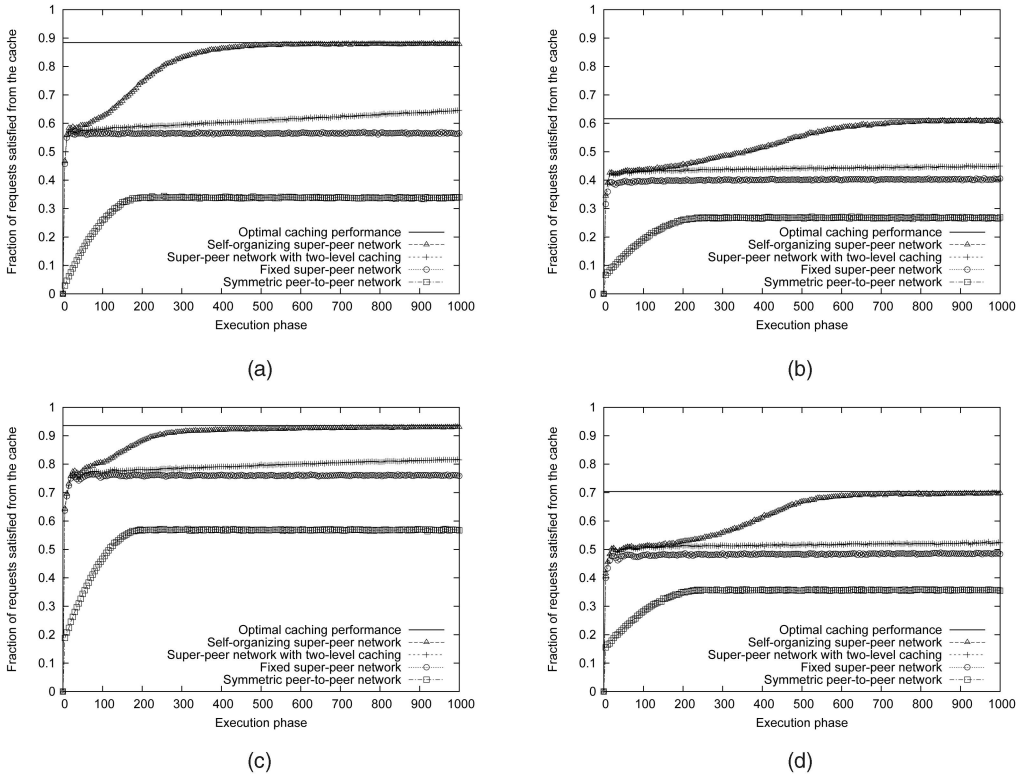


Fig. 5. Cache hit ratio for four types of P2P architectures exploiting the semantic relations between peers. (a) `suprnova.org`, (b) `piratebay.org`, (c) `suprnova_syn`, and (d) `piratebay_syn`.

All three superpeer architectures outperform the symmetric design. The cache hit ratios of the self-organizing, the two-level caching-based, and the fixed superpeer networks are very similar in the early execution phases. However, at some point, around phase 30, the performance of the fixed system stabilizes. This is the point where the items in the file caches are arranged optimally and further improvement could only be achieved by modifying the peer-to-superpeer assignments. In the two-level caching network, weak peers are not allowed to merge the contents of their superpeer caches, which has a significant performance impact. Among the evaluated systems, only SOSPNET reaches the theoretical performance upper bound.

An interesting observation regarding the optimal caching performance introduced in Section 4.3 can be made at this point. The value of *ocp* is defined with the simplifying assumption that peers of different types are using distinct sets of superpeers. Consequently, the value estimated by (10) may be higher than the actual achievable performance of an SOSPNET-like system, where superpeers are shared among peers of different semantic types. The experimental validation using a realistic system model shows, however, that this is not the case. The fact that the cache hit ratio of SOSPNET converges to *ocp* has two consequences. First, the formula in (10), which gives the performance upper bound, is also an accurate approximation of the actual value of the optimal caching performance. Second, the performance achieved by SOSPNET is close to optimal.

5.2.2 File Caching Policy

From here on, we study the performance of SOSPNET on the `suprnova.org` data set, leaving the alternative designs and data sets aside. In the following series of experiments, we investigate the properties of the novel mixed caching policy by comparing it to the traditional policies represented by LFU and LRU. The comparative results are obtained by running the simulation described in Section 5.2.1 for the SOSPNET architecture with the mixed file caching policy replaced first by LFU and then by LRU.

The convergence of the cache hit rates in the course of the simulation for the three caching policies is presented in Fig. 6a. In agreement with the expectations discussed in Section 3.3, LRU converges faster than the other two policies. Although the cache hit ratio converges to the highest value for the mixed policy, the improvement over LRU and LFU is not significant.

The real advantage of the mixed policy is visible only after presenting the cache hit rates for individual files. Fig. 6b decomposes the average cache hit ratio in phase 1,000 into the cache hit rates of requests targeting single files. The files in this figure are ranked based on their popularity. It is clearly visible that the mixed policy, compared with LFU and LRU, leads to higher cache hit rates for less popular files. This property of the mixed policy decreases the dependency of the system on the presence of a strong semantic bias in content popularity and reduces the variance of the caching performance across files. The mixed policy has thus a positive impact on the system performance in case of searches for files with heavy-tailed popularity distributions.

5.2.3 Peer Joins and Leaves

The number of execution phases needed for a system to achieve its peak performance does not say much about the bootstrapping period of an individual peer. In the next experiment, we measure how long it takes for a peer joining

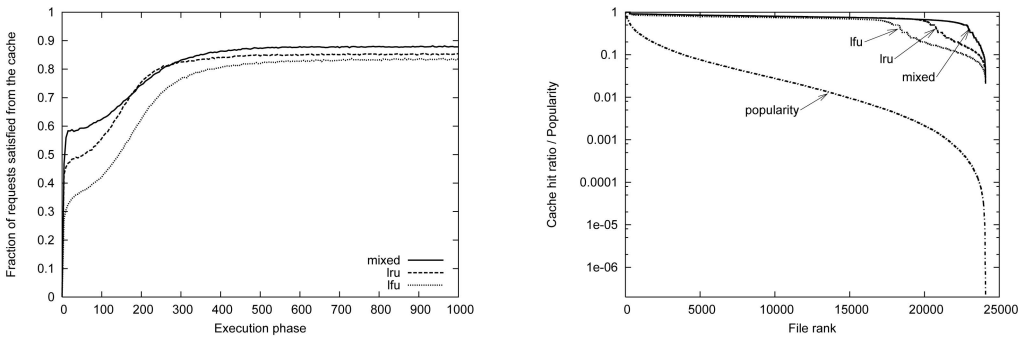


Fig. 6. The impact of the file caching policy on (a) the convergence speed and (b) the cache hit ratio for individual files.

the system to find its optimal set of neighbors. We assume that the new peer joins the network when the system has reached its maximum performance, i.e., in phase 1,000 of the experiment described in Section 5.2.1.

Fig. 7a shows how fast the cache hit ratio of the new peer increases with the phase number of this peer for the investigated architectures. For better readability, we plot every fifth data point. The number of iterations (search queries sent) required by a peer to reach its top performance is under 50 for the superpeer networks and around 100 for the symmetric network. It is also worth noting that the caching performance of a peer in a superpeer network is generally more stable compared to the performance of a peer in the symmetric network.

We expect that the self-organizing superpeer network as well as the other types of systems we evaluate is resilient to node failures (unexpected leaves) because of the redundancy built into the architecture. A peer that looses some of its neighbors can still connect to the network using other nodes in its cache. We simulate a catastrophic system failure by killing simultaneously half of peers in the symmetric network, or half of the weak peers and half of the superpeers in case of the superpeer networks, both selected randomly and uniformly. As a consequence, on average, 50 percent of the pointers stored in the semantic caches become invalid. We measure how long it takes before the system replaces the broken references with valid ones.

Fig. 7b shows how the performance of the system is affected by the failure of 50 percent of the nodes in execution phase 500. In spite of the scale of the failure, the cache hit ratio of the superpeer networks does not decrease much and goes back to the previous level in less than 50 phases. The performance degradation of the symmetric network is more visible but it does not prevent the system from recovering, which takes roughly 80 phases.

5.2.4 Clustering of Peers and Content

In Section 3.1, we claimed that the mutual dependency established between the superpeer and the file caches results in semantically related peers and files being clustered together. Here, we validate the correctness of this claim. We first investigate the correlation between the superpeer caches of weak peers of the same semantic type.

Fig. 8a presents the value of the *peer clustering coefficient* defined for each semantic type. The semantic types are sorted according to the decreasing values of the clustering coefficients. The peer clustering coefficient is the average number of identical items in the superpeer caches of peers of one semantic type divided by the size of the superpeer cache. The statistics are collected in execution phase 1,000. For the sake of comparison, we also present the peer clustering coefficient computed at the beginning of the simulation for randomly initialized superpeer caches.

A high value of the peer clustering coefficient indicates that the locality property of the weak peer requests is indeed exploited by the architecture of SOSPNET. For more than 90 percent of the types, peers of the same type have, on average, at least 3 (out of 10) identical items in their

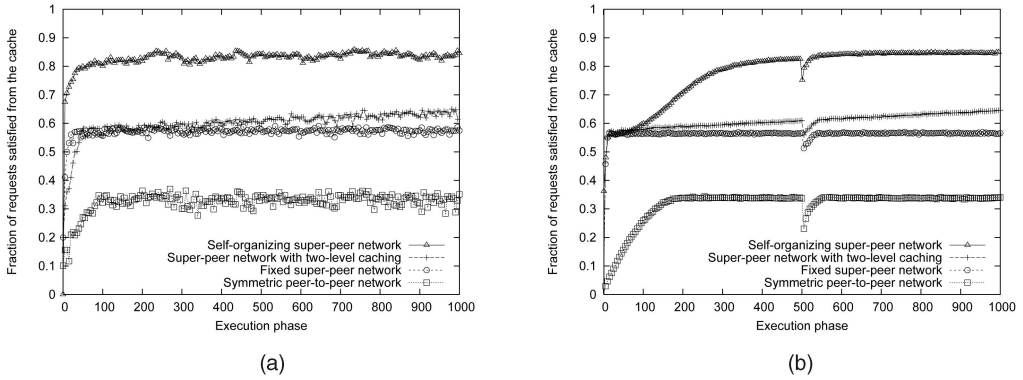
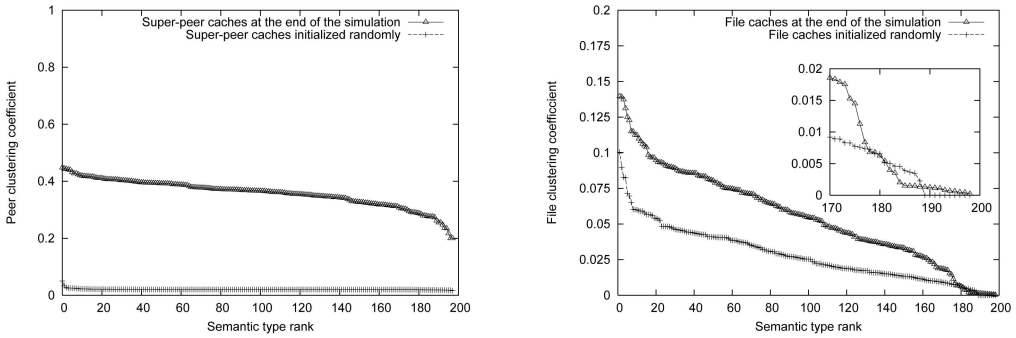


Fig. 7. (a) The caching performance of a weak peer joining the system and (b) the drop in performance when half of the peers simultaneously fail in phase 500.



(a) (b)

Fig. 8. The correlation (a) between the superpeer caches of the weak peers of the same semantic type and (b) between the items in the file caches of the superpeers.

superpeer caches, which is remarkable given that there are 1,000 superpeers in the system.

In the following experiment, we evaluate the correlation between the semantic types of files in the file caches. We define the *file clustering coefficient* of a semantic type as the average of the Jaccard coefficients of pairs of files of this type. The Jaccard coefficient [1] is a commonly accepted measure of similarity between sample data. The Jaccard coefficient $J(f_1, f_2)$ of the pair of files (f_1, f_2) can be expressed as

$$J(f_1, f_2) = \frac{|Q(f_1) \cap Q(f_2)|}{|Q(f_1)| + |Q(f_2)|}, \quad (11)$$

where $Q(f_i)$ is the set of superpeers that has a pointer to f_i in its file cache. In other words, the Jaccard coefficient of two files is the ratio of the number of co-occurrences of both files and the total number of individual occurrences of these files in the file caches. The values of the file clustering coefficients observed in our simulations in execution phase 1,000 are compared in Fig. 8b with the file clustering coefficients when the file caches are initialized with pointers to files selected randomly with a bias toward more popular files determined by the file request model. Also in this figure, the semantic types are sorted according to the decreasing values of the clustering coefficients. The use of the biased instead of the uniform random distribution during the file cache initialization makes the comparison more representative for an environment, where the numbers of file occurrences in the system are proportional to their popularity.

The lower level of file clustering in SOSPNET compared to peer clustering is expected because the targets of peer requests are not limited to files of one semantic type. Even if all peers of one semantic type use the same superpeers, the file caches of those superpeers will still maintain pointers to files of different types.

5.2.5 Load Balancing

In the last experiment, we activate the load balancing functionality of SOSPNET to assess its ability to deal with superpeers with heterogeneous capacities.

Every superpeer is randomly and uniformly assigned one of the four capacity groups. The capacity values and the number of superpeers in each group are presented in Table 2. All superpeers start with the same value of the accepted load equal to 1, resulting in superpeers accepting all requests. The parameter β controlling the speed of convergence of the exponential smoothing employed to correct the accepted loads of the superpeers (see Section 3.6) is set to 0.9. We let the simulation run for 1,000 phases before measuring the average effective load, the standard deviation of the effective load, and the cache hit ratio of the superpeers in the different capacity groups.

Ideally, the effective loads of all superpeers in the system should be the same. The results of the experiment, which are presented in Table 2, indicate that the load balancing mechanism of SOSPNET is able to distribute the system load among the superpeers according to their capacities. Furthermore, the low value of the standard deviation indicates that there are no significant differences in the amounts of load assigned to superpeers with the same capacities. Finally, the last column of Table 2 shows that the load balancing mechanism does not significantly affect the search performance by retaining cache hit rates, which are close to those observed when load balancing is disabled (see Section 5.2.1).

6 Related Work

The concept of leveraging the heterogeneity of peers by exploiting high-capacity nodes as superpeers has proved to

TABLE 2
The Performance of the Load Balancing Algorithm of SOSPNET

Super-peer capacity	Number of super-peers	Effective load		Cache hit ratio
		Average	Standard deviation	
0.25	242	172.516	30.212	0.843
0.5	252	163.754	20.864	0.835
0.75	243	148.226	19.185	0.847
1	263	159.374	21.549	0.851

have great potential [58] and has resulted in implementations in popular P2P networks. KaZaa [4] and Morpheus [5], which are both based on the FastTrack [3] protocol, are widely used file sharing systems that make use of superpeers. Although FastTrack is a proprietary technology with no detailed documentation, it is known that FastTrack peers are automatically elected to become superpeers if they have sufficient bandwidth and processing power (users can disable this feature). A central bootstrapping server provides new peers with a list of one or more superpeers to which they can connect. Superpeers index the files shared by client peers connected to them and proxy search requests on behalf of these peers. All queries are therefore initially directed to the superpeers.

An extension of the basic Gnutella [46] system has an architecture based on *ultrapeers* [49], which are conceptually equivalent to superpeers. Any new peer with enough bandwidth and CPU power immediately becomes an ultrapeer and establishes connections with other ultrapeers, forming an overlay network. A new ultrapeer is required to establish a predefined minimum number of connections to client peers. Neither FastTrack nor Gnutella ultrapeers rearrange the assignments between peers and superpeers and so both networks are instances of the fixed superpeer network architecture that we show to have performance inferior to that of SOSPNET in Section 5.

The Edutella [35], [40] network proposes a superpeer architecture based on the concept of semantic clusters. It creates a logical layer on top of the base P2P network topology by grouping peers with similar content interests. The clustering is performed by matching the semantic information provided by the peers to clusters, with each cluster being maintained by a superpeer. In addition to controlling the internal structure of the cluster, superpeers are responsible for routing messages between peers from different clusters. In contrast to SOSPNET where peers are clustered based on patterns automatically discovered in their requests, in the Edutella network, the policies defining the peer clustering rules have to be defined manually by domain experts.

Superpeer architectures have also been proposed for structured P2P networks [23], [38]. Such architectures group nearby peers based on some criteria, such as network latency or adjacency in the key space, and organize the communication between groups using a superpeer layer. To find a peer that is responsible for a key, the top layer overlay network routes among the superpeers to determine the group responsible for the key, which, in turn, uses an intragroup overlay network to find the target peer. The lookup time in structured superpeer networks depends on the size of the state maintained by each superpeer and on the total number of superpeers. Some architectures [38] are even able to guarantee a constant-time lookup. The rigid organization of content items based on their identifiers in structured P2P networks hampers optimization efforts that exploit the semantic properties of the content.

Exploiting the semantic properties of peers and content has also attracted a significant interest of the research community. Various approaches to capturing the semantic proximity between peers have been proposed. Some of them rely on a predefined ontology (semantic classification) [18]. Unfortunately, defining ontologies is often a manual, time-consuming process and the resulting semantic classes have to be continuously adjusted to reflect changes in semantic profiles of the content. Another approach is based on adding semantic shortcuts (i.e., extra links) between peers that share some interest [25], [50]. These links are created dynamically based on the set of most recent downloads, for instance. Such a mechanism is very reactive to evolving download patterns. Nevertheless, the nonintrusive nature of this approach does not allow to exploit available information such as the overlap between caches, which has also been used to approximate the semantic proximity between peers [56]. A refined proximity measure takes into account not only the content of peers' caches but also their generosity and the popularity of shared files [11]. None of the approaches discussed here combines a dynamic discovery of semantic proximity between peers and files with a multilevel P2P architecture as SOSPNET does.

7 Conclusions

We have introduced a self-organizing superpeer network architecture called SOSPNET built on top of an unstructured topology with semantic correlations between peers and files. Starting with random sets of neighbors, peers are always able to find superpeers that guarantee the highest performance of their searches. All decisions in our system are made locally by each peer based on the information collected during previous searches. We have also proposed a novel performance model of a P2P network, where peer requests exhibit semantic patterns. Through simulations with real-world trace-based data, we have shown that in SOSPNET, not only very popular files but also less popular content can be located very efficiently. Furthermore, we have demonstrated that a new peer that joins the system can very quickly find a set of superpeers that guarantees the highest performance. Finally, we have shown that our system is robust to catastrophic failures and it supports superpeers with heterogeneous capacities by controlling the amounts of load delegated to individual superpeers.

References

[1] http://en.wikipedia.org/wiki/jaccard_similarity_coefficent, 2009.
[2] http://gnutella.wego.com, 2007.
[3] http://www.fasttrack.nu, 2009.
[4] http://www.kazaa.com, 2009.
[5] http://www.musiccity.com, 2009.
[6] http://www.napster.com, 2007.
[7] http://www.suprnova.org, 2009.
[8] http://www.thepiratebay.org, 2009.
[9] Y. Breitbart, R. Vingralek, and G. Weikum, "Load Control in Scalable Distributed File Structures," *Distributed and Parallel Databases*, vol. 4, no. 4, pp. 319-354, Oct. 1996.
[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. INFOCOM*, pp. 126-134, Mar. 1999.
[11] Y. Busnel and A.-M. Kermarrec, "Proxsem: Interest-Based Proximity Measure to Improve Search Efficiency in p2p Systems," *Proc. European Conf. Universal Multiservice Networks (ECUMN '07)*, Feb. 2007.
[12] C. Chatfield, A.B. Koehler, J.K. Ord, and R.D. Snyder, "A New Look at Models for Exponential Smoothing," *J. Royal Statistical Soc., Series D, The Statistician*, vol. 50, pp. 147-159, 2001.
[13] Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker, "Making Gnutella Like p2p Systems Scalable," *Proc. SIGCOMM*, Aug. 2003.

[14] Y. Chen, Z. Xu, and C. Zhai, "A Scalable Semantic Indexing Framework for Peer-to-Peer Information Retrieval," *Proc. Workshop Heterogeneous and Distributed Information Retrieval,* Aug. 2005.
[15] V. Cholvi, P. Felber, and E. Biersack, "Efficient Search in Unstructured Peer-to-Peer Networks," *Proc. Symp. Parallelism in Algorithms and Architectures (SPAA '04),* June 2004.
[16] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Lecture Notes in Computer Science,* pp. 46-66, Springer, 2001.
[17] B. Cohen, "Incentives Build Robustness in Bittorrent," *Proc. First Workshop Economics of Peer-to-Peer Systems,* May 2003.
[18] A. Crespo and H. Garcia-Molina, "Semantic Overlay Networks for p2p Systems," technical report, Stanford Univ., Sept. 2002.
[19] T. Decker, R. Luling, and S. Tschoke, "A Distributed Load Balancing Algorithm for Heterogeneous Parallel Computing System," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications,* Nov. 2000.
[20] F. Le Fessant, S. Handurukande, A.-M. Kermarrec, and L. Massoulie, "Clustering in Peer-to-Peer File Sharing Workloads," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '04),* Feb. 2004.
[21] A.J. Ganesh, A.-M. Kermarrec, and L. Massouli, "Peer-to-Peer Membership Management for Gossip-Based Protocols," *IEEE Trans. Computers,* vol. 52, no. 2, pp. 139-149, Feb. 2003.
[22] P. Garbacki, D.H.J. Epema, and M. van Steen, "Two-Level Semantic Caching Scheme for Super-Peer Networks," *Proc. IEEE 10th Int'l Workshop Web Content Caching and Distribution,* Sept. 2005.
[23] L. Garces-Erice, E.W. Biersack, K.W. Ross, P.A. Felber, and G. Urvoy-Keller, "Hierarchical Peer-to-Peer Systems," *Proc. ACM/IFIP Int'l Conf. Parallel and Distributed Computing (Euro-Par),* 2003.
[24] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," *Proc. Symp. Operating Systems Principles (SOSP '03),* Oct. 2003.
[25] S. Handurukande, A.-M. Kermarrec, F. Le Fessant, and L. Massoulie, "Exploiting Semantic Clustering in the Edonkey p2p Network," *Proc. 11th ACM SIGOPS European Workshop,* Sept. 2004.
[26] S.B. Handurukande, A.M. Kermarrec, F. Le Fessant, L. Massoulie, and S. Patarin, "Peer Sharing Behaviour in the Edonkey Network, and Implications for the Design of Server-Less File Sharing Systems," *Proc. EuroSys Conf.,* Apr. 2006.
[27] M. Hutter, "Self-Optimizing and Pareto-Optimal Policies in General Environments Based on Bayes-Mixtures," *Proc. 15th Ann. Conf. Computational Learning Theory (COLT '02),* 2002.
[28] A. Iamnitchi, M. Ripeanu, and I. Foster, "Small-World File-Sharing Communities," *Proc. IEEE INFOCOM,* Mar. 2004.
[29] A. Iosup, P. Garbacki, J. Pouwelse, and D. Epema, "Correlating Topology and Path Characteristics of Overlay Networks and the Internet," *Proc. Global and Peer-to-Peer Computing Workshop (GP2PC '06) in Conjunction with the IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid '06),* May 2006.
[30] M. Jelasity, A. Montresor, and O. Babaoglu, "A Modular Paradigm for Building Self-Organizing Peer-to-Peer Applications," *Engineering Self-Organising Systems: Nature Inspired Approaches to Software Engineering,* G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli, eds., pp. 265-282, Springer-Verlag, Apr. 2004.
[31] D.R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," *Proc. 16th Ann. ACM Symp. Parallelism in Algorithms and Architectures (SPAA '04),* pp. 36-43, 2004.
[32] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta Algorithms for Hierarchical Web Caches," *Proc. IEEE Int'l Performance Computing and Comm. Conf. (IEEE IPCCC),* Apr. 2004.
[33] S. Le Blond, J.-L. Guillaume, and M. Latapy, "Clustering in p2p Exchanges and Consequences on Performances," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '05),* Feb. 2005.
[34] J. Liang, R. Kumar, and K.W. Ross, "The Kazaa Overlay: A Measurement Study," *Computer Networks,* special issue on overlays, vol. 49, no. 6, Oct. 2005.
[35] A. Loser, M. Wolpers, W. Siberski, and W. Nejdl, "Semantic Overlay Clusters within Super-Peer Networks," *Proc. Int'l Workshop Databases, Information Systems, and P2P Computing, Collocated with 29th Int'l Conf. Very Large Databases (VLDB '03),* Sept. 2003.
[36] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. Int'l Conf. Supercomputing (ICS '02),* June 2002.
[37] D.S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," Technical Report HPL-2002-57, HP Labs, Mar. 2002.
[38] A.T. Mizrak, Y. Cheng, V. Kumar, and S. Savage, "Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup," *Proc. Third IEEE Workshop Internet Applications,* June 2003.
[39] A. Montresor, "A Robust Protocol for Building Superpeer Overlay Topologies," *Proc. Fourth Int'l Conf. Peer-to-Peer Computing,* Aug. 2004.
[40] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, "Edutella: A p2p Networking Infrastructure Based on rdf," *Proc. 11th Int'l World Wide Web Conf. (WWW '02),* May 2002.
[41] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser, "Super-Peer-Based Routing Strategies for Rdf-Based Peer-to-Peer Networks," *Web Semantics,* vol. 1, no. 2, pp. 177-186, Feb. 2004.
[42] A. Parker, "P2p in 2005," http://www.cachelogic.com/research/, 2005.
[43] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips, "The Bittorrent p2p File-Sharing System: Measurements and Analysis," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '05),* Feb. 2005.
[44] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured p2p Systems," *Proc. IEEE INFOCOM,* Mar. 2004.
[45] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," *Proc. SIGCOMM,* 2001.
[46] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," *Proc. First Int'l Conf. Peer-to-Peer Computing (P2P '01),* Aug. 2001.
[47] J. Risson and T. Moors, "Survey of Research Towards Robust Peer-to-Peer Networks: Search Methods," *Computer Networks,* vol. 50, no. 17, pp. 3485-3521, 2006.
[48] M.T. Schlosser, T.E. Condie, and S.D. Kamvar, "Simulating a File-Sharing p2p Network," *Proc. First Workshop Semantics in P2P and Grid Computing,* May 2003.
[49] A. Singla and C. Rohrs, "Ultrapeers: Another Step Towards Gnutella Scalability," http://www.limewire.com/developer/Ultrapeers.html, 2001.
[50] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. INFOCOM,* Apr. 2003.
[51] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. SIGCOMM,* pp. 149-160, 2001.
[52] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks," *Proc. SIGCOMM,* Aug. 2003.
[53] C. Tempich, S. Staab, and A. Wranik, "Remindin': Semantic Query Routing in Peer-to-Peer Networks Based on Social Metaphors," *Proc. 13th Int'l World Wide Web Conf.,* May 2004.
[54] S. Vanichpun and A.M. Makowski, "The Output of a Cache under the Independent Reference Model—Where Did the Locality of Reference Go?" *Proc. SIGMETRICS 2004/PERFORMANCE 2004: Joint Int'l Conf. Measurement and Modeling of Computer Systems,* pp. 295-306, 2004.
[55] S. Voulgaris, A.-M. Kermarrec, L. Massoulie, and M. van Steen, "Exploiting Semantic Proximity in Peer-to-Peer Content Searching," *Proc. Int'l Workshop Future Trends of Distributed Computing Systems (FTDCS '04),* May 2004.
[56] S. Voulgaris and M. van Steen, "Epidemic-Style Management of Semantic Overlays for Content-Based Searching," *Proc. EuroPar 2005,* Aug. 2005.
[57] Z. Xu and Y. Hu, "Sbarc: A Supernode Based Peer-to-Peer File Sharing System," *Proc. Eighth IEEE Int'l Symp. Computers and Comm.,* June/July 2003.
[58] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," *Proc. IEEE Int'l Conf. Data Eng.,* Mar. 2003.
[59] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE J. Selected Areas Comm.,* vol. 22, no. 1, pp. 41-53, Jan. 2004.
[60] Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for dht-Based p2p Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 4, pp. 349-361, Apr. 2005.

Paweł **Garbacki** received the double MSc degree in computer science from Vrije Universiteit Amsterdam and Warsaw University in 2003 and the PhD degree in distributed systems from Delft University of Technology in 2008. During the Summer of 2005 and 2006, he was a visiting scientist at the IBM T.J. Watson Research Center, Yorktown Heights, New York. A year later, he pursued an internship at Google in Zurich, Switzerland. In 2008, he returned to Google to take a full-time position. His research interests include design, implementation, and performance analysis of large-scale distributed systems with emphasis on efficient data transfer protocols and high-volume data processing.

Dick H.J. Epema received the MSc and PhD degrees in mathematics from Leiden University, the Netherlands, in 1979 and 1983, respectively. From 1983 to 1984, he was in the Computer Science Department of Leiden University. Since 1984, he has been in the Department of Computer Science of Delft University of Technology, where he is currently an associate professor in the Parallel and Distributed Systems Group. During the academic year 1987-1988, the Fall of 1991, and the Summer of 1998, he was a visiting scientist at the IBM T.J. Watson Research Center, Yorktown Heights, New York. In the Fall of 1992, he was a visiting professor at the Catholic University of Leuven, Belgium. His research interests are in the areas of performance analysis, distributed systems, peer-to-peer systems, and grids.

Maarten van Steen is a full professor in the Computer Systems Group at the Vrije Universiteit Amsterdam. He teaches modules and courses covering distributed systems, computer networks, operating systems, and complex networks to academics and professionals. He has coauthored two textbooks on networked computer systems. His research concentrates on large-scale distributed systems with a strong emphasis on adaptive techniques that support automatic replication, management, and organization of wired and wireless systems. Recently, he has been exploring gossip-based solutions to achieve decentralized autonomous systems, partly focusing on very large wireless sensor networks and pervasive computing. He is furthermore a consultant for Philips Research, and closely participates with a collaboration of high tech SMEs for developing and deploying real-world pervasive computing systems.