[DLHSt+05] *Vasilios Darlagiannis, Nicolas Liebau, Oliver Heckmann, Andreas Mauthe, Ralf Steinmetz;* **Caching Indices for Efficient Lookup in Structured Overlay Networks;** Fourth International Workshop on Agents and Peer-to-Peer Computing. Springer, Juli 2005, S.

Caching Indices for Efficient Lookup in Structured Overlay Networks

Vasilios Darlagiannis[†], Nicolas Liebau[†], Oliver Heckmann[†], Andreas Mauthe[‡], and Ralf Steinmetz[†]

[†] Multimedia Kommunikation (KOM), Darmstadt University of Technology,
Merckstr. 25, 64293 Darmstadt, Germany,
`[bdarla, liebau, heckmann, steinmetz]@kom.tu-darmstadt.de`
[‡] Lancaster University, Computing Department, Lancaster, LA1 4YR, UK
`andreas@comp.lancs.ac.uk`

Abstract Structured overlay networks for Peer-to-Peer systems (e.g. based on Distributed Hash Tables) use proactive mechanisms to provide efficient indexing functionality for advertised resources. The majority of their occurrences in proposed systems (e.g. Chord, Pastry) provide upper bounds (logarithmic complexity with respect to the size of the graph representing the network) on the communication cost in worst case scenarios and their performance is superior compared to unstructured alternatives. However, in particular (empirically observed) scenarios where the popularity of the advertised resources follows a distribution considerably different from the uniform distribution, structured P2P networks may perform inferiorly compared to well designed unstructured P2P networks that exploit effectively the resource popularity distribution. In order to address this issue, a very simple caching mechanism is suggested in this paper that preserves the theoretical superiority of structured overlay networks regardless of the popularity of the advertised resources. Moreover, the churn effect observed in Peer-to-Peer systems is considered. The proposed mechanism is evaluated using simulation experiments.

1 Introduction

Structured overlay networks for Peer-to-Peer (P2P) systems, e.g. Chord [27], Pastry [24], Tapestry [29] and Omicron [7], use proactive mechanisms to provide efficient indexing functionality for advertised resources. The majority of their implementations provide theoretical upper bounds on the communication cost in worst case scenarios, assuming that the maintenance of the topology heals the divergence (caused by the dynamic participation of the peers) from the "ideal" network structure. Modeling the topology of a P2P network with a graph, the maximum distance between any two nodes is equal to the *diameter* of the graph. In graphs representing networks such as Chord (each node maintains $O(log(N))$ [illegible] where N [illegible]
$D_{CH} = O(log(N))$. The number of nodes [illegible]
the peers, e.g. in the case of Pastry or Chord or equal to the [illegible]

constructed clusters of peers, e.g. in the case of the two-tier architecture of Omicron ($D_O = O(log(N/l))$, where l is the average population of each cluster). However, a more useful metric to evaluate the communication cost for routing messages in structured overlay networks is the *average inter-peer distance*. The average cost for graphs such as the one representing Chord is $\mu_{D_{CH}} = D_{CH}/2$ [27]. On the other hand, the average inter-peer distance for networks such as Omicron based on de Bruijn graphs [5] is $\mu_{D_O} \simeq D_O - (k-1)^{-1}$, where k is the degree of the nodes [13]. However, since the graph nodes in Omicron represent clusters of peers, the actual average inter-peer distance is smaller than the average inter-peer distance in Chord.

Structured overlay networks have been designed mainly to overcome the intrinsic scalability issue of *flat* and *unstructured* networks, such as Gnutella v0.4 [20]. However, for several reasons, structured overlay networks have not been utilized in widely-deployed P2P systems (with the exception of the Kademlia network [18]). Instead, system designers opt for *hierarchical* or *hybrid* approaches where a subset of peers (usually termed as *super-peers*, or *ultra-peers*) is responsible for indexing and finding the advertised resources. Moreover, a number of mechanisms have been suggested to improve the performance of unstructured networks, e.g. expanding rings or multiple random walks [16]. The success of these mechanisms is based on the assumption of uneven popularity of the available resources. In fact, this assumption is validated by a number of empirical observations of file sharing systems (cf. [26], [9] and [4]) where the popularity of the resources is reported. While there is a disagreement on the exact distribution that describes the popularity of the resources (Zipf, lognormal, etc.), it can be safely concluded that it is not uniform.

Therefore, an interesting debate has arisen lately on whether structured overlay networks can perform efficiently if non-uniform popularity of resources is observed [15]. Apparently, structured networks perform equally well in any lookup request, thus, providing upper bounds, though not exploiting effectively the query frequency. Some hybrid approaches have been suggested to address this issue, such as hybrid PIER [14] or OceanStore [22]. Though, in these hybrid approaches the formation of two separate overlay networks is suggested, a structured one and an unstructured one to deal with unpopular and popular queries, respectively. The shortcomings and weaknesses of these solutions are mainly (i) the increased complexity, (ii) the additional maintenance cost that is out-of-band, (iii) the lack of adaptability to both uniform and non-uniform distributions and (iv) the increased delay when the initial overlay network selection for searching the resource fails and the fall-back alternative must be followed.

The aforementioned concerns are taken into account in the solution investigated in this paper. A simple though efficient mechanism is suggested that capitalizes on the adequateness of caching resources following non-uniform distributions and the higher interest of the P2P users to a relatively small subset of the available resources. It extends the capabilities of structured overlay networks without any additional maintenance effort and very low additional routing cost compared to the original algorithms of structured networks in worst case scenarios where the cache is not properly updated. No extension of their signalling protocols is required, thus, avoiding increasing further the complexity of their operation[1]. Merely, we invest on existing information collected through the normal network operation to improve the routing performance. The observed churn rate of the P2P networks, which is the most critical factor (together with the popularity distribution) is considered in our simulation experiments. While caching methods have been proposed for unstructured or hybrid overlay networks (cf. [17], [12]), they lack investigation on the structured counterparts. Moreover, several caching mechanisms have been extensively used for increasing the performance of Web technologies [1].

The rest of the paper is organized as follows. In Section 2 the proposed mechanism and its advantages, together with the related algorithms are presented. Afterwards, the simulation experiments are described in Section 3, followed by the related work in Section 4. The paper is concluded in Section 5.

2 Index caching mechanism

2.1 Basics

In the common design approaches of structured overlay networks, e.g. based on Distributed Hash Tables (DHTs) [2], queries are forwarded via intermediate peers towards the destination peer that is responsible for the part of the DHT which includes the globally unique identifier (GUID) characterizing the query. It is only the destination peer(s) that has the required information to reply to the query. Such design is suitable for evenly popular items since there is non-ambiguous mapping of the resources to the system and the workload is evenly distributed. Thus, in such designs it is necessary to follow the whole path before it is possible to match the query.

The common core functionality provided by the majority of structured overlay networks could be described by the following basic operations:

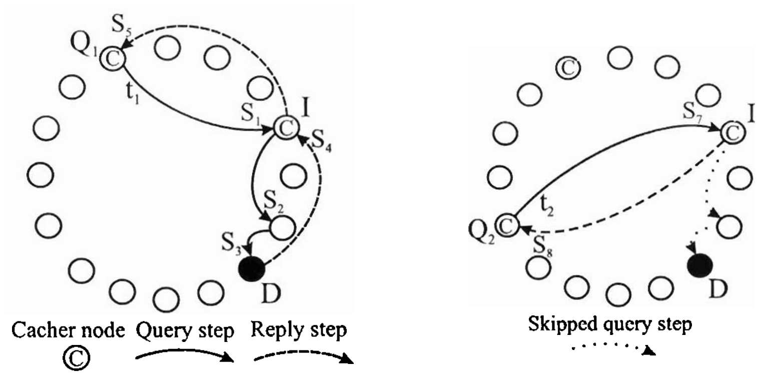
1. The *Routing* operation that requires the construction of a routing table for selecting the most "promising" neighbor to forward the queries.
2. The *Indexing* operation that constructs and updates the necessary distributed data structures for replying to queries.
3. The *Maintenance* operation that maintains the ideal network topology so that the theoretical upper bounds for the communication cost in worst case scenarios can be met.

Chord, Pastry and Tapestry are examples of structured networks that offer the aforementioned functionality. However, Omicron [7] suggests an additional function, that of *caching* to offer more efficient services, though it is proposed as an optional functionality for systems dealing with non-uniformly popular resources. The exploitation of the adequate design of the caching mechanism for [illegible]

[1] [illegible] structured networks [illegible] than their counterparts.

2.2 Mechanism design

The rationale behind the caching mechanism is described as follows. Since peers participate both in generating queries and routing them towards the destination, it may be advantageous to reuse the information gained from the replies they received from locally generated queries. Thus, peers may provide directly the position of the requested resource instead of forwarding the query until it reaches the final DHT destination. Moreover, if peers monitor the popularity of forwarded requests, they could additionally consider caching the most popular of them provided that they hold the necessary indexing information. A simple mechanism to develop such indexing knowledge is to modify the semantics of the routing procedure. For popular requests, intermediate peers may consider storing locally the incoming queries and generate identical ones (though originated at the intermediate peer) and forward them instead of the original queries. The received replies can be used both to reply the stored pending queries and to populate the local cache with useful and popular information. However, the gathered information may be used for a maximum amount of time t_{Th} that depends on the peer uptime distribution [8]. In fact, t_{Th} defines the maximum time a cache entry can be used, thereby, providing a simple mechanism deal with the high churn rate. Expired entries are removed from the cache after the t_{Th} time.



(a) Lookup steps for peer Q_1. (b) Lookup steps for peer Q_2.

Figure 1. Lookup operation using cache indices.

The proposed scheme is illustrated in Figure 1 using a Chord-like structured network. There, at time t_1 peer Q_1 queries for a resource indexed at peer D (Figure 1(a)). Assume that peer I considers that the specific query is popular. Then, instead of forwarding the query, peer I generates an identical query that eventually arrives at peer D. Peer D replies to peer I, which both updates the local cache and provides the reply to peer Q_1. Apparently, peer Q_1 may also update its local cache if it considers the query popular. After[illegible] at time t_2, with $t_1 < t_2 < t_1 + t_{Th}$ (where t_{Th} is the threshold time indicating that the cache content is valid with high probability), peer Q_2 queries for the same item and peer I is in the path towards peer D. In that case peer I provides the cached information to peer Q_2 immediately skipping the rest of the lookup steps towards D (Figure 1(b)). Furthermore, peer Q_2 may update its local cache if it considers the query popular. However, in the latter case it is important to consider the "aging" of the information as it is not directly provided by peer "D", but from a cached index. Peer Q_2 has to set the lifetime of the entry in the cache to $T'_{Th} = T_{Th} - (t_2 - t_1)$.

Two important factors drive the design mechanisms of caching. First, the scalability of the solution can be only provided if the size of the information that is additionally requested by each peer is *constant*. However, this constraint is not necessarily a practical limitation since this mechanism is designed to operate in systems where a small portion of the resources is frequently requested. Thus, each peer can locally decide which resources are popular by simply using a counter and the elapsed time since the first accounted appearance to estimate the rate of querying them and maintain the c most popular resources.

The second critical factor that has to be considered is the high churn rate of the peers. Nonetheless, conditional reliability mechanisms [8] may reduce the side-effects. Naturally, popular resources are being held by several peers. Assuming that the responsible DHT nodes can provide back either the complete set of these peers or an adequate subset of them, the intermediate peers have sufficient information for locating a reliable peer that is still alive.

Key	Expiration	Frequency	Window reset	Indices	Pending queries	Marked	
...	...	...	...	...	...	...	c

Figure 2. Abstract description of the cache structure.

The proposed cache structure is illustrated in Figure 2. Each row contains information for a single advertised resource. The first field includes the *key* of the resource. The second field contains the *Expiration* timer set to the maximum lifetime of the cache entry. As it has already been mentioned, to set the value of the expiration time the "age" of the index has to be considered. This mechanism assumes that also the indexing mechanism uses an expiration timer to remove old advertisements[2]. The third field is the *Frequency* field, which is a local counter that indicates how many times a query for that item has arrived on the particular intermediate peer. The value of the counter is reset periodically and the *Window reset* field stores that time. The fifth field includes the list of collected *Indices* about peers that posses the requested resources and may be directly contacted. The subsequent field contains the list of the *Pending queries* for this resource. Finally, the *Marked* field indicates that the cache replacement algorithm has selected this entry to be removed from the cache. However, the list of pending

[2] [illegible] A [25] is an example [illegible] system [illegible] tion th
to remove old advertisements [illegible] of the advertisements [illegible]
re-advertised their services and resources.

queries for this resource is not empty and the deletion of the selected entry has to be delayed until the reply will be received and the pending queries replied.

Further, an additional characteristic that may be successfully exploited to increase the efficiency of the structured networks is the fact that peers are also owners of resources. In cases where the requested resource is being hold locally on the intermediate peer it can be safely provided to the requestor. It may be additionally argued that instead of developing the index caching mechanism, intermediate peers can provide the requested resources themselves. Nevertheless, this possibility is application depended and many factors (e.g. copyrights, technical limitations, system design) have to be considered. Moreover, if further constraints apply (e.g. find a resource or service provider in the closest vicinity to the requestor) this solution may not provide optimal performance.

2.3 Algorithms

Several cache replacement policies have been developed to fit to the requirements of different problems (cf. least frequently used (LFU) [23], least recently used (LRU) and LRU-K [19]). In fact, the replacement policy adopted for the indices cache on each peer is a variation of the LFU algorithm, which is further enhanced with timeouts on the maximum lifetime of each entry. The latter improvement is mandatory for capturing the dynamics of P2P overlay networks. The pseudo-code of the LFU variation is provided in Algorithm 2.1. If there is an entry with 0 popularity and no pending queries, then this entry is removed. Otherwise, the least popular entry is returned[3].

Algorithm 2.1: LFU_REPLACEMENT(*cache*, *pendingQueries*)

```
found = cache.get(1)
for i ← 2 to cache.size()
  do
    queryList = pendingQueries.remove(i)
    if (cache.get(i).popularity == 0 and queryList.isEmpty())
      then
        cache.remove(i)
        return (null)
    else if (found.popularity > cache.get(i).popularity and
        ( not cache.get(i).isMarked()))
      then
        found = cache.get(i)
return (found)
```

The pseudo-code for filling a cache entry with information obtained from a reply is listed in Algorithm 2.2. Upon the reception of the reply all the pending queries are further replied. Moreover, if the cache entry is not marked, it is filled with the received lookup information.

3 The popularity of an entry on a particular peer is calculated [illegible] queries traversing this peer over the last time window.

```
Algorithm 2.2: FILLCACHEENTRY(cache, entry, pendingQueries)

queryList = pendingQueries.remove(entry.ID)
for i ← 1 to queryList.size()
    do { lookupMsg = queryList.remove(1)
         lookupMsg.setDestination(lookupMsg.initiator)
         lookupMsg.setSender(localGUID)
         lookupMsg.setValue(entry.value)
         replyMessage(lookupMsg)
if (entry.isMarked())
  then {cache.remove(entry)
  else
    entry.setValue(srcs)
```

```
Algorithm 2.3: GETCACHEENTRY(cache, id, pendingQueries, msg)

entry = cache.get(id)
if (entry == null)
  then { entry = createNewCacheEntry(id, null)
         cache.put(id, entry)
         return (entry)
if (entry.hasExpired() and ( not entry.isMarked()))
  then { cache.remove(id)
         entry = createNewCacheEntry(id, null)
         cache.put(id, entry)
  else
    entry.updateUsage()
if (entry.frequency > FREQUENCY_THRESHOLD and
    entry.getValue == null)
  then { queryList = pendingQueries.get(id)
         if (queryList.isEmpty())
           then { lookupMsg = createLookupMessage(id)
                  forwardMessage(lookupMsg)
         queryList.add(msg)
if (cache.size() − marked >= MAX_CACHE_SIZE)
  then { removed = LFU_Replacement(cache)
         queryList = pendingQueries.remove(removed.ID)
         if (queryList.isEmpty)
           then {cache.remove(removed)
           else
            removed.mark()
return (entry)
```

Finally [illegible]

is set based on the [illegible] uptime) the entry is [illegible] new one is created, otherwise the frequency field is updated. Further, if the frequency of the query is higher than a threshold then, the message is stored as a pending query and a new lookup message is being created for the queried GUID, if this is the first pending message[4]. Moreover, if the size of the cache has exceeded its maximum value, the least frequently used entry is either removed if no pending queries are present or is marked for deletion at the arrival of the reply.

3 Evaluation

3.1 Experiments description

The goal of the simulation experiments is to evaluate the performance improvement of the Chord network using the proposed indices cache mechanism and compare to the original network.

The simulation experiments have been performed using a general purpose discrete event simulator for P2P overlay networks [6]. The population of the peers is consisted of 4096 peers distributed randomly over a Chord ring with key range of 65536. Peers and resources share the same key range. Each experiment lasts approximately 30 minutes of simulation time. Peers randomly select a resource to query every 20 seconds (asynchronously from each other). The process is repeated for 80 times resulting to a total number of approximately 327000 queries.

Peers start requesting the resources after a certain stabilization period. The probability distribution of the resource selection follows a lognormal distribution with parameters $\mu = 0.82$ and $\sigma = 2.9$ following the guidelines in [4]. The selection of the lognormal distribution over the Zipfian distribution is motivated by the greater challenge of the former since the popularity of the resources is more widely distributed. The implemented lognormal generator produces randomly selected GUIDs limited to the aforementioned key range. On average, approximately $4000 - 4100$ different keys are generated on each run.

Figure 3 displays a representative cumulative distribution of the resource popularity, where the resources are sorted from the most to the least popular. From this figure, it can be concluded the first 25 most popular resources contribute to approximately 80% of the query load. Thus, an equivalently small cache size is adequate to store them and achieve high performance, provided that the popularity identification algorithm operates correctly. Nevertheless, in real experiments, the cache size may have to be bigger to capture effectively the popular resources since the key range may be considerably larger.

3.2 Results

In this section the measurement observations of the simulative experiments are reported. Figure 4 displays the reduced routing communication cost in terms

[illegible]ould be noted that [illegible]ry [illegible]tains no indexing [illegible]on to indicate the status of the [illegible] routing mechanism.

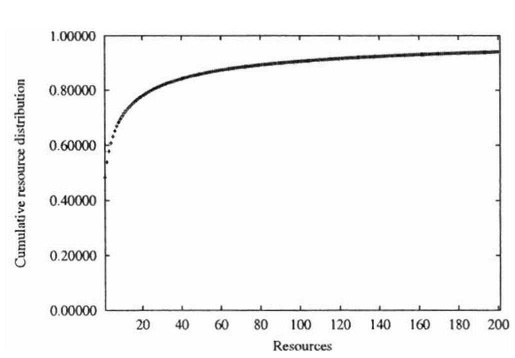


Figure 3. Cumulative resource popularity distribution.

of the original Chord network, as a function of the expiration timeout. Two different experiment have been selected:

1. Experiment A, where the *FREQUENCY_THRESHOLD* is 5, the maximum cache size is set to 80 and the frequency counter is reset every 200 seconds.
2. Experiment B, where the *FREQUENCY_THRESHOLD* is 3, the maximum cache size is set to 300 and the frequency counter is reset every 100 seconds.

We can observe that the total communication load for query routing can be considerably reduced using the caching mechanism down to 50% of the original load.

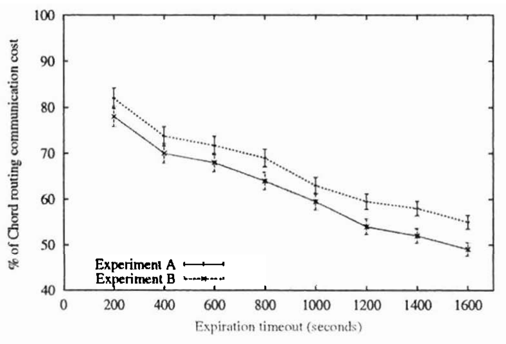
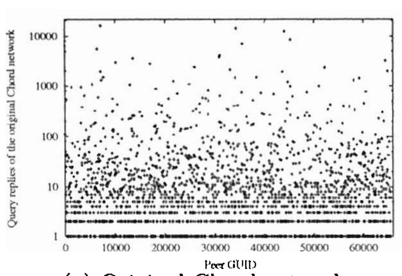
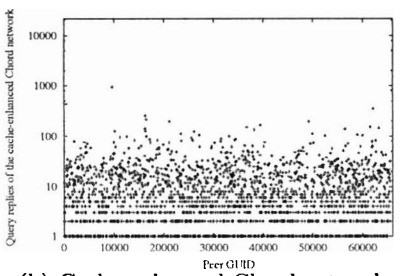


Figure 4. Cache routing load as a percentage of the original Chord network

Moreover, peers responsible for popular resources may become "hot spots" and potential bottlenecks of the system. By utilizing the cache mechanism the load for replying to the queries is getting more evenly distributed. Figure 5(a) displays the load balance in the original Chord network, while Figure 5(b) shows the query replying in the cache-enhanced Chord network. It should be noted that the vertical axis is logarithmically scaled. Moreover, many peers reply with cached values which are not considered in this figure.



(a) Original Chord network.



(b) Cache-enhanced Chord network.

Figure 5. Load distribution for replying queries.

4 Related work

OceanStore [11] is a P2P storage system built on top of Tapestry [29] to take advantage of its scalable lookup capabilities. However, OceanStore, employs an additional probabilistic mechanism based on attenuated Bloom Filters [3], resulting to a hybrid solution for improving Tapestry's routing performance when the popularity of the queries is not uniform [22]. In the context of the OceanStore algorithm, the first Bloom filter (located at position '0') is a record of the objects contained locally on the current node. The *i*th Bloom filter is the union of all of the Bloom filters for all of the nodes a distance *i* through any path from the current node. An attenuated Bloom filter is stored for each directed edge in the network. A query is routed along the edge whose filter indicates the presence of the object at the smallest distance. When the fast probabilistic algorithm fails to provide the requested results, OceanStore activates the Tapestry routing mechanism to forward the request to the final destination. However, the routing cost is increased when Bloom Filters provide false replies. Moreover, the maintenance of two different overlay networks increases considerably the operational cost of the system (both overlays are based on proactive mechanisms).

Hybrid PIER [14] is an overlay network designed to improve the performance of PIER [10] when looking up for popular resources. It is composed of two components. (i) an UltraPeer-based Gnutella network[5] and (ii). a structured Content Addressable Network (CAN) [21] where only UltraPeers participate. [illegible]

identify and p[illegible] [illegible] us into the [illegible] [illegible]

[5] Based on Gnutella v0.6 protocol.

The search algorithm uses flooding techniques for locating popular items, and structured (DHT) search techniques for locating rare items.

Caching mechanisms have been also utilized in P2P storage systems such as PAST [25], which is deployed on top of Pastry, a structured overlay network. The goals of the caching mechanism in PAST are (i) to minimize client access latencies, (ii) to maximize the query throughput and (iii) to balance the query load in the system. However, the utilized caching management system deals with the stored content and not with the indexing mechanism, which is the focus of this paper.

In addition, the use of caching has been investigated for the case of unstructured P2P overlay networks. Markatos [17] exploits network locality in unstructured networks (i.e. Gnutella) using caching mechanisms. Peers cache received replies and provide them to other peers sending similar queries instead of further forwarding the queries. Therefore, the overall traffic is reduced. Similarly, Liu et al. [12] investigate the reduced traffic and response time when caching the results, using simulation based experiments. Boykin et al. [4] study the statistical properties of queries in Gnutella-like systems and provide analytical results on query cache performance.

5 Conclusions

While caching has been extensively used in Web technologies and in unstructured P2P overlay networks, it has not received sufficient attention for structured P2P network approaches. The adequacy of caching popular indices in intermediate peers along the paths towards the responsible indexing peer(s) for structured networks is demonstrated in this paper.

The proposed caching mechanism reduces significantly the routing cost in structured P2P networks. Compared to alternative proposals, the achieved performance improvement is combined with a set of attractive features. Since the mechanism is locally applied to peers it can be incrementally deployed. Moreover, there is no need to introduce multiple specialized overlay networks operating in parallel or additional protocols to update the cached information.

Though this work identifies the critical parameters that have to be considered for the caching problem, there are several issues that can be further developed. Selecting the optimal values for the critical parameters can improve even further the observed performance. Moreover, a mechanism to adapt the values of the parameters to the dynamics of the network has significant practical and theoretical interest. Finally, different cache operation algorithms may provide better results in certain scenarios. The problem requires further analytical investigation to understand better its dynamics.

Acknowledgements

This work has been performed partially in the context of the project Premium "Preis- und Erlsmodelle im Internet - Umsetzung und Marktchancen" where TU

Darmstadt has been funded by the German Bundesministerium fuer Bildung und Forschung (BMBF).

References

1. Martin Arlitt, Rich Friedrich, and Tai Jin. Performance evaluation of Web proxy cache replacement policies. *Performance Evaluation*, 39(1-4):149–164, 2000.
2. Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up Data in P2P Systems. *Communications of the ACM*, 46(2):43–48, 2003.
3. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
4. P. Oscar Boykin, Jesse S.A. Bridgewater, and Vwani Roychowdhury. Statistical Properties of Query Strings. Preprint, January 2004.
5. N.G. De Bruijn. A combinatorial problem. In *Proceedings of the Koninklije Nederlandse Academie van Wetenshapen*, pages 758–764, 1946.
6. Vasilios Darlagiannis, Andreas Mauthe, Nicolas Liebau, and Ralf Steinmetz. An Adaptable, Role-based Simulator for P2P Networks. In *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods*, pages 52–59, June 2004.
7. Vasilios Darlagiannis, Andreas Mauthe, and Ralf Steinmetz. Overlay Design Mechanisms for Heterogeneous, Large Scale, Dynamic P2P Systems. *Journal of Networks and System Management*, 12(3):371–395, 2004.
8. Vasilios Darlagiannis, Andreas Mauthe, and Ralf Steinmetz. Optimizing Overlay Network Stability using Burn-In Methods. Submitted for publication, March 2005.
9. Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, Modeling, and Analysis of Peer-to-Peer File Sharing Workload. In *Proceedings of 19th ACM Symposium on Operating Systems Principles*, October 2003.
10. Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. In *Proceedings of VLDB'03*, September 2003.
11. John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. OceanStore: an Architecture for Global-scale Persistent Storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201. ACM Press, 2000.
12. Yunhao Liu, Li Xiao, and Lionel M. Ni. Building a Scalable Bipartite P2P Overlay Network. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, April 2004.
13. Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience. In *Proceedings of ACM SIGCOMM'03*, pages 395–406, August 2003.
14. Boon Thau Loo, Ryan Huebsch, Ion Stoica, and Joseph M. Hellerstein. The Case for a Hybrid P2P Search Infrastructure. In *Proceedings of the 4th International*

[illegible] Crowcroft [illegible], and [illegible] Survey and Comparison of [illegible]-Peer Overlay Network Schemes. [illegible] *[illegible]munications Survey and* [illegible], March 2004.

16. Qin Lv, Sylvia Ratnasamy, and Scott Shenker. Can Heterogeneity Make Gnutella Scalable? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, March 2002.
17. Evangelos P. Markatos. Tracing a large-scale Peer-to-Peer System: an hour in the life of Gnutella. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and Grid*, pages 65–74, May 2002.
18. P. Maymounkov and D. Maziéres. Kademlia: A Peer-to-peer Information System Based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, 2002.
19. Elizabeth O'Neil, Patrick O'Neil, and Gerhard Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of data*, pages 297–306, 1993.
20. M. Portmann, P. Sookavatana, S. Ardon, and A. Seneviratne. The cost of peer discovery and searching in the Gnutella peer-to-peer file sharing protocol. In *Proceedings of the International Conference on Networks*, pages 263–268, 2001.
21. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable Content Addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172. ACM Press, 2001.
22. S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, June 2002.
23. John Robinson and Murthy Devarakonda. Data cache management using frequency based replacement. In *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 134–142, 1990.
24. Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
25. Antony I. T. Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.
26. Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, 2002.
27. Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE Transactions on Networking*, 11(1):17–32, February 2003.
28. Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf, May 2003.
29. Ben Y. Zhao, [illegible] [illegible] Tapestry: A [illegible] Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.