

A Survey of Blind Search Techniques in Structured P2P Networks

Jamie Furness, Mario Kolberg

Dept. of Computing Science and Mathematics, University of Stirling

{jrf,mko}@cs.stir.ac.uk

Abstract—The ability to perform complex queries is one of the most important features in many of the P2P networks actually deployed today. While structured P2P networks can provide very efficient look-up operations via a Distributed Hash Table (DHT) interface, 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 there is no guarantee, in fact it is highly unlikely, that similar search terms will have their data stored together. This means in a simple DHT it is not possible to perform range queries, wild-card or full-text searching, which limits their application in the real world.

In this work we review the existing methods for performing complex queries on top of structured P2P networks; focusing on methods which allow for full-text search rather than only keyword queries. It should be obvious that to perform blind search with support for full-text queries the query must be processed locally at each node, and as such the problem of blind search is almost identical to the problem of efficiently broadcasting; with the difference that queries need not always reach all nodes to be successful. The majority of existing algorithms exploit the structure inherent in DHTs to efficiently broadcast the search query over the entire network.

I. INTRODUCTION

In the kind of P2P networks found today locating data without the use of complex queries is almost impossible. For example to locate a song in a specific format with the bit-rate within a certain range without the use of complex queries would require the user to guess the exact file naming scheme used and perform a new query for every possible bit-rate value within the desired range. Obviously this is not a sensible approach, and can be one reason why structured P2P networks are not as popular in the real world.

While structured P2P networks can provide very efficient look-up operations via a Distributed Hash Table (DHT) interface, 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 there is no guarantee, in fact it is highly unlikely, that similar search terms will have their data stored together. This means in a simple DHT it is not possible to perform range queries, wild-card or full-text searching, which limits their application in the real world. Unstructured networks usually implement wild-card search by a form of flooding or random walks, however flooding is inherently inefficient due to the large number of redundant messages sent [1], and random

walks tend to be slow with no guarantee of actually finding the data even if it exists.

It should be obvious that to perform blind search with support for full-text queries the query must be processed locally at each node, and as such the problem of blind search is almost identical to the problem of efficiently broadcasting; with the difference that queries need not always reach all nodes to be successful. The majority of existing algorithms for performing complex queries on top of structured P2P networks exploit the structure inherent in DHTs to efficiently broadcast the search query over the entire network; this allows every node in the network to process the query locally, removing the restrictions placed on the complexity of queries. We focus on methods which allow for full text search rather than only keyword based queries, and refer to this method of searching within a DHT as broadcast search. Some alternate methods [2], [3], [4], [5], [6] which do not require broadcasting can still perform wild-card and range queries, but are limited to keyword-based search rather than full-text search so we do not cover them here.

II. BROADCAST SEARCH TECHNIQUES

A. Efficient Broadcast

In [7] an algorithm for broadcasting complex queries, or indeed any data, over DHTs is proposed by El-ansary et al. using Chord [21] as an example. In Chord nodes form a ring structure, in a network of size N each node maintains routing state information for at most $\log(N)$ other nodes, namely the node succeeding it, known as the successor, and a set of finger nodes. The finger nodes are 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 the proposed algorithm, to initiate a query, a node n will send the query along with a limit to every (non-redundant) node in its finger table. The *limit* parameter given is the identifier of the next finger in the finger table, and is used to restrict the forwarding space of the receiving node to $]n, \text{limit}[$. The last node in the finger table is given a limit of the originating node. When the message is received by a node it forwards the broadcast to any fingers it has within the defined forwarding space, giving each one a new limit.

Using simulation El-ansary et al. show that there are no redundant messages generated and that exactly $N - 1$ messages are needed to cover every node. The network was

Proposal	Section	Reference	Contribution
Efficient Broadcast	II-A	[7]	Describes a method for efficiently broadcasting complex queries over a Chord network.
Self-correcting Broadcast	II-A1	[8], [9]	Extends the Efficient Broadcast algorithm to handle out-dated routing tables when broadcasting over DKS.
Pseudo-reliable Broadcast	II-A2	[9]	Describes a method for detecting and handling node failures during broadcasting.
Recursive Partitioning Search (RPS)	II-A3	[10], [11], [12]	Introduces a TTL value and other optimizations to try and decrease query traffic without affecting success rate.
A Partition-based Broadcast Algorithm	II-A4	[13]	Alters the Efficient Broadcast algorithm to perform binary search, resulting in a slower but more balanced search.
Dynamic Querying over DHT	II-A5	[14], [15], [16]	Uses an iteratively increasing TTL value to decrease query traffic without affecting success rate.
Efficient Broadcast in P2P Grids	II-A6	[17]	Proposes using a mixture of efficient broadcast and epidemic communication to achieve a better success rate under churn.
An Efficient Broadcast Algorithm	II-A7	[18]	Attempts to detect and handle failed nodes, but the implementation seems unfinished.
Pastry's broadcast mechanism	II-B	[19]	Describes a method for efficiently broadcasting over a Pastry network, which can be used for complex queries.
CAN broadcast	II-C	[20]	Describes a method for efficiently broadcasting over a CAN network, which can be used for complex queries.

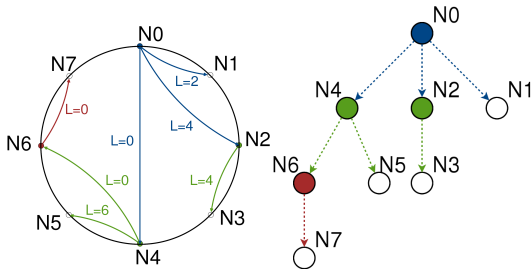


Figure 1. Dissemiation of an example broadcast message using Efficient broadcast in a Chord network, where N is a node and L is the limit.

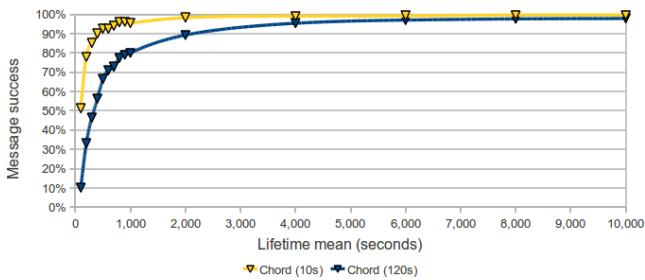


Figure 2. Effects of churn on Efficient broadcast success rate.

first populated with $2^{3...14}$ nodes and then the broadcast was initiated. During the simulations no nodes ever joined or left the network, so the effects of churn were not taken into account. Due to the tree structure used node failures have a large impact on the performance of efficient broadcast. For example in figure 1 if node $N4$ was to fail then half the network would not receive the message. To confirm this we ran simulations with a 10,000 node network and lifetime mean ranging from 100 seconds to 10,000 seconds; the maintenance delay of Chord was set to 10 seconds and 120 seconds. As can be seen in figure 2 the success rate drops dramatically when the lifetime mean drops below around 30 minutes.

Since the Efficient Broadcast algorithm was proposed it has been extended and improved in various different ways,

outlined below.

1) *Self-correcting Broadcast*: Extending on their own work, in [8] Ghodsi et al. claim the algorithm proposed will fail to cover all nodes when the routing tables are not up-to-date. A similar algorithm called Self-correcting Broadcast is proposed [9], which uses a correction-on-use technique over DKS [22].

Distributed K-ary Systems (DKS) are configured with parameter, $k \geq 2$, such that the look-up length is guaranteed to take at most $\log_k(N)$ hops for a network of size N . Each node maintains a routing table, consisting of $\log_k(N)$ levels. At each level l a node n has a view of the identifier space defined as $[n, n \oplus \frac{N}{k^{l-1}}[$. This means that at level one, the view consists of the whole identifier space, and at any other level, one k^{th} of the previous level is considered. At every level the view is partitioned into k equally-sized intervals. This method for partitioning the identifier space is known as the k-ary principle, and is described fully in [9].

In this algorithm a node starting the broadcast iterates through all levels of the routing table, starting at the first level. At each level, the node moves in counter-clockwise direction through all of its intervals, broadcasting a message to each responsible node r . Each broadcast message carries with it the parameters l (level), i (interval) and the *limit*. The message first delivers the intended data to the receiving node. Secondly it serves as a request to cover all nodes in the interval $]r \oplus i * \frac{N}{k^l}, limit[$. Due to outdated routing tables some intervals might not seem to have any nodes even though they are populated. The responsibility of covering those intervals is delegated to the next interval.

If node n gives another node the responsibility to cover other preceding intervals and other nodes exist in those intervals, the node will trigger correction-on-use, and the routing information will be corrected at node n .

Via simulation, in [8] it was shown that coverage is always 100% and no redundant messages are ever received. Network sizes of 500, 1000, 2000, 3000, 4000 were used, to start 10% of the nodes were added then the broadcast was initiated while the other 90% of nodes joined. While these results show that

nodes joining the network are handled successfully they do not take into account nodes departing the network. Since this algorithm is based on the Efficient broadcast algorithm, nodes departing are likely to have a similar effect in both.

2) *Pseudo-reliable Broadcast*: In [9] Ghodsi notes that the current algorithms will be providing best effort delivery in the presence of failures. In many cases a best effort broadcast may not be sufficient. Assuming the initiating node attempts to broadcast and it has $M = \log_2(n)$ pointers (as in Chord), the broadcast will partition the space into M intervals, the last of which will cover roughly half of the identifier space (see figure 1). If the node responsible for the last partition was to fail then roughly half the nodes will not receive the broadcast.

In the Pseudo-reliable Broadcast algorithm every broadcast has a globally unique random identifier associated with it, which is included in every message. Nodes keep track of previously seen identifiers and filter any message which has an identifier previously seen, hence redundant messages are filtered. Each node has an ACK set, used to keep track of the children it is responsible for. In addition to keeping the identifies of the children the algorithm also keeps track of the interval delegated to each child. The algorithm is made resilient to failures by using a bulk operation [9] to resume after a failure. Whenever a node suspects that one of its children has failed, it uses the ACK set to determine the interval delegated to the failed node. Thereafter, the bulk operation is used to cover all nodes in that interval.

The failure detector suggested uses timeouts, meaning it could potentially give false-negatives, suspecting that an alive but busy/slow node has failed. The only consequence of this however will be wasted bandwidth consumption from the redundant messages being sent.

The time it takes for a broadcast to complete depends on the depth of the broadcast tree. Therefore, using a timeout when waiting for an acknowledgment from a child is problematic, as the parent does not know the depth of its sub-tree and therefore cannot determine the time to wait before triggering a timeout. Because of this issue the failure detector is assumed to be implemented independently from the broadcast, this can be achieved by the failure detector periodically sending a message to its children and awaiting an acknowledgment.

However, this algorithm still sends redundant messages after a failure, and could be improved. Consider a node who's children are all done covering their delegated intervals, except one. If the node fails, it's parent will detect that and reassign the interval to a new node. The new node will attempt to cover all of the original nodes children, rather than just the remaining one. To avoid these redundant messages, nodes could periodically send an update of their current ACK set to their parent. If this was the case, when a node failed only the children it was waiting for would need to be covered rather than all of them.

3) *Recursive Partitioning Search (RPS)*: In [11] a framework called Recursive Partitioning Search (RPS) for blind search over structured peer-to-peer overlays is presented by Vishnevsky et al., with a realization for Chord [21]. RPS is a

version of the Efficient Broadcast algorithm from section II-A, however it includes a TTL value as well as a limit, and a node will only broadcast the query to its sub-tree if it cannot satisfy the query itself first. With enough data replication the whole network does not need to be searched to find the data, and once we have found one copy we have no need to continue looking.

Vishnevsky et al. consider ways to restrict the number of nodes visited but without reducing the query success rate dramatically. In a network of size N , if the TTL value is set to $\log(N)$ then in a stable network the algorithm results in a 100% success rate, so it is a maximum reasonable value for the TTL. In most cases a smaller TTL value can still achieve high success rates because most resources are replicated among multiple nodes.

A variation of the algorithm to control the query message traffic is as follows: The query originator selects a partial list of its fingers and sends the query to them first. The tag in the message now contains two values, the nodes local search limit and the global search limit. Receiving nodes update the local limit as before, but do not touch the global limit. This variation allows a node to divide the search space and perform RPS sector by sector sequentially, if required.

In [10] realizations of RPS for both Chord [21] and Pastry [23] are presented. To reduce network load the use of go-stop signals (RPS+G) and local indices (RPS+L) are suggested.

Typically blind search algorithms generate multiple query messages that traverse the network concurrently, thus even when one of the query messages finds a result the others continue to search. To reduce the number of these messages some intermediate nodes inquire if the search originator has already found the target resource. The initiator either sends a "go" signal if the search is still pending, or a "stop" signal to terminate the query message. By using go-stop signals the scalability of blind search algorithms can be enhanced.

Vishnevsky et al. performed simulations over both Chord and Pastry, allowing a comparison of the two overlays. Using a network with 1,000 nodes, it was found that performing RPS over Pastry not only provided a higher success rate, but was faster than Chord. For a 95% success rate over Chord a TTL value of 6 was required, giving a mean delay of 99ms and 0.43 messages/sec per node. However, for a success rate of 99% over Pastry a TTL value of 3 was sufficient, giving a mean delay of 42.8ms and 0.27 messages/sec per node. This was mainly due to the Pastry routing table being roughly twice the size of the Chord routing table.

The use of RPS+G and RPS+L showed a reduction in network load by almost 50% (using the Chord implementation), as shown in Figure 3.

Extending this work by briefly looking at the effects of churn [12] notes that when RPS sends a query message to a node, it is expected that the node will propagate that message to other nodes which are within its search sector. If a node fails before doing so, all other nodes within its sector will not receive the query.

Simulations using Pastry showed that for a 1% increase

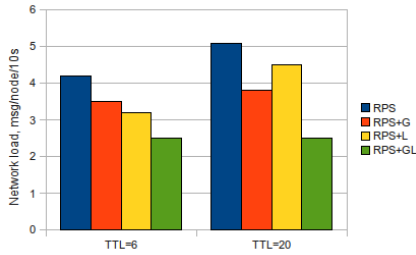


Figure 3. Network load of RPS and its modifications, using a Chord overlay.

in churn rate (in this case churn rate was taken to mean the ratio of nodes temporarily out of the network to the total number of nodes) there is roughly a 2% drop in the amount of nodes receiving the broadcast. However due to the duplication of documents in the network, RPS exhibits quite a limited dependence on peer churn. In the simulations performed there were 100 unique resources, with duplication ranging from 0.01% to 10% depending on the documents popularity (modeled using a zipf-like distribution). For each simulation 1,000 queries were sent. What resource to query for was chosen based on the resources popularity. RPS showed to have a success rate of about 99% even if 15% of peers have failed. In [24] we discuss the effects of different data replication strategies on churn.

4) *A Partition-based Broadcast Algorithm:* Another partition based approach is proposed in [13] which works by hierarchically partitioning the identifier space into two subspaces to construct a binary partition tree. Upon receiving a broadcast message with a limitation value l , each node n partitions its limited identifier space (n, l) into two subspaces (n, f) and $[f, l)$, where f is the closest finger node to the limitation value l . Next, the node n , selects the first finger found in (n, f) as its left child, and f as its right child. Finally the node forwards the broadcast message with the limitation value f to its left child and with the limitation value l to its right child.

The main difference between this algorithm and the Efficient Broadcast algorithm is that each node sends to at most two children. This has the effect that the broadcast tree is balanced and hence the branching factor is fixed at 2, but at the cost of increased hop count and hence time taken for a broadcast to complete.

5) *Dynamic Querying over DHT:* In [14] Talia and Trunfio propose a blind search method known as Dynamic Querying over Distributed Hash Tables (DQ-DHT) which uses the Dynamic Querying technique [15] originally developed for unstructured networks in combination with the Efficient Broadcast algorithm described in section II-A. Dynamic querying is interested in ensuring only the required amount of results are obtained from a search to avoid wasted effort in finding 100% of matches, this is done using a small TTL and iteratively increasing it as required. This approach is converted to work over DKS by Trunfio et al. in [16].

6) *Efficient Broadcast in P2P Grids:* The Efficient Broadcast in P2P Grids algorithm [17] combines the Efficient

Broadcast method described in section II-A with an epidemic distribution scheme as is often found in unstructured networks.

Like the Efficient Broadcast algorithm, the broadcast message contains a limit argument l . When a node receives the broadcast message, it picks the finger nearest to the middle (m) between its own address and l , and sends the broadcast message to it. The node itself continues the broadcast distribution within an interval between itself and $m - 1$. However it is again noted that a failure will make the entire subtree under the node unreachable. To solve this, all nodes that have received the broadcast, periodically forward the broadcast information to their randomly chosen neighbours as part of epidemic communication.

Using simulations in [17] it is shown that the combination of efficient broadcast and epidemic distribution will reach more nodes that simply efficient broadcast does under churn, and significantly faster than purely epidemic communication.

Due to its use of epidemic communication, this method obvious does result in duplicate messages being generated, however through simulation it is shown that before failure the algorithm distributes the broadcast almost entirely by means of efficient broadcast, with the message complexity increasing after simulated node failure. It is important to note that the message complexity is always significantly lower than that of purely epidemic communication.

7) *An Efficient Broadcast Algorithm:* In [18] an algorithm is proposed in which the ring is split into d equal sized partitions. Each partition has a responsible node, which is the first node encountered in the partition when moving clockwise around the ring. To contact the node a message can be routed using the underlying DHTs routing function. This means that the responsible nodes don't need to be in the partitioning nodes routing table, however it does mean traversing one level in the broadcast tree is not strictly a one-hop operation like in the efficient broadcast algorithm.

In a similar way to the Pseudo-reliable Broadcast algorithm (section II-A2) each node maintains an ACK set, containing the nodes they have forwarded the broadcast to and their allocated partition. After a node has received an ACK from all of its children (or if its a leaf node, right away), it will send an ACK to its parent node. Nodes maintain a timer for each partition, if a node does not receive an ACK from a child node within the given timeout it will select the next node from the child nodes successor list and retransmit the broadcast message to the this new node. At first this may sound good, but it seems to have a major flaw; if the child node is dead how can its successor list be accessed? A possibly work around could be to just retransmit the message to the partition again using the DHTs routing function; if the first node has died it should eventually be removed from the routing tables by the DHTs maintenance algorithm.

The timeout used by a node at layer i in the broadcast tree is calculated using $T_i = n_{succ}^{h-i-1} \times T_B$, where n_{succ} is the number of successors a node has (a design choice of the DHT), and the base timeout is similar to that in TCP, defined as $T_B = 2(\Delta + 4\sigma)$ where Δ is the mean time taken for one

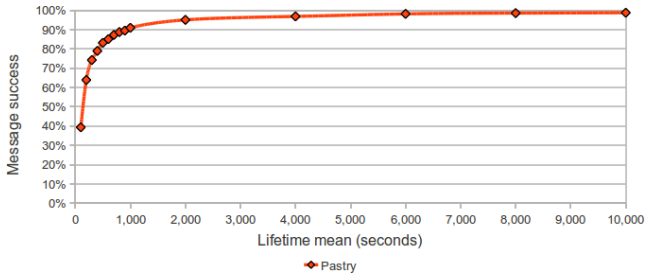


Figure 4. Effects of churn on Pastry broadcast success rate.

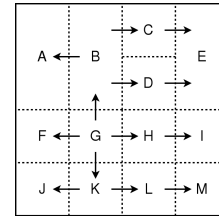


Figure 5. Broadcasting a message from node G in a 2-dimensional CAN overlay with 13 nodes.

hop and the standard deviation σ is 0.1Δ . In the paper there is no indication as to what h stands for, however for the formula to make sense we assume it must refer to the depth of the broadcast tree. This however is a problem as it is not possible for a node to actually know this value. In [25] an algorithm for network density estimation is presented that could be used to estimate the depth of a broadcast tree, however it only provides a rough estimate and adds extra delay and message complexity to the operation.

B. Pastry’s broadcast mechanism

In [19] a method for broadcast over Pastry is described. While not specifically designed with search in mind, it can still be used to broadcast complex queries.

Each pastry node maintains a routing table, a neighborhood set, and a leaf set. In a network of size N using identifiers with base 2^B , each nodes routing table is designed with $\log_B(N)$ rows, where each row holds $B - 1$ entries. All the entries at row r of the routing table each refer to a node 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.

A node broadcasts a message by sending the message to all nodes in its routing table; each message is tagged with the routing table row. When a node receives a message tagged with routing table row r it forwards the message to all nodes in its routing table with rows greater than r . This continues until a node receives a message tagged with r and it has no entries in rows greater than r .

In [26], using simulations with a Pastry network of size 10,000 and average session times of $\{5, 15, 30, 60, 120, 600\}$ minutes, it is shown that the query success rate starts to drop dramatically when the average session time drops under 30 minutes.

We ran simulations with a 10,000 node Pastry network with lifetime mean ranging from 100ms to 10,000 ms which showed similar results, as can be seen in figure 4.

[27] presents a performance analysis of Pastry’s broadcast mechanism using an analytical model and simulations. The replication load and hop count are studied, with comparisons drawn to Scribe [28].

C. Content Addressable Network (CAN) broadcast

A solution for application-level multicast in CAN is presented in [20]. In their solution multicast is performed by

creating a mini-CAN of participating users, then broadcasting over that CAN. While using this technique specifically for complex queries isn’t suggested, broadcasting is the basis of most complex querying techniques.

CAN is a rather unique type of overlay, designed around a virtual d -dimensional Cartesian coordinate space. The entire coordinate space is dynamically partitioned among all the nodes in the system such that every node owns an individual zone. Keys are mapped onto a point in the coordinate space using a consistent hash function, and the responsible node is the node whose zone contains that coordinate.

In a CAN with dimension d , each node has at least $2d$ neighbours; one to move forward in dimension d and one to move backwards. To initiate a broadcast the source node forwards the message to all its neighbours. When a node receives a broadcast message from a node with which it neighbours along dimension i , it will forward the message to those neighbours along dimension $1 \dots (i - 1)$ and the neighbours in dimension i on its other side. To prevent the message from looping back around a node does not forward a message along a particular dimension if that message has already traversed at least half-way across the space from the source coordinates along that dimension. An example broadcast can be seen in figure 5.

It is worth noting that for a perfectly partitioned coordinate space the algorithm ensures each node receives the message exactly once. For an imperfectly partitioned space however, a node may receive the same message from more than one neighbour. For example, in figure 5, node E would receive a message from both neighbours C and D .

III. CONCLUSIONS

This paper presents an overview of current blind search methods for structured P2P networks. It should be obvious that to perform blind search with support for full-text queries the query must be processed locally at each node, and as such the problem of blind search is almost identical to the problem of efficiently broadcasting; with the difference that queries need not always reach all nodes to be successful. Broadcast-based schemes allow for processing of any type of query but the total message complexity increases linearly with the network size. While this is much less efficient than standard operations in a DHT, which can usually be completed with $\log(N)$ messages, it is a major improvement over flooding in unstructured networks where large numbers of duplicate messages are generated. It is however important to note that

broadcast-based schemes tend to perform badly under high churn rates [24]. Some methods (II-A2, II-A7) attempt to handle this by detecting failures with a timeout and retransmission, however this adds to both the complexity and search latency; other methods (II-A6) combine efficient broadcasting with flooding, allowing some redundant messages to be generated, resulting in a higher success rate than efficient broadcast but more quickly and efficiently than flooding alone. From this we can see that while it is possible to broadcast with near 100% success rate, all approaches have a negative effect on some other aspect of the system.

Methods for restricting the search space (II-A3, II-A5) show that it is possible to reduce the overall number of nodes queried without altering the success rate, and due to data replication a 100% message success rate is not necessarily required to achieve a 100% query success rate.

We feel there is a notable lack of work on performing complex queries in one-hop networks, and in future work aim to adapt the efficient broadcast algorithm to work on the variable-hop overlay Chameleon [29], which aims at having one-hop performance for high bandwidth peers and multi-hop performance for low bandwidth peers. Instead of trying to recover from failures we aim to reduce the number of failures in the first place by taking advantage of Chameleons high routing table accuracy, and mitigate the effect of failures with the use of a novel data replication strategy. Making use of the fact that Chameleon produces a variable-hop network will allow forwarding of messages to $2 \dots N$ nodes depending on available bandwidth, allowing for a heterogeneous network in which low bandwidth peers use less bandwidth than required by efficient broadcast but with higher performance assuming a high enough proportion of high bandwidth nodes.

REFERENCES

- [1] M. Castro, M. Costa, and A. Rowstron, "Should we build Gnutella on a structured overlay?" *Computer Communication Review*, vol. 34, no. 1, pp. 131–136, January 2004.
- [2] Y.-J. Joung and L.-W. Yang, "Wildcard Search in Structured Peer-to-Peer Networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 11, pp. 1524–1540, November 2007.
- [3] Y.-J. Joung, L.-W. Yang, and C.-T. Fang, "Keyword Search in DHT-based Peer-to-Peer Networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 46–61, January 2007.
- [4] C. Schmidt and M. Parashar, "Scalable Keyword Searches with Guarantees in Peer-to-Peer Storage Systems," 2002.
- [5] —, "Enabling flexible queries with guarantees in p2p systems," *IEEE Internet Computing*, vol. 8, no. 3, pp. 19–26, May 2004.
- [6] —, "Flexible Information Discovery in Decentralized Distributed Systems," in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, Seattle, Washington, 2003.
- [7] S. El-ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient Broadcast in Structured P2P Networks," in *2nd International Workshop On Peer-To-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, 2003.
- [8] A. Ghodsi, L. O. Alima, S. El-ansary, P. Brand, and S. Haridi, "Self-Correcting Broadcast in Distributed Hash Tables," in *15th International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, Marina del Rey, CA, USA, 2003.
- [9] A. Ghodsi, "Distributed k-ary System: Algorithms for Distributed Hash Tables," Ph.D. dissertation, The Royal Institute of Technology (KTH), 2006.
- [10] V. Vishnevsky, A. Safonov, M. Yakimov, E. Shim, and A. D. Gelman, "Scalable Blind Search and Broadcasting in Peer-to-Peer Networks," in *6th IEEE International Conference on Peer-to-Peer Computing (P2P'06)*. Cambridge, UK: IEEE, 2006, pp. 259–266.
- [11] —, "Tag Routing for Efficient Blind Search in Peer-to-Peer Networks," in *11th IEEE Symposium on Computers and Communications (ISCC'06)*. IEEE, 2006, pp. 409–416.
- [12] —, "Scalable blind search and broadcasting over Distributed Hash Tables," *Computer Communications*, vol. 31, no. 2, pp. 292–303, February 2008.
- [13] K. Huang and D. Zhang, "A Partition-Based Broadcast Algorithm over DHT for Large-Scale Computing Infrastructures," in *4th International Conference on Advances in Grid and Pervasive Computing (GPC '09)*, ser. Lecture Notes in Computer Science, vol. 5529. Geneva, Switzerland: Springer-Verlag, 2009, pp. 422–433.
- [14] D. Talia and P. Trunfio, "Dynamic Querying in Structured Peer-to-Peer Networks," in *International workshop on Distributed Systems: Operations and Management: Managing Large-Scale Service Deployment*, vol. 5273, Samos Island, Greece, 2008, pp. 28–41.
- [15] Fisk, A., "Gnutella Dynamic Query Protocol," 2003. [Online]. Available: http://www9.limewire.com/developer/dynamic_query.html
- [16] P. Trunfio, D. Talia, A. Ghodsi, and S. Haridi, "Implementing Dynamic Querying Search in k-ary DHT-based Overlays," in *3rd CoreGRID Integration Workshop*, Hersonissos, Greece, 2008, pp. 253–264.
- [17] P. Merz and K. Gorunova, "Efficient broadcast in P2P grids," in *International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. Cardiff, UK: IEEE, 2005, pp. 237–242.
- [18] W. Li, S. Chen, P. Zhou, X. Li, and Y. Li, "An Efficient Broadcast Algorithm in Distributed Hash Table Under Churn," in *International Conference on Wireless Communications, Networking and Mobile Computing*, no. 60672086. IEEE, September 2007, pp. 1929–1932.
- [19] 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 *22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, vol. 2. IEEE, 2003, pp. 1510–1520.
- [20] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-Level Multicast using Content-Addressable Networks," *Networked Group Communication*, vol. 2233, pp. 14–29, 2001.
- [21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*. New York, NY, USA: ACM Press, 2001, pp. 149–160.
- [22] L. O. Alima, S. El-Ansary, P. Brand, and S. Haridi, "DKS(N, k, f): a family of low communication, scalable and fault-tolerant infrastructures for P2P applications," *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.*, pp. 344–350, 2003.
- [23] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *International Conference on Distributed Systems Platforms*, Heidelberg, Germany, 2001, pp. 329–350.
- [24] J. Furness and M. Kolberg, "The Effects of Churn on Complex Search Techniques," *unpublished*.
- [25] M.-Y. Lue, C.-T. King, and H. Fang, "Scoped broadcast in structured P2P networks," *Proceedings of the 1st international conference on Scalable information systems - InfoScale '06*, p. 51, 2006.
- [26] M. Castro, M. Costa, and A. Rowstron, "Debunking some myths about structured and unstructured overlays," in *2nd conference on Symposium on Networked Systems Design & Implementation*, vol. 2, 2005, pp. 85–98.
- [27] M. Wahlisch, T. C. Schmidt, and G. Wittenburg, "Broadcasting in Prefix Space: P2P Data Dissemination with Predictable Performance," *2009 Fourth International Conference on Internet and Web Applications and Services*, pp. 74–83, 2009.
- [28] M. Castro, P. Druschel, A.-m. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, October 2002.
- [29] A. Brown, M. Kolberg, and J. Buford, "Chameleon: An Adaptable 2-tier Variable Hop Overlay," in *6th IEEE Consumer Communications and Networking Conference (CCNC 2009)*. Las Vegas, NV, USA: IEEE, 2009.