Improving Scalability of Autonomic Systems: The Frequency-Aware Search Approach

Pedro Fonseca
University of Lisbon/LaSIGE
pmf@lasige.di.fc.ul.pt

Hugo Miranda
University of Lisbon/LaSIGE
hmiranda@di.fc.ul.pt

ABSTRACT

Resource and data indexing in distributed, self-manageable systems can leverage on the experience gained from peer-to-peer networks, often built using distributed indexing. This paper presents FASE, a distributed indexing algorithm for unstructured overlays with flat topologies. FASE combines a replication policy and a search space division technique to achieve low hop counts using a small number of messages. The unexpected departure of nodes from the overlay, which may be observed in heterogeneous networks built over an unreliable medium, is mitigated by a distributed monitoring algorithm designed with FASE in mind. Simulation results validate FASE efficiency when compared to other search algorithms. The evaluation of the distributed monitoring algorithm shows that it maintains FASE performance when subjected to a constant arrival and departure of nodes.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Data sharing; H.3.3 [**Information Search and Retrieval**]: Search process

General Terms

Peer-to-peer search

Keywords

peer-to-peer, resource location, unstructured overlays

1. INTRODUCTION

Distributed systems operating over heterogeneous environments, such as sensor networks, mobile networks and grids, call for autonomous systems obeying strong self-management requirements. Managing data or resources in such systems using some form of centralized index would be a step backwards from the distributed nature of autonomic systems, not to mention an undesirable point of failure. Instead, a distributed indexing approach, that can be easily adaptable to the heterogeneity of the underlying medium, should be preferred. Peer-to-peer (P2P) systems are typically designed to manage large-scale distributed indexes in heterogeneous environments. Autonomous systems can leverage on the existing research on P2P systems for data management. Of the different classes of P2P systems that have been recently proposed, unstructured overlays seem particularly suitable for this task, in account for their robustness and weak filiation constraints.

In contrast with their high resilience to membership changes, resource location in unstructured overlays with moderate resource consumption is a complex challenge. Original search algorithms for these kind of overlays were resource intensive (e.g. flooding [4]), and presented scalability problems. These were succeeded by hierarchical overlays using a two-tier approach (like [9]). In unstructured hierarchical overlays, a small subset of nodes are elected as *supernodes*. These are responsible for indexing the content of all the regular nodes they have associated and to propagate the queries they are unable to resolve to the remaining supernodes.

The introduction of supernodes results in an excessive unbalance, as it imposes a much higher resource consumption to a few nodes than to the majority (see for example [9]), what may not be acceptable by some autonomic systems with restrictive requirements. The present work is motivated by the observation that a fair network is also a more resilient one as, if all the nodes have the same role, the loss of any can be more easily tolerated. Moreover, a "fair" organization might appeal when nodes cannot (by design or due to policy) contribute with more resources than their peers. The challenge is how to achieve low latency searches and scalability in such networks, since flat, i.e. non-hierarchical, unstructured overlays that use flooding based search are not scalable.

The problem becomes more relevant since, in unreliable heterogeneous networks, nodes may abandon and join the overlay at a high rate, in a process designated as *churn*. The negative effects of churn are more significant on overlays which put additional effort on maintaining a predefined topology. Operating under churn, structured distributed hash tables (DHT) experience an increase in lookup latency as well as inconsistent results [13]. Additionally, DHT filiation has strong constraints and can have a higher cost than filiation in unstructured overlays as, instead of joining a random neighbor, a node is restricted to join nodes that are close (the closeness being measured by the proximity of the node's IDs).

In this paper we present the Frequency Aware SEarch

Autonomics 2008, September 23 - 25, 2008, Turin, Italy

algorithm (FASE). FASE uses a search space partitioning mechanism, combined with a replication policy. The algorithm is adequate for autonomous systems because it does not rely on a hierarchical architecture and does not require the exchange of routing information between nodes, which would impact on its capability to recover from node failures. The search is an uninformed search that leverages on the partitioning mechanism. Additionally, we implemented a distributed monitoring algorithm that improves the performance of FASE while subjected to churn and removes references to outdated resources.

The remainder of this article is organized as follows. Section 2 presents a survey of the related work, mostly addressing research on P2P networks. FASE and the monitoring algorithm are described in Secs. 3 through 5. The algorithm evaluation is presented in Sec. 6. Finally, Sec. 7 presents the final remarks.

2. RELATED WORK

The problem of devising efficient resource location algorithms in distributed systems has been actively researched, mostly by the P2P research community. Alternatives range from identifying relationships between the nodes to concentrate the indexes on the nodes presenting more resources, to name a few. In this section we survey several indexing and resource location strategies used in P2P networks.

A class of protocols establish semantic or interest based relationships between nodes to improve search. One of such algorithms, *Acquaintances* [3], keeps random *neighbor links* and *acquaintance links* to nodes which returned a positive response to a previous query. Following a similar rationale, in [15] peers establish shortcuts to others with similar content, to where queries are preferably forwarded. In the *Self-Organizing Super-Peer Network* (SOSPNet) [6] interest relations are established using supernode caches at regular nodes and file caches at supernodes. The cache entries are prioritized according to previous search results and increase the probability of nodes with the same interests knowing the same supernode and of supernodes knowing similar content. Algorithms that identify relations between nodes are an interesting approach. However, these algorithms assume that nodes have a diverse range of interests, maintained over time. It is unlikely that these assumptions hold for the generality of applications in autonomous systems, where large proportions of the nodes may cooperate to achieve some specific task or change their interests according to their role at the time.

Gia [2] adapts the overlay topology to the capacity of the participating nodes and directs queries to the higher capacity nodes. Furthermore, it defines a replication policy that dictates that the index of a node is replicated on its neighbors at the distance of one-hop. Gia preferably routes random walks to the nodes with higher capacity, thus more likely to know an item due to the one-hop replication policy. Queries use a single random walk which requires the use of "keep-alive" messages. Like query responses, "keep-alive" messages are sent via reverse path. The progressive adaptation of the topology towards an hierarchical overlay is an interesting aspect in Gia. However, as discussed in Sec. 1, it makes Gia ill-suited for autonomous systems.

Scalable Query Routing (SQR) [7] uses information encoded in probabilistic routing tables to forward queries to the (probable) location of the desired object. The routing tables store one Exponentially Decaying Bloom Filter (EDBF) for each neighbor link as well as a local EDBF, which indexes the node contents. In an EDBF, which is similar to the standard Bloom filter [1], the membership of an item is tested by a function $\theta(x)$ which yields the number of bits in the filter that are set to 1. Queries for a key x are forwarded to the node corresponding to the filter with greater $\theta(x)$. SQR periodically transmits routing table updates, by merging filters and decaying their information according to a parameter d, that is, the resulting filter bits remain set to 1 with a probability $\frac{1}{d}$. The maintenance of the probabilistic routing table implies a non-negligible message overhead. To mitigate this problem, information for each update degrades with the distance to the host, thus implying an increasing "noise". In consequence, a search too far away from the searched object behaves like a random walk. The problem is amplified in the presence of churn, where the loss of accuracy caused by out of date tables will be the rule rather than the exception.

3. FASE

The *Frequency Aware SEarch*, or FASE, is a search algorithm for flat unstructured overlays. FASE partitions both the key space and the nodes in a common "frequency space" and replicates the keys in nodes with corresponding frequencies. A frequency is defined to be any element of a finite and discrete set Φ. We define functions $f_N : N \mapsto \Phi$ and $f_K : K \mapsto \Phi$ mapping respectively the set of nodes (N) and keys (K) in frequencies. Mappings should uniformly distribute nodes and keys by frequencies. A non-uniform mapping will result in unbalanced partitions leading to a sub-optimal performance of the algorithm. A good example of a mapping function is the hash of a node IP address and port. The set Φ and functions f_N and f_K are known by every node in the network. Node neighbors are freely created and are expected to be from different frequencies. A "frequency path" is defined as an uninterrupted sequence of neighboring nodes of the same frequency.

A pointer is a $\langle$**Key,URL**$\rangle$ tuple, where a key is some entity that can be used for identifying the item and an URL is an unique, complete address to a node. We defer a more precise definition of a key to Sec. 5.1. A pointer maps a key to the unique identifier of the node that is storing the item referred by the key. In FASE, pointers to the items are stored on nodes with the same frequency of the item's key. The placement of pointers is restricted to the same frequency of the key but the placement of the original item is not constrained in any way. FASE improves search results by having nodes to forward random walks preferably to nodes in the same frequency of the search key.

FASE makes no assumptions about overlay structure, nor is overlay membership the focus of this work. The algorithm does not require node membership to be constrained by the node identification or that node insertion conforms to a given structure. However, it is assumed that the majority of nodes in the overlay have a constant degree. There are two reasons for this assumption. First, with a uniform distribution of frequencies (assumed to be provided by functions f_N and f_K) and for a given degree and number of frequencies, each node will have the same probability of having a neighbor with some frequency f, that is, a node in f can be reached from every node n, $f_N(n) \neq f$, with the same probability p. Second, a constant degree, that can be offered by mem-

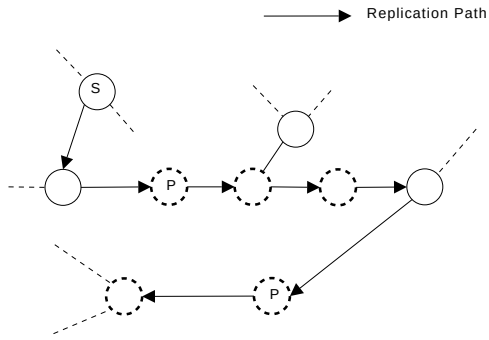


Figure 1: Pointer replication.

bership protocols such as Hyparview [8], evenly balances the load between nodes when combined with a uniform distribution of nodes and items over Φ.

3.1 Pointer Replication

To facilitate item location, nodes replicate pointers using REPLICATION messages. Each REPLICATION message carries one or more pointers with keys of the same frequency. Therefore, it is said that the REPLICATION message belongs to that frequency. For simplicity, we denote by $f_K(M_R)$ the frequency of the keys carried in message M_R.

REPLICATION messages carry a counter that defines the maximum number of replicas of each pointer to be stored, defined by a configuration parameter. The replica counter is decremented by every peer deciding to store a pointer and the message is discarded when the counter reaches 0. A REPLICATION message M_R is successively forwarded at each node $n \in N$ to one of n's neighbors (nn) not visited by the message. Preference is given to neighbors in the frequency of the message, that is, with $f_N(nn) = f_K(M_R)$.

A node n' stores the pointer when it belongs to the same frequency of the message and it has received a message from a node n with a different frequency, that is, $f_N(n') = f_K(M_R) \wedge f_N(n) \neq f_N(n')$.

The pointer replication algorithm is illustrated in Fig. 1, which shows the path of a REPLICATION message M_R starting at node S. The interrupted circles denote nodes in the same frequency of the message. Node S does not have any neighbor in M_R frequency, $f_K(M_R)$, hence the message is forwarded to a random neighbor; in the following hop there is a neighbor n with $f_N(n) = f_K(M_R)$ and subsequently the message is forwarded to that neighbor, which will store a pointer. The node that stores a pointer has a neighbor of the same frequency to where the message is forwarded, but that will not store the pointer since it is in the same frequency of the previous node. The figure shows that a REPLICATION message is always forwarded to (unvisited) nodes of the same frequency. The nodes labeled with P store the pointer carried by M_R. Note that the pointers are stored once per path. The forwarding of M_R stops when the replica counter reaches 0 or when all the neighbors of the current node have been visited before by M_R.

While the process of replication may sound costly, as several nodes may be visited before the replica counters reaches 0, keep in mind that it happens only once for a key or set of keys.

3.2 Item Retrieval

If the mapping functions, f_N and f_K, uniformly distribute nodes and keys by frequencies, the search space of some key should be approximately $\frac{|N|}{|\Phi|}$. Depending on the number of frequencies $|\Phi|$, this may represent a substantial reduction of the search space. The retrieval algorithm takes advantage of this property by forwarding random walks preferably to nodes in the same frequency of the key.

To locate an item (through the location of the keys contained in the disseminated pointers), peers use QUERY messages, forwarded in one or more random walks. The message includes a search string that is mappable to the same frequency of the key. Queries are forwarded to a neighbor picked randomly from the set of unvisited neighbors with the same frequency of the key; or, if there is no node with the same frequency, to a random unvisited neighbor. That is, once a random walk enters a path of nodes in the frequency of the target key, it will follow that path.

When a QUERY message is delivered to a node, the node first verifies the items that are stored at the node. If the node has the same frequency of the query, then the pointers stored in the node are verified. Upon a hit, i.e. when an item corresponding to the query criteria is found, the search terminates and the associated URL is returned.

4. MONITORING ALGORITHM

The monitoring algorithm main contribution is to prevent query performance degradation due to the progressive loss of replicated pointers resulting from churn. As it is closely related to FASE, henceforth we will refer to this algorithm as FASE+. FASE+ aims at maintaining a sufficient number of pointers. Without pointers, items may still be reachable, but only within an abnormally high hop count, as the random walks will be biased to nodes where the item cannot be found (unless the key frequency is coincident with the frequency of the node storing the item).

4.1 Backup Creation

A strategy to avoid the loss of pointers based on their repeated dissemination could present a high overhead, since the periodic repetition of the dissemination algorithm could generate an unaccountable number of pointers, that would increase over time, thus wasting bandwidth and memory at the nodes. FASE+ follows a decentralized monitoring approach, storing backup pointers, that replace the primary pointers when the later are lost due to the departure of nodes from the overlay.

When a node, lets call it N_p, is requested to store a pointer of some item, it immediately creates and forwards a backup. The backup is forwarded to a neighbor of N_p in the same frequency. If there is not any neighbor in the same frequency, the first neighbor is chosen from the node's ordered list of neighbors. When the backup message reaches a neighbor in the same frequency that is not holding a pointer of the item, the backup is stored and the message is discarded. The node storing the backup associates with it the identity of N_p. When a node stores a backup, it sends back to the backup source (that is, N_p) a message to register itself. In this manner, nodes holding pointers and their respective backups monitor themselves. Therefore, there is, at most, one backup per pointer.

Let Δ be the timeout for backup verification, N_p a node

holding the pointer p and N_b the node holding b, that is the backup of p. Primary and backup nodes periodically cross-check each other according to the following rules:

- Every Δ, counting from the moment when the node joined the overlay, N_b will try to contact N_p. If N_p has failed, b is self promoted to a primary pointer p' and a new backup b' is created using the procedure described above.
- Every Δ, N_p will try to contact N_b. If N_b has failed, N_p creates a new backup using the procedure described above.

Backup forwarding is a low overhead operation because, in most cases, some one-hop neighbor of N_p will satisfy the conditions required for becoming a backup holder.

4.2 Stale Items Update

In an overlay subject to churn, pointers to items which are no longer available may persist. FASE+ uses two strategies to deal with this problem. For simplicity, we designate the node referred by a pointer as the *holder*, that is, a node that holds some referred item. First, before creating a backup from a pointer p, the node that has p tries to contact the holder. If the holder is down, p is removed and the backup creation is aborted.

The second strategy is implemented in the search algorithm. Whenever a query finds a pointer which matches its search criterion, it contacts the holder. If the holder is unreachable or does no contain the item, the pointer is removed and the query continues. In this scenario, the backup of the pointer that is removed would still be out of date. For this reason, when the loss of a pointer is detected and a backup is to be promoted, the node tries to contact the holder and, upon verifying that the holder is down, removes the backup.

5. IMPLEMENTATION

In this section, we discuss the implementation of the algorithms presented in the previous sections.

5.1 Key Definition

FASE does not enforce a strict definition of key. Instead, it can be adapted to the preferred type of search. Two possible definitions of key are *i*) a **String**, one or more words describing the item, or *ii*) an **ID**, the unique id identifying an item, e.g. a cryptographic hash of the item.

FASE only requirement for a key is that it must be mappable to a frequency. To broaden the possible search criteria, keys should be as flexible as possible. We note that a key does not need to identify an unique item. For instance, if it is a **String**, it can be a single word that is mappable to several items. Following the same rationale, an item may be associated to different keys, for instance, based on a set of strings that describe the item.

5.2 Dissemination of Multiple Keys

A single REPLICATION message can be used to announce multiple items and/or multiple keys of the same item, if we extend the syntax of the pointer construct. Since the keys may share a common frequency, a pointer can carry multiple keys of the same frequency. In FASE, the pointers transmitted in REPLICATION messages can be defined as a tuple $\langle$**Set**$\langle$**Key**$\rangle$**,URL**$\rangle$, that refers multiple keys sharing the same frequency.

The key set can be represented in a space-efficient way by using Bloom filters. To advertise multiple keys belonging to different frequencies, peers prepare one REPLICATION message per frequency (unless there are no keys belonging to the frequency) and apply the replication algorithm independently to each. Given that the number of frequencies is expected to be small in comparison to the number of keys, we expect the multiplication of Bloom filters per peer to present an acceptable overhead to the replication costs.

5.3 Random Walk Forwarding

We opted for including the list of traversed nodes in REPLICATION messages. Although it may lead to significantly large messages, this approach permits to nodes to avoid next hops that have been previously visited so that the probability of the message being dropped is reduced. Additionally, it avoids cycles. As REPLICATION messages are generated only once per frequency and per node and their length is bounded by the number of pointers to insert, we expect the overhead to be acceptable.

QUERY messages are more frequent than REPLICATION messages and can use multiple random walks. Storing every ID of the visited nodes in all the random walks could add a non-negligible cost to the transmission of queries. In this case we opted for a hybrid approach. Nodes store the IDs of the queries that visit them. We assume that the storage cost of a circular queue with capacity for a large set of IDs is acceptable. When a node, n, receives a previously seen ID, it sends the query back to the last hop, n'. If this happens, n' forwards the query to any neighbor except n. This method alone could increase the search latency, for instance, if two nodes shared many neighbors and both were previously visited. In this case, the query could waste a few hops being forwarded from one node to the other. For this reason, we store a small set S of node IDs in the random walk message; its length should be set to a value that results in no more than a small increase of the QUERY message size. S contains the IDs of the more recently visited nodes. Random walks are not forwarded to any node belonging to S. Concerning the termination of random walks, we adopted the "adaptive termination" [10] method, where walkers periodically contact the source of the query to check if they should terminate.

6. EVALUATION

FASE is first evaluated using a stable network, that is, a network were the nodes remain from the beginning to the end of the simulation. Afterward, FASE and FASE+ (that is, FASE coupled with the distributed monitoring algorithm) are evaluated in a network subject to churn. Simulation results were obtained using PeerSim [11] event driven simulation engine. The simulations use a 10,000 node network. For each test case we used 10 different topologies with constant degree, generated by HyParView [8], running multiple simulations. Plots present the aggregation of the multiple simulations and the standard deviation.

Each simulation creates 1000 unique data items each advertised by a distinct node selected at random. In some application scenarios it is assumed that items have a non-uniform distribution that depends on their popularity. We opted for a uniform distribution model where each item is

hosted by a single node, thus representing the least popular items of the networks. The rationale for this model is that the search for rare items is the worst case scenario and that an algorithm that shows good performance in the search for rare items will show good performance in the search for more popular items, as popular items will be even more replicated. Moreover, this model can be used to represent systems where resources are unique or limited such as a set of specific sensors in a sensor network (the data from the sensor can be replicated but the sensor itself cannot). Experiments where conducted with different numbers of pointers of each item. The *replication ratio* is given by the number of replicas over the total number of nodes in the network.

Queries are performed for items selected uniformly at random. In the case of a stable network, simulations have a deployment phase, where items are replicated, followed by a querying phase, where each node issues a query. In the case of a churning network, the querying phase is split in two: one *warm-up phase*, where items entering the overlay are replicated and nodes join and depart the network but no queries are performed, and a *search phase*. In the first half of the search phase, nodes arriving the network perform a query for existing items; on the second half new items cease to be announced, to measure the effect of churn over old content.

The simulations are used to evaluate two metrics:

Hop Count the number of hops until an item is found.

Hop Limit the highest hop count for a given success rate, e.g. the hop limit for 90% is the highest hop count observed in the 90% most successful queries.

If multiple random walks (RW) are used, we observe the *message count*, that is, the number of messages effectively used by a query. Additionally, we observe the *failure ratio*, that is, the ratio of failed queries for items existing in the network. A query is considered failed when its last RW exceed its TTL without finding an item or when it cannot be forwarded; the TTL was set to the same size of the network.

6.1 Number of Frequencies and Degree

In this section we investigate the effect, on FASE performance, of the number of frequencies and of $\frac{D}{|\Phi|}$, the ratio of the degree (i.e. the average number of neighbors of each node) to the number of frequencies.

We experimented several values of $|\Phi|$ over networks with $degree = 10$. Queries were performed using one random walk. Figure 2 depicts the average hop count in function of $\frac{D}{|\Phi|}$ for various replication ratios. The replication ratios in the x axis are presented in logarithmic scale for legibility.

With the exception of the 0.04% replication ratio, the higher values for the average hop count occur when $\frac{D}{|\Phi|}$ is lower than 0.5. The hop count increases for all the replication ratios when the number of frequencies is much higher than the degree, in this case, when $|\Phi| \geq 21$. As each node is only aware of other nodes with, at most, 10 different frequencies and FASE performance depends on finding nodes with the same frequency of a query at a close distance, the hop count increases. If the pointers have a small replication ratio (0.04%) the lowest hop count occurs when the ratio is between 0.7 and 0.5. In this case, the search algorithm benefits from a higher number of frequencies, as increasing $|\Phi|$ reduces the search space (which should be $\frac{|N|}{|\Phi|}$). However, the impact of higher replication rates in performance is greater than the increase of frequencies and, for the replication rates that achieve lower hop counts, from the tested values, the best results are achieved when $\frac{D}{|\Phi|} = 1.1$. The lowest number of failures occur for the same ratio, as it is shown in Fig. 3.

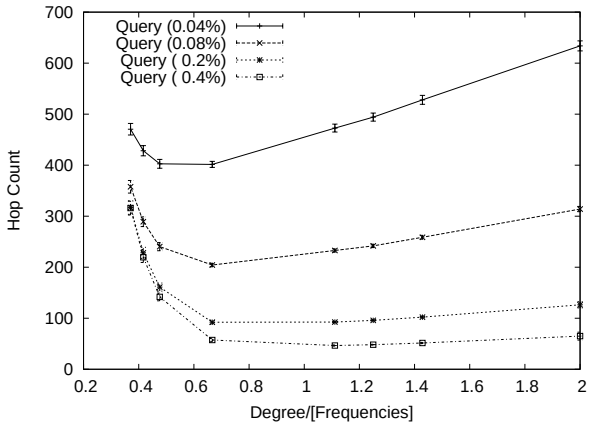


Figure 2: $\frac{D}{|\Phi|}$ and average hop count.

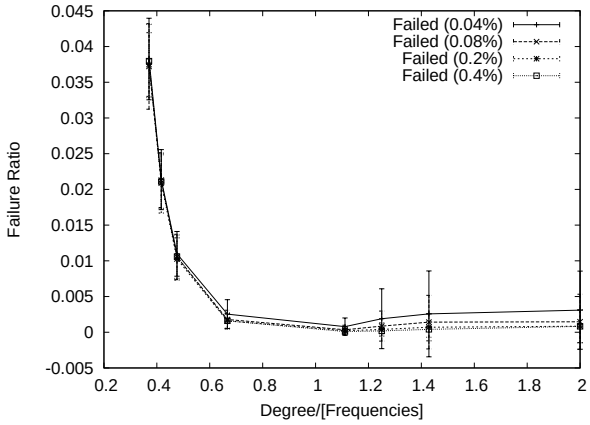


Figure 3: Failure ratio.

Figure 4 depicts the replication costs in function of $\frac{D}{|\Phi|}$ for various replication rates. The figure shows that for $\frac{D}{|\Phi|} = 1.1$, the hop count is 45% lower than in the worst case. As expected, the number of hops spent in replicating pointers grows with the probability of having multiple neighbors of the same frequency. As the average path length increases, replication messages are required to traverse more hops before storing each replica. While 1.1 is not the optimal $\frac{D}{|\Phi|}$ value for replication, for a process that will happen at most once per node and per frequency, it presents a good trade-off between the replication cost and query hop count. This is justified by observing Figs. 4 and 2, where it can be seen that, for replication rates that allow the query hop count to reach an acceptable number of hops, a ratio of 1.1 has the lowest hop count, and a replication cost approximately 50% below the ratio that consumes more replication messages. Additionally, a value of 1.1 has the lowest number of failed queries (Fig. 3).

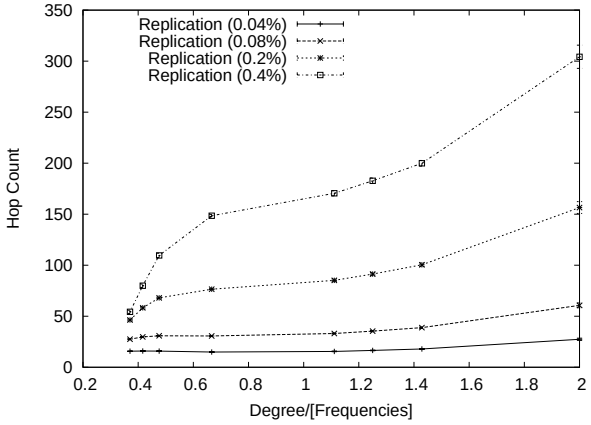


Figure 4: Frequencies vs. replication costs.

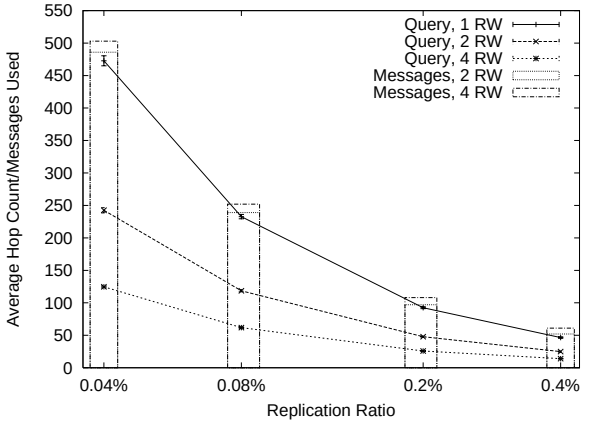


Figure 5: Hop count and message cost.

6.2 Stable Network Evaluation

In this section we observe the behavior of FASE in a stable network. We present results for two degrees (10 and 30), using a $\frac{D}{|\Phi|}$ ratio of 1.1, that is, we used respectively 9 and 27 frequencies. Furthermore, we studied the hop count and number of messages used with multiple RW. The "adaptive termination"[10] parameter of the RW was configured so that each checks for termination every four hops. Unless otherwise noted, the data refers to the 10th degree topologies.

Figure 5 depicts the average hop count and the number of messages used by multiple RW (that is, the sum of the hops traversed by the multiple walkers) for the various pointer replication values. The figure proves the advantage of using more than one RW. When 4 RW are used the hop count is 70% and 72% lower (respectively, with 0.4% and 0.2% pointer replication). However, due to the increased probability of finding an item and the adaptive termination, the total number of messages used only increases by 32% and 16%, when compared to the single random walk version.

Since the number of frequencies is greater with a higher degree, the number of nodes and keys per frequency is reduced. Therefore, with equal replication ratios (that is, with a equal number of pointers), the use of a topology with a higher degree reduces the query hop count and the number of messages used, as is shown in Fig. 6. This presents an interesting trade-off: when deploying FASE, or a similar search algorithm, one has to weigh the cost of establishing and maintaining a high number of connections against the cost of search and of replication. In FASE, a topology with $degree = 30$, with a replication ratio of 0.04%, achieves lower query hop counts than with a replication ratio of 0.08% in a topology with $degree = 10$, since the lower degree topology has more pointers but also more nodes per frequency. Unless the cost of maintaining more connections during the lifetime of the node is higher than the cost of search, having a high degree is preferable.

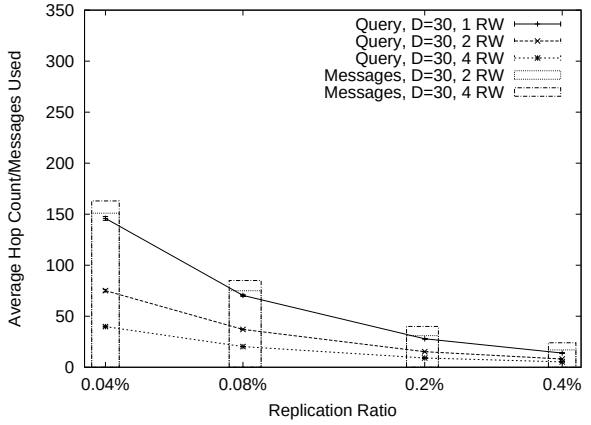


Figure 6: Hop count and messages, $deg = 30$.

6.3 Churning Network Evaluation

The goal of this evaluation is twofold. First, to find if FASE is capable of sustaining the pointer loss and hop count degradation observable in FASE in the presence of churn. Second, to verify if the adopted churn mitigation strategy in FASE+ is adequate, particularly if it can use spaced enough updates to maintain a low overhead.

In the evaluation of an overlay subject to churn, FASE and FASE+ simulations use a $degree = 10$ network, four random walks and replicate pointers by 0.2% and 0.4%. This choice of parameters is justified by FASE results in the stable network evaluation, as it performed better using these values. It was shown in the previous section that using multiple random walks decreases the hop count with a moderate cost and that replication ratios lesser than 0.2% result in high hop counts. The length of the simulation warm-up and search phases was set to 10800s. In the case of FASE+, the Δ parameter was set to 900s and 1200s.

We examined the average query hop count, the hop limit for 90% success and the query hop count over time. Additionally, in the case of FASE+, we observed the changes in number of pointers and backups during the simulations as well as the total number of items losing all their pointers during the simulation.

A cleanup algorithm was executed during the simulation for both FASE and FASE+. Note that this not affect the evaluation of FASE, since all queries are for items that are present in the network and, if a query finds a key but the item is no longer present, the result is discarded but not counted as failed.

6.3.1 Churn Model

The session length metric is frequently used in the literature [14, 12, 9, 16] to characterize churn in P2P networks. It is defined as the time between the arrival and departure of a node. [16] identified Weibull and log-normal distributions as the more accurate distributions for modeling session lengths. Churn is modeled according to workloads observed in P2P networks, thus mimicking the effect of user behavior. The model can also reflect the effect random failures. In this paper, we assume that churn estimations for P2P can be considered as a worst case scenario for autonomous networks.

To simulate churn we opted to distribute the session length of the nodes in the overlay according to the Weibull distribution using shape (k) and scale (λ) parameters for the three data sets (designated as "Red Hat", "Debian" and "FlatOut") presented in [16]. The RedHat and Debian sets have a similar session length distribution, although Debian has a slightly higher percentage of long lived peers. In our evaluation these sets present similar results. For this reason, results for the Debian set have been omitted. The interested reader is referred to [5], where full results are presented. The FlatOut set has less extremely short and extremely long lived peers than the previous sets.

We simplify our model by maintaining the network size constant. When a node departs from the overlay another node joins, not necessarily in the same position, but establishing connections to nodes which are accepting new neighbors, thus contributing to maintain node degree. Nodes that depart from the overlay remain offline during the remainder of the simulation.

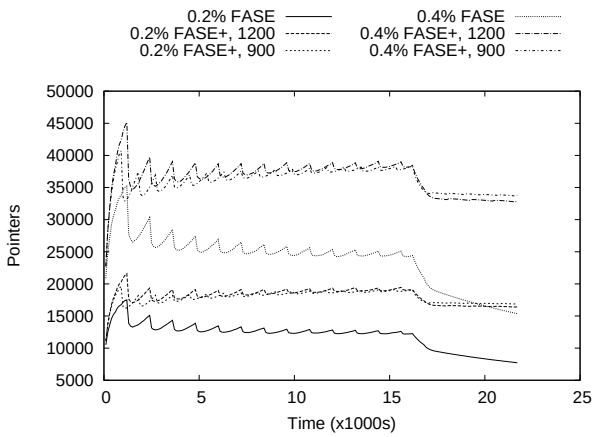
6.3.2 Pointer and Backups Evolution

In this section we evaluate the ability of FASE+ to maintain pointers under churn. The evolution of the number of pointers and backups was recorded with intervals of 100 seconds.

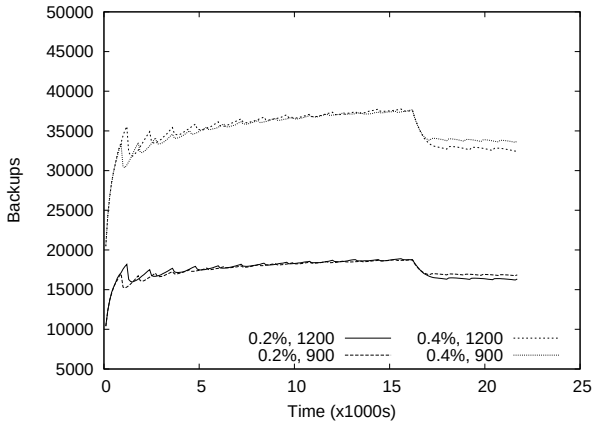
Figures 7 and 8 show the evolution of the total number of pointers (for all items) over time, in FASE and FASE+, for the RedHat and FlatOut sets, with update intervals of 900s and 1200s.

The peaks observed in the figures can be explained by the stale pointer cleanup algorithm. As they occur at times which match the Δ intervals counting from the simulation start. But for such peaks to happen, a high percentage of nodes should be timing out at the same time. There are two explanations for this. First, as Stutzbach et al. found in [16], a large portion of peers are stable while short-lived peers constitute most sessions (as they have a high turnover rate). Therefore, the probability of a node at the start of the simulation to have a long uptime (relative to the length of the simulation) is not low. This is confirmed by the session time distribution of the peers. Second, the granularity of the snapshots (100s) which implies that every node that performs cleanup within that interval contribute to the peaks.

During the first half of the querying phase the patterns are similar although FASE+ is able to maintain a significantly higher number of pointers. Observe that the replica growth is due to nodes with new items entering the overlay and executing their initial replication algorithm. In the second half, after the announce of new items is stopped, both algorithms lose pointers, albeit FASE loses them at a higher rate whereas FASE+ is capable of maintaining the number of pointers after an initial decrease. The behavior of FASE+ in the second half of the simulation shows that the algorithm is capable of maintaining the availability of older items. Figures 7(b),and 8(b) proves that the number of backups is kept on par with the number of pointers. Another interesting result is that the algorithm maintains the number of backups and replicas even when each update is spaced by 20 minutes.



(a) Evolution of the number of pointers.



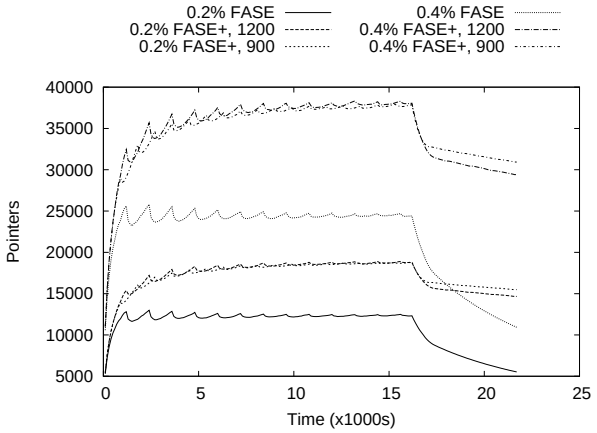
(b) Evolution of the number of backups.

Figure 7: Pointers and backups (RedHat).

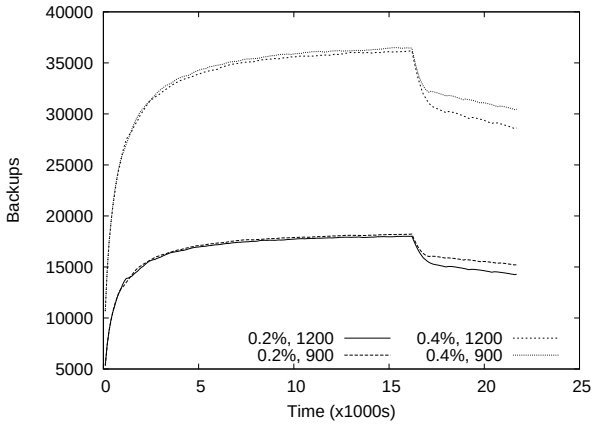
FASE+ also reduces the number of live objects losing all their pointers. Figure 9 presents the cumulative count of objects that lost all their pointers present in the overlay. In the figure, the lines that present the results for FASE+ with various parameters overlap each other. As expected FASE with a replication ratio of 0.2% loses more items than with a replication ratio of 0.4%, as with more pointers there is a greater probability that some survive until the end of the simulation. FASE+ is able to substantially reduce the number of items that lose all their pointers.

6.3.3 Query Performance

Figure 10 presents the average hop count and the hop limit for 90% success for the RedHat set, with an update interval of 1200s. In comparison with FASE, in the RedHat set FASE+ shows a lower average hop count, less 84%

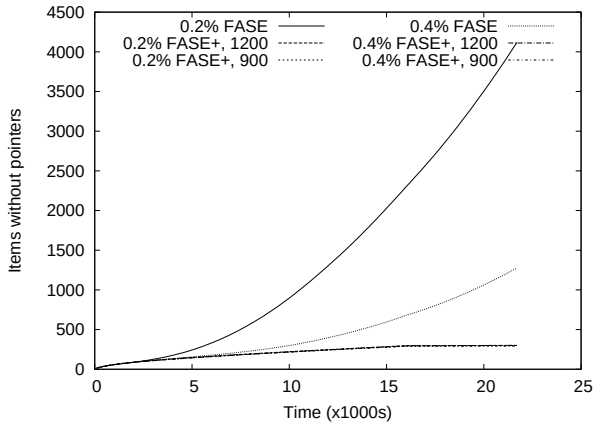


(a) Evolution of the number of pointers.

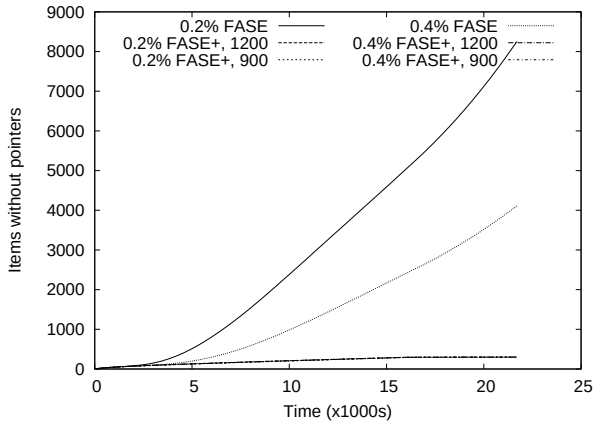


(b) Evolution of the number of backups.

Figure 8: Pointers and backups (FlatOut).



(a) Lost pointers in the RedHat set.



(b) Lost pointers in the FlatOut set.

Figure 9: Live items that lost all pointers.

and 78% hops for, respectively, 0.2% and 0.4% replication rates. FASE+ shows a significant improvement in the hop limit, about 72% less for 0.2% replication and 70% for 0.4% replication. Percentages for the FlatOut plot were similar. As FASE+ is able to maintain the number of pointers defined by the replication ratio, the queries take less time to find a key than in FASE, that loses pointers as an item age increases.

In the second half of the querying phase, FASE starts losing pointers of the old items while FASE+ mitigates that loss. This is confirmed by the following figure, which plot the average hop count with the update interval set to 1200s. Figure 11 show the results of queries over time for the FlatOut set. While new items are being announced, FASE+ maintains the hop count values even when it increases in FASE; in the second half, where no new items are advertised and therefore, queries are only addressed to old items, FASE shows an increase while FASE+ is able to maintain the hop count. The FlatOut distribution has less nodes with very short session times and less nodes with very long session times, and presents higher hop counts when compared with the other sets. However, FASE+ is able to maintain the hop count during both halves of the querying phase.

6.4 Comparison with SQR

We implemented the Scalable Query Routing (SQR) algorithm (see Sec. 2), to be used as a comparison to FASE. SQR was chosen because it is an algorithm which aims at efficiency and scalability in unstructured peer-to-peer networks using a distinct approach from FASE. While FASE approach consists in dividing the search space and increasing the probability of an item to be found in a specific set of nodes, the SQR algorithm uses routing tables (which have to be maintained) at each node to guide query forwarding. This allows the comparison of these two different strategies in maintaining a distributed index.

One could argue that, while FASE does not have the advantage of SQR, since SQRs probabilistic routing tables provide hints to where the queries should be directed, the pointer replication mechanism gives a better advantage to FASE in finding rare items. For that reason, test cases where SQRs contents are replicated will be presented for replication rates from 0.04% (4 items) to 0.4% (40 items).

Due to memory constraints imposed by its routing tables, SQR was evaluated only up to $degree = 10$ topologies. SQR's parameters d (the filter decay rate) and k (the number of hash functions) were selected based on the values used in [7] to achieve a low cost for the maintenance of the routing tables: $d = 8$, $m = 40960$ and $k = 32$; the m parameter (the EDBF length) is lower than the value used in [7] as we used

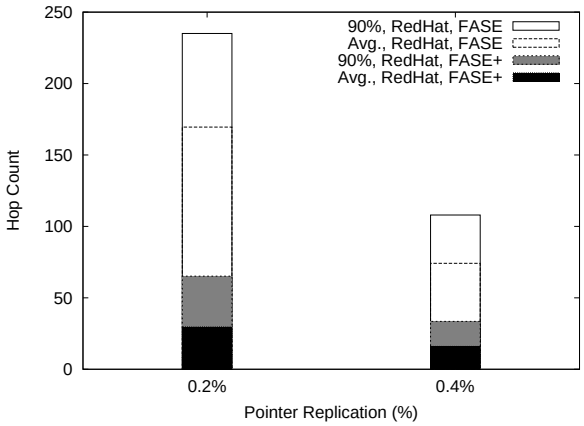


Figure 10: Avg. hop count and hop limit.

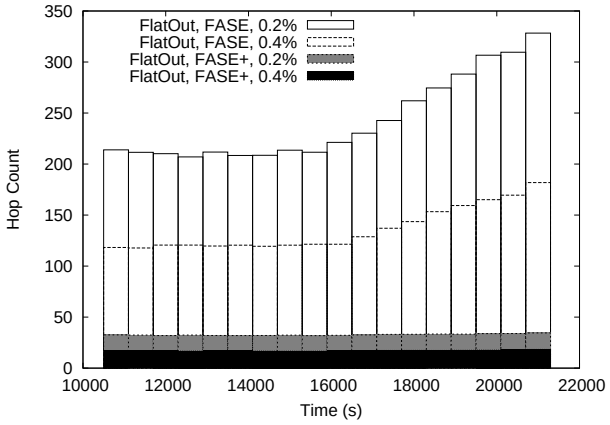


Figure 11: Query hop count vs. time (FlatOut)

substantially less objects.

Figure 12 compares FASE and SQR average hop count. SQR is not suited to search for rare items: the average of the hop count for finding a unique item, the value for 0.01% replication ratio, is of 1360. As there is enough information to route the query only at a close distance from the item, depending from the decaying factor, the queries are forwarded in a random fashion most of the time. This results seems to point that replication may be more adequate than building routing tables as a strategy to increase the performance of unstructured overlays. We tested SQR with its contents replicated with the same ratios observed in FASE pointer replication. With one random walk and without the benefit of routing information, FASE results are close to SQRs. This proves the benefit of the division of the search space in multiple frequencies. Since the replication is equal, this division is the only leverage of FASE in this case. The hop limit for 90% follows the same pattern of Fig. 12.

SQR does not perform well in overlays subject to churn. Like in the stable network scenario, we first tested SQR with a unique item and set the interval of filter dissemination to 60s. Then we compared SQR to FASE+ with a replication ratio of 0.2%, using a single random walk and an update interval of 1200s in FASE+. Table 1 presents the results for the RedHat set. Without replication the failure ratio, that

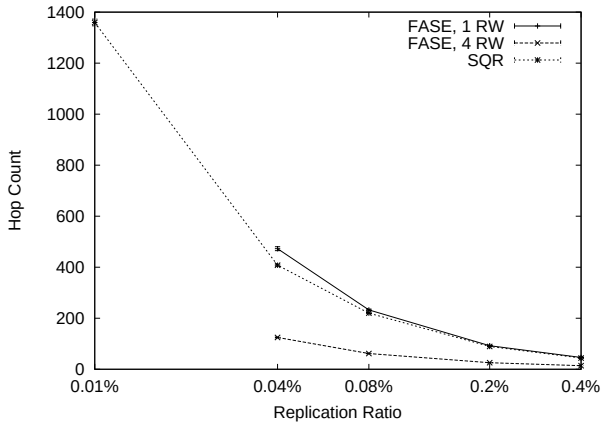


Figure 12: Hop count for FASE and SQR.

Table 1: FASE+ and SQR subject to churn.

	Hop Count	Failure Ratio
FASE+ (0.2%, 1 random walk)	108	0
SQR (no replication)	2227	0.0560
SQR (0.2%)	1584	0.001

is, the number of queries that exceed their TTL (set to the size of the network) is above 50%. Churn emphasizes the random behavior seen in the stable simulations, leading to query hop counts that exceed the TTL. With 0.2% replication SQR performance increases, but it is significantly lower than in stable conditions (from, approximately, 120 to 1580 hops); FASE+ shows a performance under churn as good as SQR in a stable network, in the same conditions.

7. CONCLUSION

An autonomic system that has to operate in an heterogeneous and distributed (and potentially unreliable) environment can leverage on a peer-to-peer system to manage data and/or resources. This paper presented FASE, a protocol to manage a distributed index over unstructured, non-hierarchical overlays. FASE uniformly partitions the nodes and the keys used to locate resources in a common frequency space. The keys are replicated in the nodes that share their frequency. Queries are mappable into the frequencies and directed to the corresponding nodes. To mitigate the effect of churn and node failures in FASE we implemented a replica monitoring algorithm.

Evaluation showed that FASE can achieve low hop counts when locating resources in overlays where every participant contributes with similar resources. The efficiency of FASE depends on the balance of the cost of replication, which happens once in the lifetime of an item, and search. If an item is to be searched for often, the replication costs will be progressively lower, as the existence of more pointers, which implies a higher replication cost, reduces the cost of every query. Additionally, it was proved that the monitoring algorithm sustains FASE performance for different rates of churn and that, when the overlay is subjected to churn, FASE performs better than SQR.

8. REFERENCES

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pages 407–418, New York, NY, USA, 2003. ACM Press.

[3] V. Cholvi, P. Felber, and E. Biersack. Efficient Search in Unstructured Peer-to-Peer Networks. In *SPAA '04: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 271–272, New York, NY, USA, 2004. ACM.

[4] Clip2. The Gnutella Protocol Specification v0.4, 2001. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`.

[5] P. Fonseca. Search strategies in unstructured overlays. Master's thesis, University of Lisbon, 2008.

[6] P. Garbacki, D. H. J. Epema, and M. van Steen. Optimizing Peer Relationships in a Super-Peer Network. In *27th International Conference on Distributed Computing Systems (ICDCS 2007)*, Toronto, Canada, June 2007.

[7] A. Kumar, J. Xu, and E. Zegura. Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks. *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, 2:1162–1173, 2005.

[8] J. Leitão, J. Pereira, and L. Rodrigues. Hyparview: a membership protocol for reliable gossip-based broadcast. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 419–429, Edinburgh, UK, June 2007.

[9] J. Liang, R. Kumar, and K. Ross. The Kazaa Overlay: A Measurement Study. *Computer Networks (Special Issue on Overlays)*, 2005.

[10] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *ICS '02: Proceedings of the 16th International Conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[11] G. P. J. Márk Jelasity, Alberto Montresor. PeerSim: A Peer-to-Peer Simulator, -. `http://peersim.sourceforge.net`.

[12] Y. Qiao and F. E. Bustamante. Structured and unstructured overlays under the microscope: a measurement-based view of two p2p systems that people use. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 31–31, Berkeley, CA, USA, 2006. USENIX Association.

[13] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proceedings of USENIX'04 Annual Technical Conference*, 2004.

[14] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking (MMCN '02)*, 2002.

[15] K. Sripanidkulchai, B. M. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *INFOCOM*, 2003.

[16] 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, New York, NY, USA, 2006. ACM.