Considering Complex Search Techniques in DHTs under Churn

Jamie Furness, Mario Kolberg
Dept. of Computing Science and Mathematics, University of Stirling
{jrf,mko}@cs.stir.ac.uk

***Abstract*—Traditionally complex queries have been performed over unstructured P2P networks by means of flooding, which is inherently inefficient due to the large number of redundant messages generated. While Distributed Hash Tables (DHTs) can provide very efficient look-up operations, they traditionally do not provide any methods for complex queries. By exploiting the structure inherent in DHTs we can perform complex querying over structured P2P networks by means of efficiently broadcasting the search query. This allows every node in the network to process the query locally, and hence is as powerful and flexible as flooding in unstructured networks, but without the inefficiency of redundant messages.**

While there have been various approaches proposed for broadcasting search queries over DHTs, the focus has not been on validation under churn. Comparing blind search methods for DHTs though simulation we see that churn, in particular nodes leaving the network, has a large impact on query success rate. In this paper we present novel results comparing blind search over Chord and Pastry while under varying levels of churn. We further consider how different data replication strategies can be used to enhance the query success rate.

I. Introduction

The ability to perform complex queries is one of the most important features in many of the P2P networks actually deployed.

While structured P2P networks can provide very efficient look-up operations via a Distributed Hash Table (DHT), they traditionally do not provide any methods for complex queries. This can be attributed to the use of consistent hashing, which causes data to be distributed uniformly over the entire network; since consistent hashing does not preserve locality it is only possible to perform exact look-ups. This means in a simple DHT it is not possible to perform wild-card or full-text searching, limiting their application in the real world. Unstructured networks usually implement complex queries by a form of flooding or random walks, however flooding is inherently inefficient due to the large number of redundant messages [3], and random walks are slow with no guarantee of actually finding the data even if it exists. Making use of the structure inherent in DHTs we can perform complex querying over structured networks by means of efficiently broadcasting the search query. This allows every node in the network to process the query locally, removing the restrictions placed on the complexity of queries. We refer to this method of searching within a DHT as blind search.

Churn, the act of nodes arriving and departing from a network, is an important issue which should not be ignored when validating algorithms in P2P networks. In many types of P2P overlay it is not uncommon for routing tables to be slightly out-of-date due to churn; if an algorithm is only tested with fully up-to-date routing tables then there is no indication as to how it would actually perform in the real world. We note that there are existing methods for blind search in structured P2P networks [9], [14], [10], [15], [19], [20], [21], [22], [11], [6], but with very little, or in most cases no validation under churn. In this paper we aim to investigate the difference in performance of blind search over Chord and Pastry, when subjected to various levels of churn.

In sections II and III we discuss what is meant by the terms *complex query* and *blind search*. Related work is discussed in section IV. In section V we describe the general approach for blind search in Chord [17] and Pastry [16]. In sections VI and VII we explain the aim of our investigation and the parameters used. In section VIII we present novel simulation results validating and comparing these methods under varying levels of churn, and investigate how data replication strategies can enhance the query success rate. In section IX we draw conclusions from the results obtained, and speculate on how they may be used in the future to produce a method specifically aimed at high churn environments.

It is important to note that throughout the paper we refer to the message success rate as the percentage of messages successfully received during broadcasting. The query success rate refers to the percentage of search queries in which the requested data was found at least once. When the replication rate is one, in other words there is no redundancy, these should be similar, however with a higher replication rate the query success rate is expected to be higher.

II. What are Complex Queries?

Complex query is a vague term which is often used to describe any form of search query more complex than an exact match [7]. In this paper we refer to the ability to support all such types of queries as support for complex queries. In this section we discuss different types of queries which are often referred to as complex. For completeness we start with the basic exact match.

A. Exact Match

Due to the hash-table nature of a DHT, the only form of search query supported without some form of extension is the exact match. All DHTs support, either directly or indirectly,

a hash-table interface providing the methods $put(key, value)$ and $get(key)$. Given the key corresponding to a service, a DHT can guarantee the return of results, usually within a specific number of hops, if any such results exist. In many cases such limited support for search is not sufficient and clearly does not support complex queries, which has resulted in many proposed systems which build support for more complex queries on top of the exact match facility.

B. Keyword Search

If exact match look up is not sufficiently powerful, DHTs can be extended to support keyword-based queries. Keyword-based search provides the ability to associate multiple keywords with a single document.

C. Range Queries

Range queries are a type of query which look for any document, possibly indexed by keywords, which lies within a given range. For example, a user may wish to find available services within a certain price range.

D. Wild-card Search

The term wild-card search is used to describe a search in which part of the search term is unknown. Note that wild-card search is related to range queries, but not exactly the same. For example, assuming keywords were restricted to $a \ldots z$, the wild-card search for "acm*" can be converted to a range query for everything between "acm" and "acn". However other forms, such as "*acm", "a*m", or "*acm*" are difficult to map to a range query [12].

E. Full-text Search

Full text search refers to a technique for examining all words in all documents, to try match the search term supplied by the user. While this type of search may initially sound similar to the keyword search, keyword search solutions assume a limited number of keywords and do not scale well above a certain limit.

F. Semantic Search

Semantic search is a content-based search, in which queries are expressed in natural language instead of keywords. The goal in semantic search is to use semantics, the meaning of words, to return results which are relevant to the search terms, without simply matching keywords.

G. Regular Expressions

Although not often mentioned, in some circumstances it may be desirable to have the ability to search for data using regular expressions.

III. What is Blind Search?

The term blind search is used to describe a search operation in which no information about the search space is known, other than to distinguish the goal state from all others. In other words, as a query traverses through the network it has either reached the goal or not, there is no concept of distance to the goal as with regular operations in a DHT. Blind search can be thought of as the structured equivalent to flooding, providing all the flexibility of flooding, without the downside of redundant messages. Blind search can be used over structured P2P networks to provide support for complex queries. A survey of blind search techniques is provided by Furness and Kolberg [8].

IV. Related Work

While there appears to be no work on the comparison of blind search methods under churn, some of the approaches mentioned in section I have independently undergone minimal testing.

Using simulations with a pastry network of size 10,000 and average session times of $\{5, 15, 30, 60, 120, 600\}$ minutes, Castro et al. [4] show that the message success rate for broadcasting over Pastry starts to drop dramatically when the average session time drops below 30 minutes - however their paper focuses on comparing a modified version of Pastry known as HetroPastry against the unstructured overlay Gia, rather than against other structured networks.

Merz and Gorunova [15] suggest a broadcast solution which combines efficient broadcast with an epidemic communication algorithm. They compare the message success rate of an efficient broadcast algorithm over Chord against their combination of an efficient broadcast algorithm and epidemic communication algorithm, when under churn.

Recursive Partitioning Search (RPS) [22] over Pastry is simulated under churn, and compared against the effect of churn on flooding and random-walk. It is found that churn has a fairly low impact on flooding, though a high impact on random-walk. Data was replicated using a Zipf-like popularity distribution, with the replication ratio ranging from 10% to 0.01% of nodes. Due to the data replication, RPS is found to have limited dependence on churn, with a query success rate of 99% even if 15% of peers are temporarily out of the network.

V. Blind Search Methods

Both Chord and Pastry assume a circular identifier space of size N, in which each node maintain pointers to a list of successors and a routing table.

In Chord the routing table is a set of nodes, known as fingers, chosen at logarithmically increasing distance around the ring, the i^{th} entry in the table at node n contains the identity of the first node that succeeds n by at least 2^{i-1} ($i \geq 1$). In Pastry the routing table is set of rows, with all the entries at row r referring to nodes whose identifier shares the current node's identifier in the first r digits, but whose $(r+1)^{th}$ digit does not match that of the current node.

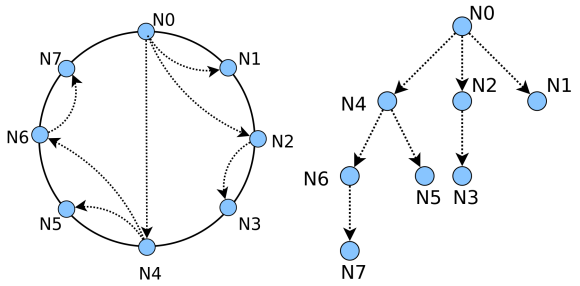

Figure 1. Dissemination of an example broadcast message using efficient broadcast in a Chord network, where N is a node and L is the limit parameter.

In the efficient broadcast algorithms nodes send the broadcast to each node in their routing table, giving it a limit parameter. In the case of Chord this limit is the identifier of the next finger, and is used to restrict the forwarding space of the receiving node to $(n, limit)$, as shown in figure 1. In Pastry the limit is the routing table row number, and restricts the forwarding space of the receiving node to rows greater than the limit. When a node receives a broadcast message it forwards it to all nodes in its routing table within the given forwarding space.

A detailed explanation of efficient broadcast for Chord is presented by El-Ansary et al. [6], who assumed the algorithm would perform similarly to the underlying DHT when under churn - as we explain in section VIII-A this is not the case. The efficient broadcast algorithm is presented by Castro et al. [5] for Pastry.

VI. Aims

In this paper we aim to compare the performance of the efficient broadcast search algorithms over Chord and Pastry when subjected to various levels of churn, and investigate the underlying causes of any differences found.

While it is suggested that broadcast over Pastry performs better than Chord [22], we wanted to determine why there is a difference and if there are extra costs associated with increased performance. We also feel that the simulations done by Vishnevsky et al. [22] in which data replication depended on the documents popularity is more representative of the data placement found in unstructured networks. We look at different types of replication strategies commonly implemented in structured overlays and compare their effect on performance under churn.

VII. Network Setup

Using the OverSim peer-to-peer network simulation framework [1] for OMNeT++, we ran two sets of simulations, with network sizes of 1,000 nodes and 10,000 nodes. To simulate churn the network was filled, thereafter nodes would join and leave based on times drawn from a Weibull distribution, as recommended by Stutzbach and Rejaie [18]. In the 1,000 node network the lifetime mean ranged from 100 seconds to 10,000 seconds. In the 10,000 node network the lifetime mean ranged from 100 seconds to 6,000 seconds.

Data was distributed randomly among the nodes, such that each node would have on average 2 data items, plus any replica. The replication rate was varied from 1 (no redundancy) to 32.

Once the network reached its target size we initiated 20 searches sequentially, each from a random node and for a random data item known to be currently within the network. If the data item queried was found at least once the search was considered a success.

In the Chord network we used parameters: $N = 8$, $S = 1$, and $D = \{120, 10\}$, where N is the number of successors, S is the stabilise delay, and D is the finger table maintenance delay in seconds.

In the Pastry network we used parameters: $L = 16$, $M = 0$, and $B = 4$, where L is the leaf set size, M is the neighbour set size, and B is the number of bits per digit.

VIII. Results

A. General Observations

To broadcast over a network efficiently requires building a broadcast tree. In both of the methods studied this is done on-demand using only information from the structure of the underlying DHT. This means that it is important for the DHT to be as up-to-date as possible to prevent a broken broadcast tree being constructed, causing less than perfect success rate. For example in figure 1, imagine node 4 had already left the network - none of its children would receive the broadcast and half of the network would go uncovered. We note that there are two types of node failure to consider:

1) When a node leaves the network the routing tables may become out-of-date for a period of time. If a node tries to send a message to a node which has already left the network, it will result in failure. A solution to this category of failures would be to implement an acknowledgement and timeout, allowing for message retransmission upon failure. Since the network should self-repair the message would be successfully delivered eventually. However while this would increase the success rate, it would also double the number of messages required, and increase search latency. Additionally when the nodes routing tables self-repair the new finger chosen may be in a partition that has already received the message, and retransmission would result in duplicates. While the amount of duplicates would probably be minimal and a small price to pay for higher message success rate, in the interest of a fair comparison we did not implement this addition.
2) If a node fails during the broadcast it may have received the message, but not yet forwarded it to all of it's children within the delegated section.

B. Algorithm Validation

1) Efficient Broadcast: As found by El-Ansary et al. [6], the efficient broadcast algorithm over Chord generated zero redundant messages and exactly $N-1$ messages were required to cover a network of size N. We found that in some cases

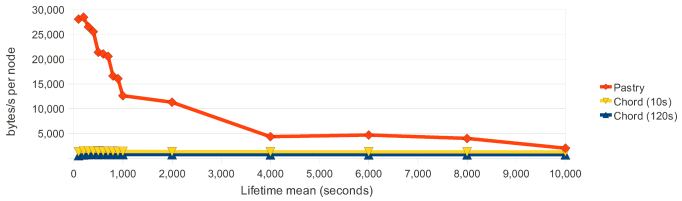

Figure 2. Bandwidth usage for broadcast in Chord and Pastry, 1,000 nodes.

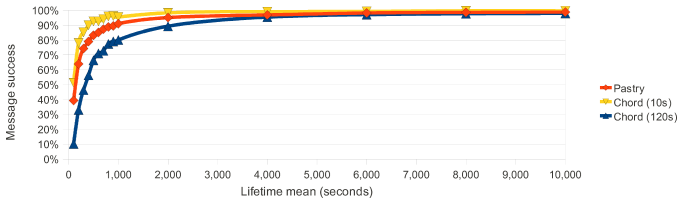

Figure 3. Average message success rate in Chord and Pastry, 1,000 nodes.

a node that joined after the broadcast had started managed to receive a message. This is because by the time the broadcast had propagated down the tree, the new node had been added to it's predecessors routing table. We felt this wasn't a problem, in a search context the more nodes reached the better.

2) Pastry's Broadcast Mechanism: Our simulations over Pastry also generated zero redundant messages, however did not always reach 100% of the nodes within the network. This is because it is possible for rows in the Pastry routing table to have gaps. According to the algorithm proposed by Castro et al. [5], upon finding a gap the message should be routed to the middle of the section missing from the routing table. However this addition adds complexity and results in redundant messages being generated. In the interest of a fair comparison we did not implement this addition.

C. Bandwidth Consumption

Comparing the Chord and Pastry overlays (figure 2) the most obvious difference is the number of messages sent, and hence bandwidth consumed. During our 1,000 node simulation the average number of sent messages/sec per node in the Chord networks with a maintenance delay of 120 seconds and 10 seconds was 9.3 and 19.5 respectively, using an average bandwidth consumption of 702 bytes/sec per node and 1.31 kilobytes/sec per node. In the Pastry network the average number of messages/sec per node was 181.6, using an average bandwidth consumption of 15.83 kilobytes/sec per node. In terms of bandwidth consumption this is between 12 and 23 times as much! However it is important to note that this is only an average, taken with lifetime mean ranging from 100 seconds to 10,000 seconds. At low churn rates the difference is much lower, for example with a lifetime mean of 10,000 seconds Chord was sending 8.6 messages/sec (664 bytes/sec) per node with a maintenance delay of 120 seconds, and 18.1 messages/sec (1.23 kilobytes/sec) per node with a maintenance delay of 10 seconds, and Pastry 22.1 messages/sec (1.95 kilobytes/sec) per node. While bandwidth consumption is still around 1.5 to 3 times higher with Pastry, this is nowhere near the differences seen at higher churn rates. The results from the 10,000 node simulation were similar.

As can be seen in figure 2 Chord uses an almost constant amount of bandwidth, the churn rate has no effect. This is to be expected due to Chords use of periodic maintenance to keep its routing table up-to-date. However as the churn level increases, the amount of bandwidth required by Pastry also increases rather dramatically. This is to be expected due to its use of reactive maintenance for the whole routing table. In many cases where either the churn rate is expected to be high or the bandwidth available is low, or both such as in mobile handsets, this high bandwidth requirement may disqualify Pastry immediately.

We should note that the bandwidth used here was the average from all simulation runs, including replica maintenance and query traffic.

D. Message Success Rate

When we start to look at the message success rate however, in figure 3 we see the benefit of keeping a more up-to-date routing table. In the 1,000 node network with Chord ($D = 120s$) we see an average message success rate of 71.3%, ranging from 10.2% (with a lifetime mean of 100 seconds) to 97.8% (lifetime mean of 10,000 seconds). When the maintenance delay in Chord is reduced to 10 seconds we see an average message success rate of 91.1%, ranging from 51.3% to 99.6%. With Pastry the average message success rate is 84.6%, ranging from 39.4% to 98.8%. While both networks converge towards a message success rate of 100% as lifetime mean increases, the interesting point is how they compare under high churn.

E. Search Delay

Comparing the average hop count on received search queries, we find that Pastry comes out ahead. In the 1,000 node networks Chord had an average hop count of 4.77, while Pastry had an average hop count of 2.66. In the 10,000 node network this increased to 6.12 for Chord and 3.40 for Pastry. This can be explained by Pastry's larger routing table, which results in a wider and shallower broadcast tree than that in Chord.

F. Data Replication

One easy way we can improve our query success rate is by optimising our data replication strategy.

1) Neighbour Replication: Neighbour replication is the term used to describe various different replication strategies, such as successor replication and leaf-set replication, in which data is replicated at neighbours of the original node, as can be seen in Figure 4. Neighbour replication is used by both Chord and Pastry, as when a node leaves the network the responsibility falls to its successor in Chord or nearest neighbour in Pastry, replica of any required data will already be in the right place.

However when considering search, neighbour replication is not a good choice. When we lose a branch of the broadcast tree we tend to lose multiple close together nodes, so ideally

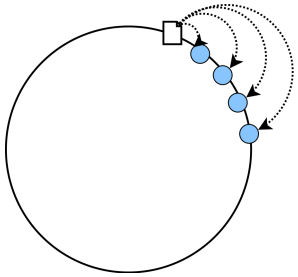

Figure 4. Placement of replica in the neighbour replication strategy.

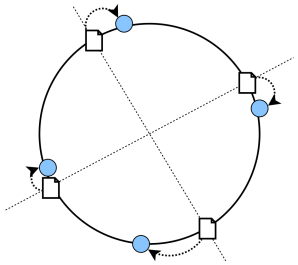

Figure 5. Placement of replica in the symmetric replication strategy.

we want replicas to be spread more evenly throughout the network, as is achieved using multi-publication replication.

A separate simulation ran with a 1,000 node Chord network showed that using 8 times replication only increased the average query success rate by 6%.

2) Multi-publication Replication: Multi-publication replication is different from neighbour replication in that the data is published at multiple keys within the network, each of which is of equal importance. There are three main strategies for implementing multi-publication replication:

- Multiple hash functions - To generate R unique keys R different hash functions are used. How replica are spread will depend on the hashing functions chosen.
- Correlated hashing - To generate R unique keys the numbers $r = 0 \ldots R$ are prepended to the original key before hashing. Assuming consistent hashing, replica should be spread evenly throughout the network.
- Symmetric replication - To generate R unique keys, first the hash h is calculated, then the keys are defined as $(h + (r * \frac{N}{R}))\%N$ where $r = 0 \ldots R$ and N is the size of the key space. Using this formula, replica will be spread perfectly evenly throughout the network. An example of replica placement when using symmetric replication can be seen in Figure 5.

Multi-publication replication has the advantage that replicas should be spread evenly throughout the network, assuming uniform hashing. It also has the advantage that any node can calculate the keys for all replica, allowing the issuing of parallel look-ups, or even use heuristics to query the "closest" replica. For our simulations we decided to implement symmetric replication, but assuming uniform hashing all 3 strategies should perform similarly.

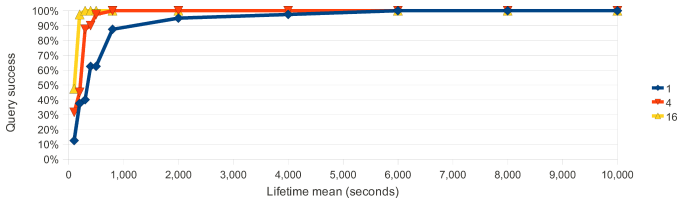

Figure 6. Effect of symmetric replication on query success rate in Chord ($D = 120s$), 1,000 nodes.

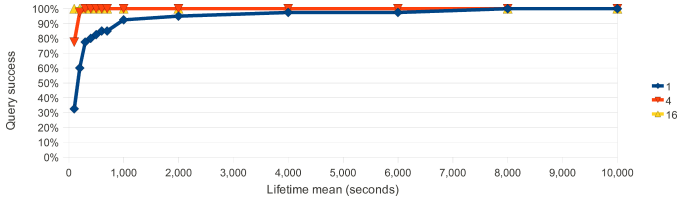

Figure 7. Effect of symmetric replication on query success rate in Chord ($D = 10s$), 1,000 nodes.

In figure 6 we can see how the query success rate increases the higher the replication rate. However in a 1,000 node Chord network with a maintenance delay of 120 seconds, even a replication rate of 16 gives a low query success rate of around 45% under high churn (100 seconds lifetime mean).

Comparing this with a 1,000 node Chord network with the maintenance delay is set to 10 seconds, in figure 7 we see that a lower maintenance delay does indeed improve query success rate. In fact now a replication rate of 16 is enough to give 100% query success rate, even under high churn (100 seconds lifetime mean).

Similarly, in figure 8 we can see that in a 1,000 node Pastry network it is possible to achieve 100% query success rate using around 16 replica, even under high churn (100 seconds lifetime mean).

Comparing the average query success rate over the different set ups we can see the importance of keeping routing tables as accurate as possible. We can also see that with enough replication in the network the effects of churn can be counteracted, however it is important to note that as the network size increases, the number of replica has to increase to compensate. We obviously need to remember that adding replicas is not free, requiring storage on the nodes as well as extra maintenance traffic. Ktari et al. found that multi-publication replication requires much more bandwidth during maintenance [13], and our simulations mirrored this.

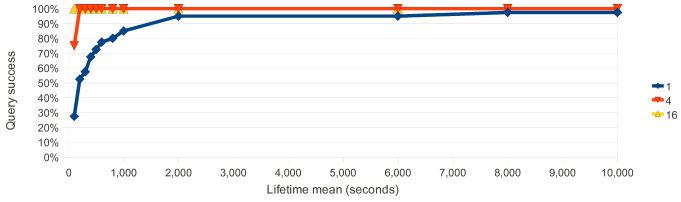

Figure 8. Effect of symmetric replication on query success rate in Pastry, 1,000 nodes.

IX. Conclusions

By simulating search via broadcast under churn over Chord and Pastry we clearly see the benefit of having the routing table kept as up-to-date as possible. When Chord's maintenance delay is turned appropriately for the level of churn in the network its message success rate can match, or even surpass, that of Pastry. When augmented with reasonable data replication we see it is possible to achieve a query success rate of nearly 100%, even under very high churn levels.

However it is obvious that the huge amounts of bandwidth required by Pastry will disqualify its use in many circumstances.

In section VIII-A we noted that simply using acknowledgement messages and resending if no acknowledgement is received would increase the message success rate, at the cost of a roughly twofold increase in messages sent. From the results in section VIII-C we can see that implementing this in Chord would probably be beneficial and still result in reasonable bandwidth usage. In future work it would be interesting to implement both this addition to Chord, and the addition to Pastry mentioned in section VIII-B2 and compare the results with the standard algorithms tested here.

In future work we aim to look at adapting the efficient broadcast algorithm for use for the variable-hop overlay Chameleon [2], which aims at keeping routing tables for high bandwidth nodes at 99% accuracy, using a more efficient maintenance algorithm.

We noted that having a larger routing table allows construction of a wider and shallower broadcast tree, resulting in lower hop counts and hence faster searches. In a Chameleon based overlay it would be possible to allow nodes to dynamically decide how many of their available fingers to forward the search to based on their available bandwidth, resulting in a heterogeneous network where high bandwidth availability equates to lower hop count and hence faster searches.

Chameleon does not specify the type of replication to be used. When using purely neighbour replication we see little, if any, benefit to blind search - however it is the default replication strategy used by most DHTs because it is advantageous during regular look-ups. On the other hand, using purely symmetric replication shows a large improvement during blind search, but does not have the advantages during regular look-ups that neighbour replication does; it also costs more in terms of complexity and bandwidth to restore failed replicas. We propose that using a combination of both strategies would provide the benefits from both approaches, and benefit both regular look-ups and blind searches.

References

[1] I. Baumgart, B. Heep, and S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *IEEE Global Internet Symposium, 2007*, pages 79 – 84. IEEE, 2007.

[2] A. Brown, M. Kolberg, and J. Buford. Chameleon: An adaptable 2-tier variable hop overlay. In *CCNC'09: Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, pages 770 – 775. IEEE Press, 2009.

[3] M. Castro, M. Costa, and A. Rowstron. Should we build gnutella on a structured overlay? *SIGCOMM Computer Communication Review*, 34(1):131 – 136, 2004.

[4] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 85 – 98. USENIX Association, 2005.

[5] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An Evaluation of Scalable Application-level Multicast Built Using Peer-to-Peer Overlays. In *INFOCOM 2003: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1510 – 1520. IEEE, 2003.

[6] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient Broadcast in Structured P2P Networks. pages 304 – 314. Springer-Verlag, 2003.

[7] J. Furness and M. Kolberg. Wide Area P2P Service Discovery Mechanisms using Complex Queries. *unpublished*.

[8] J. Furness and M. Kolberg. A Survey of Blind Search Techniques in Structured P2P Networks. In *PGNet 2010: Proceedings of The 11th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, pages 313 – 318, 2010.

[9] A. Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD thesis, The Royal Institute of Technology (KTH), 2006.

[10] A. Ghodsi, L. O. Alima, S. El-ansary, P. Brand, and S. Haridi. Self-Correcting Broadcast in Distributed Hash Tables. In *Proceedings of the 15th International Conference on Parallel and Distributed Computing and Systems*, 2003.

[11] K. Huang and D. Zhang. A Partition-Based Broadcast Algorithm over DHT for Large-Scale Computing Infrastructures. In *GPC '09: Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing*, pages 422 – 433. Springer-Verlag, 2009.

[12] Y.-J. Joung and L.-W. Yang. Wildcard Search in Structured Peer-to-Peer Networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1524 – 1540, 2007.

[13] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod. Performance Evaluation of Replication Strategies in DHTs Under Churn. In *MUM '07: Proceedings of the 6th International conference on Mobile and Ubiquitous Multimedia*, pages 90 – 97. ACM, 2007.

[14] W. Li, S. Chen, P. Zhou, X. Li, and Y. Li. An Efficient Broadcast Algorithm in Distributed Hash Table Under Churn. In *WiCom 2007: International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1929 – 1932, 2007.

[15] P. Merz and K. Gorunova. Efficient Broadcast in P2P Grids. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid*, pages 237 – 242, 2005.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware 2001: Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, 2001.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. pages 149 – 160. ACM, 2001.

[18] D. Stutzbach and R. Rejaie. Understanding Churn in Peer-to-Peer Networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet Measurement*, pages 189 – 202. ACM, 2006.

[19] D. Talia and P. Trunfio. Dynamic Querying in Structured Peer-to-Peer Networks. In *DSOM '08: Proceedings of the 19th IFIP/IEEE international workshop on Distributed Systems: Operations and Management*, pages 28 – 41. Springer-Verlag, 2008.

[20] V. Vishnevsky, A. Safonov, M. Yakimov, E. Shim, and A. D. Gelman. Scalable Blind Search and Broadcasting in Peer-to-Peer Networks. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 259 – 266. IEEE Computer Society, 2006.

[21] V. Vishnevsky, A. Safonov, M. Yakimov, E. Shim, and A. D. Gelman. Tag Routing for Efficient Blind Search in Peer-to-Peer Networks. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 409 – 416. IEEE Computer Society, 2006.

[22] V. Vishnevsky, A. Safonov, M. Yakimov, E. Shim, and A. D. Gelman. Scalable Blind Search and Broadcasting over Distributed Hash Tables. *Computer Communications*, 31(2):292 – 303, 2008.