Self-Organization in Peer-to-Peer Systems

Jonathan Ledlie, Jacob M. Taylor, Laura Serban, Margo Seltzer
Harvard University
33 Oxford St., Cambridge MA., USA
{jonathan,margo}@eecs.harvard.edu, {jmtaylor,serban}@fas.harvard.edu

Abstract

This paper addresses the problem of forming groups in peer-to-peer (P2P) systems and examines what dependability means in decentralized distributed systems. Much of the literature in this field assumes that the participants form a local picture of global state, yet little research has been done discussing how this state remains stable as nodes enter and leave the system. We assume that nodes remain in the system long enough to benefit from retaining state, but not sufficiently long that the dynamic nature of the problem can be ignored. We look at the components that describe a system's dependability and argue that next-generation decentralized systems must explicitly delineate the information dispersal mechanisms (e.g., probe, event-driven, broadcast), the capabilities assumed about constituent nodes (bandwidth, uptime, re-entry distributions), and distribution of information demands (needles in a haystack vs. hay in a haystack [13]). We evaluate two systems based on these criteria: Chord [22] and a heterogeneous-node hierarchical grouping scheme [11]. The former gives a $>1\%$ failed request rate under normal P2P conditions and a prototype of the latter a similar rate under more strenuous conditions with an order of magnitude more organizational messages. This analysis suggests several methods to greatly improve the prototype.

1. Introduction

Large-scale decentralized distributed systems — peer-to-peer, ubiquitous, or sensor network systems with multiple millions of nodes — are just beyond their infancy and have not yet had dependability quantified in a consistent manner. Many P2P projects are in their research phases and only a handful are in common use. Most of these designs work well when the rate at which nodes enter and exit the system is small, but none that we have seen explicitly discuss the design tradeoffs between the type of information exchange for which the system is designed, the physical characteristics of the constituent nodes (*e.g.,* mean time to failure), and how reliable information exchange needs to be in order to fulfill the system's goals. Before more large-scale decentralized distributed systems are designed and built, the community needs to reach agreement on the meaning of acceptable and unacceptable functionality under a variety of dynamic conditions representative of P2P systems.

The intuitive notion of "dependability" for these systems is one of *reachability of information*. Accordingly, dependability should be measured by the percentage of times that a request results in the proper information moving from its source(s) to its destination(s). The requirements for dependability vary greatly within the parameter space of P2P systems. Consider a point-to-point system designed to answer existence queries. An instance where every node has a completely up-to-date and accurate picture of the rest of the system and where the bandwidth consumed by queries and state transfer does not exceed the capacity of any links would be perfectly dependable. However, such a design might not work if the type of information exchanged was event-driven: if, for example, one node needed to notify another node or collection of nodes when there was an abrupt temperature change or if a bridge were about to collapse.

In this paper, we define dependability in P2P systems and discuss the way in which Chord and our own hierarchically grouped system self-organize to overcome the unreliability of nodes that comprise the system.

2. Reliability in Decentralized Systems

A system's dependability is defined in terms of three characteristics: the type and method of information exchange (e.g., probes, point-to-point streams, broadcast streams, etc.), the individual nodes' capabilities, and the distribution of data and queries among the nodes. One thread links all three components: local information must provide a quantifiable and probabilistically accurate depiction of the global state. The required level of this accuracy depends on system usage; increased tolerance for out of date local information leads to diminished state, messages, bandwidth, and uptime requirements. For example, in Chord, the likelihood that requests will be fulfill-able depends on the join/failure rate and on the rate at which a node ring stabilization procedure is run, which in turn depends on the node's capacity for topology messages.

The first component to the overall dependability of a decentralized system is the type of information exchanged across it. We divide information exchange into the following five categories:

probe Probe queries test for the existence of an object. These queries often use a filter structure (*e.g.,* DHTs or Bloom filters) or resource intensive naive broadcast queries (*e.g.,* Gnutella [8], Freenet [5]); the latter gives the significant advantage of high tolerance against node failure and allows for receiver-interpreted queries.

event-driven point-to-point A node registers an interest and is contacted when something matching this interest enters the system. Examples include abrupt temperature change, sensor aggregators [9], change in file contents, file creation, new authorship, and distributed triggers[3].

event-driven broadcast This is a broadcast from one node to all other nodes, used to distribute information globally. This could be used, for example, to implement a software update.

continuous stream point-to-point This exchange provides a path for streaming data for an indeterminate duration to another node or other nodes. Internet routing is one such example. The requirement of continuity may mean pro-active measures against unknown failures will be necessary (*e.g.* using multiple paths), compared with just post-failure cleanup and recovery.

continuous stream broadcast One node continuously updates the entire system, similar to continuous stream point-to-point. The ubiquitous nature of this type of exchange may make it much easier to implement in P2P systems without pro-active routine measures.

Most P2P designs focus on probe queries while sensor network systems fall into one of the remaining four classes. That said, one could imagine other systems where P2P systems support event-based queries and sensor networks use probes. Regardless, the categories of requests must be considered when defining the system's local state.

The nodes' capabilities are the second characteristic that defines a system's dependability. Liben-Nowell *et al.* introduced the idea of the half-life of a system as the amount of time it takes for half of the nodes to exit [12]. One can generalize this concept to include the probability of the node returning to the system and the length of time between a node's exiting and returning. Most of the current crop of popular P2P research systems [6, 7] were designed with a large-scale static system in mind, and do not perform well under high volatility. Unfortunately, this high volatility is the norm: a study of Mojo Nation found that 80% of the nodes exist in the system for less than one hour [25]. If they are to be deployed on dynamic P2P networks or on battery-conscious sensor networks, new designs need to incorporate the distribution of node join and return from the beginning.

The third component of dependability is the distribution of information and requests across the nodes. Adar and Huberman [1] and Ripeanu [18] show that a small percentage of files make up the bulk of the queries and files in Gnutella, and that the distribution of searches are heavy-tailed. Lv *et al.* [13] use the correlation between file and query distributions to explain why Gnutella has not collapsed under the weight of its naive, flooding-based probe scheme, as Ritter predicted [19]. The simulation studies of this paper assume a uniform distribution of files and file requests, which is probably not appropriate for the Gnutella world. However, it provides an upper-bound on the query failure rate for a correlated system. Also, Gnutella-like file correlations are clearly inappropriate for a library citation, DNS, or similar environment where users search for both hay and needles in a haystack of information [13]. We see that it is essential to consider the distribution of both data and query elements in order to evaluate the system's dependability.

Terminology from the fault tolerant community can be misleading when applied to distributed decentralized systems. Mean-time-to-failure here refers to the mean-time-to-node-departure. Mean-time-to-data-loss has less meaning when nodes are always entering and exiting the system; queries always have a significant chance of failing under realistic conditions. For P2P filesharing systems we can define mean-time-to-query-failure (MTQF). More generally, the dependability can be quantified by the mean-time-to-request-failure (MTRF), which allows for all five categories of information exchange to be considered.

For Chord and the hierarchical grouping system we describe below, we assume exact probe searches on a uniform distribution of files present among live nodes in the system: only files that currently exist are searched for. We assume that nodes are not overburdened with other activities, *e.g.,* actually moving data. In the Chord experiments, we assume a uniformity of node capabilities — somewhat realistic in a sensor network with uniform components, but unrealistic in a P2P setting. In the hierarchical grouping simulator, nodes can have both uniform and heterogeneous characteristics, and we present results for both of these situations. These assessments are clearly only the beginning to a long series of possible evaluations. It seems unlikely that one system will work well under the whole range of P2P parameters; designers must explicitly state their target node and data audience and then evaluate dependability accordingly.

3. Topology of Hierarchical Groups

We have designed and implemented a simulator for a hierarchical grouping scheme which is designed for P2P and sensor network systems. In earlier work, we evaluated a prototype of its search mechanism [11], but its grouping mechanism has not been previously described. For completeness, we present both here. We refer the reader to other papers [22, 23] for an introduction to Chord.

3.1. Search Overview

We begin by describing how the search system works and then move into details of how the system self-configures. A system consists of many hierarchical groups, each shaped as a tree. Every group has a root node. The root is responsible for:

1. Calculating a summary of all objects in the group.
2. Maintaining summaries for each of its immediate children (which in turn maintain summaries for their children).
3. Directing searches of the group.

Summaries are represented as Bloom filters [4], whose size is computed by the root. Bloom filters are bit strings whose size is proportional to the number of objects they summarize [16]. All bits are initialized to zero; the addition of each object to the filter sets "on" the bits signaled by several hash functions for which the object being added is the input. Bits that are already set remain on. To probe a filter for a match, the same hash functions are performed and the bit array is checked: if all of the bits are set, the filter matches. Bloom filters only give false positives, not false negatives. Nodes underneath the root are arranged in a $k-$tree structure with $\log_k n$ nodes at level n. Nodes communicate with their children, their parent, with the root of their group, and with a dynamically changing collection of extra-group nodes.

Given this configuration, a search originating at node N, proceeds as follows:

1. Consult the group summary filter of node N. If the filter indicates that the object could exist in the tree, iterate over all possible children that might contain the object (using the child filters stored at N). If the object is found in any child, the search concludes successfully.
2. If the object is not found in the current tree, node N passes the query to root(N). Root(N) conducts a search on its descendants (pruning the part of the tree already searched by N).
3. If root(N) fails to locate the object in its current tree, it sends the request to any groups whose filter indicates that the object could reside there.
4. Each group queried takes one of three actions:
   (a) If the current group filter indicates that the object cannot be in the group, the group responds with the new group filter.
   (b) If the object isn't in the group, respond with a NACK and some suggested groups that might be queried (that is, consult any other group summaries present and for any potential hits, tell the initiating group of the potential hit).
   (c) Return the location of the node that has the object in the group.

3.2. Hierarchy Structure

We define the ideal topology as a collection of groups of nodes, where the nodes in a group are related based on low intra-group latency and varied mean-time-to-failure (MTTF), and heterogeneous bandwidth. Our goal is to come as close to this ideal topology as possible using only local information.

Nodes benefit from being in a group because they share information about other groups, so that when node a in G_1 receives information about G_2, that information is accessible to all other nodes in G_1, because G_1's root caches G_2's filter. These benefits increase as groups grow in size. Nodes also benefit from the existence of other groups, because transmitted group summaries serve as an efficient mechanism to prune the search space.

Larger groups provide more shared information, but this benefit is offset by the cost of keeping the group reasonably balanced, maintaining group summaries, and the load on the root. The root's workload grows with the size of the group as it will broker all group searches, maintain group and child summaries, control entry to the group and determine the time for partitioning of the group.

It is this last responsibility, determining partition time, that makes the system feasible. When the root becomes overloaded, it sheds load by partitioning the group. This partitioning, in conjunction with responding to requests to join the group, is what provides the dynamism and self-configurability of the system.

Several other projects have proposed "supernodes" as the solution to the heterogeneity empirically extant in P2P systems. Saroiu *et al.* have shown that there are multiple, distinct categories of nodes, ranging from always-on high-bandwidth nodes to 56k modems only connected for an hour or less [20]. Hierarchies form a good extension to the "supernodes" currently proposed in several research projects (*e.g.,* Gnutella++ [8], Brocade [26]). In these projects, there are two levels of nodes: "supernodes" that do most of the routing, and regular nodes. A more general heterogeneous system should use a heterogeneous topology, with "better" nodes living closer to the root of each group.

3.3. Node Entry

1. When a node x enters the system, it immediately becomes its own group G_x. G_x forms a list of credentials, including its bandwidth capabilities and its number of public files.

2. Node x contacts a well-known location, called a "node cacher", to find other nodes S in the system (similar to Gnutella and Mojo Nation). This bootstrapping component only keeps a list of other nodes that have recently contacted it, also trying to find other nodes in the system. Because this list is the only state it contains, it can be easily replicated, and can pop in and out of existence.

3. Using the nodes S that the new node learns about from the "node cacher," G_x forwards its credentials to the groups containing $s \in S$, by sending messages to each s, which then forward this information to their roots.

4. Each root that considers G_x valid for entry respond to G_x with its credentials. G_x chooses which group to join by picking the one with the best credentials. If this group agrees, G_x then merges with this group. If it refuses, G_x tries another group.

3.4. Node Exit

We have experimented with two designs for keeping the descendants of a node part of a group when a node exits. In one mechanism, nodes try to maintain knowledge of their siblings and grandparents. This information is sufficient to elect a new leader to take the place of the missing parent and then contact the grandparent to inform it of the new topology, including the summary filter change. The other, lazy mechanism just drops children from a group when their parent dies. This expends fewer topology messages and is the one we use in the simulation.

3.5. Summary Propagation

The root determines the group filter size n by using the number of objects in the tree to estimate how many bits are required to produce a Bloom filter with approximately half of the bits set [16]. The lowest leaf nodes generate filters of size n bits by hashing on their stored objects, turning on these bits in their filters, and sending their filters to their parents. These internal nodes also perform their hashing modulo n bits and logically *OR* their filters together with their children's filters, and pass these filters up the tree. Finally, the root node will have the summary of all of the objects in the group, with ideally about half of the bits in its filter set (more than half-full filters tend to give many false positives). The hierarchy of filters also makes it likely that more bits are set higher up in the tree and, conversely, that filters are more sparse — and therefore more accurate and more compressible where bandwidth is less — as searches wind down the tree. This information hierarchy is used both outside the group to determine whether to contact the group at all and within to better direct queries between nodes.

As nodes join and expire from the network, we have them self-form into a communication- and information-based hierarchy based on local information. With global information, nodes would form themselves into ideal groups. With local information, however, they will form themselves into a close approximation of this ideal, forming a continuously maintained "almost-ideal" state.

4. Grouping Analysis

We base our grouping model on natural systems that exhibit self-configuration, driven by particle interactions that lower energy costs when an organized state is realized. Evolution models [2], non-equilibrium phase transitions [24], and crystal facet structure formation [15] among others, all show this behavior, and these ideas have been widely applied to a number of economics and engineering problems.

We derive a node's cost based on its bandwidth consumption, though a refinement to include latency would be a natural extension of this method. To make grouping decisions, nodes compare their current cost with the cost of being in the other groups of which they are aware. If, by forming a group, two nodes can lower the number of queries they receive and the efficiency of the queries they generate, then a group configuration is more desirable. However, there is an activation cost to form a new group, comprised of the one time cost of distributing filters and reorganizing the tree. If groups only form when the cost of the old state exceeds the sum of the activation cost and the new state's cost, we can encourage stability. Having groups flit in and out of existence is expensive and is mitigated by this activation cost.

As noted above, the overall cost that each node seeks to minimize is the weighted sum of the bandwidth costs. Bandwidth usage consists primarily of queries and filter updates. We assume nodes have poor knowledge of the system outside their own group, making query estimates difficult. The only reliable computation nodes can perform with regards to the costs outlined above are those local to the group, that is, specific to the filters. We set the individual group filter cost to the fraction of bandwidth consumed by filter messages to total bandwidth:

$$C_f(g) = \frac{f_{msg}(g)}{\tau(g)} \frac{f_{size}(g)}{BW(g)}$$

where $f_{msg}(g)$ is the number of filter messages sent out over the time $\tau(g)$ with average size $f_{size}(g)$, and $BW(g)$ is the total bandwidth of the group. A more sophisticated method of estimating the filter message rate involving a weighted average favoring near-time events would be a natural extension of this model.

The estimate of the combined cost is then

$$C_f(g_1 \cup g_2) = \left(\frac{f_{msg}(g_1)}{\tau(g_1)} + \frac{f_{msg}(g_2)}{\tau(g_2)}\right) \frac{f_{size}(g_1 \cup g_2)}{BW(g_1 \cup g_2)}.$$

From this we can guess that the activation cost should go as

$$H = \frac{\text{size}(\text{g}_1 \cup \text{g}_2) f_{size}(g_1 \cup g_2)}{t_{\text{filterdist.}}(g_1 \cup g_2) BW(g_1 \cup g_2)}.$$

where $t_{\text{filterdist.}}$ is the average time to redistribute a filter. Assuming these factors are uniform for groups of a given size, the activation cost sets the cost scale of the system.

We can also deduce simple information about the effectiveness of searches from the local group information, such as using connectivity information to determine the quality of searches. We define the search quality factor to be

$$Q_s = a(\text{size}(g_1 \cup g_2)) + b(\text{known nodes}(g_1 \cup g_2))$$

with a and b proportional to the inverse of the number of nodes in the system. In the high bandwidth limit we set a to zero and b to the inverse of the total number of nodes in the system. Then Q_s goes to one when a group is connected to the entire system. However, as we may expect many duplicate files, the advantage of having many members in one's own group and not just connected may be significant; this is represented by non-zero a.

The combined cost function is then

$$\text{Cost} = \alpha C_f + \beta / Q_s$$

with the additional constraint of a hard wall in bandwidth usage, that is

$$C_f < \epsilon$$

where $\epsilon \ll 1$ for most P2P systems where bandwidth should be allocated for file-transfer. The parameters for group formation are then α, β, $\gamma = a/b$, and ϵ. In the limit of $\alpha \gg \beta$, bandwidth consumption is minimized, while $\beta \gg \alpha$ (but reasonable ϵ) allows for quality of search maximization constrained by reasonable bandwidth consumption. Small γ corresponds to an emphasis on highly connected groups, while large γ should tend to favor large groups.

4.1. Analysis of Summary Filters

We present a brief analysis of the probability of a false positive for a file not in the system, or the *false positive rate*. As described above, each root node in the system maintains an up-to-date Bloom filter representing the files in its group. In addition, root nodes acquire the aggregate filters of other groups. Let g be the number of groups in the system, n the number of distinct files per group, b the number of bits per file used and k the number of independent hash functions used in the Bloom filter data structure. Assuming that hash functions are perfectly random, the theoretical probability of a false positive for a file not in the system, or the system's false positive rate is:

$$g\left(1 - (1 - 1/nb)^{kn}\right)^k \approx g\left(1 - \exp(-k/b)\right)^k$$

where $p_s = \exp(-k/b)$ is the probability that a specific bit in any of the aggregate bloom filters is still 0. Note that given g, b and n, the number of hash functions k can be optimized to minimize that false positive rate. Namely, taking $k = b(\ln 2)$ yields an optimal false positive rate of $f_s = g(1/2)^k = g\,(0.6185)^b$.

The false positive rate derived above corresponds to a theoretical upper bound on the fraction of groups contacted per search. Since the factor $(1 - p_s)^k$ decreases exponentially with b, the number of bits allocated per file, for optimal number of hash functions k, so does the fraction of redundant search messages between groups.

One potential source for concern in our design is whether the root nodes will have the capacity to store and keep updated filters of even a small fraction of the rest of the system. A rough calculation shows this is possible. If we assume that nodes on average store $100 \approx 2^7$ files (as they do in Gnutella[1]), and that aggregate filters are built using $8 = 2^3$ bits per file (giving a false positive rate of $\approx 2\%$), with 1000 nodes, storing all of the filters takes $2^{10} \times 2^7 \times 2^3 = 2^{20}$ bits $=$ 128 kilobytes of storage and 1 million $\approx 2^{20}$ nodes consumes 128 megabytes, still not an unreasonable amount of storage. Of course, as has been noted above, roots are not required to cache all or even most other groups' filters, so the actual amount stored is only a fraction of these values.

This hierarchical use of Bloom filters is different from the attenuated Bloom filters used in OceanStore [17, 10]. Its usage of using a combined filter to describe a distinct subgroup of neighboring nodes is similar the logical *OR* presented here. However, OceanStore does not have the concept of hierarchies of filters with increasing numbers of bits set or of representative group filters.

4.2. Compressed Bloom Filters

Large sparse Bloom Filters can be greatly compressed. Theoretically, an m-bit filter can be compressed to $mH(p)$ bits where p is the probability that a bit in the filter is 0 and $H(p) = -p \log_2 p - (1-p) \log_2 1 - p$ is the entropy function. For sufficiently large filters, arithmetic coding guarantees close to optimal compression, so if p is small enough,

$H(p)$ is much smaller than 1, and significant savings in the transmission size can be achieved[14].

The bulk of the update messages consist of sparse filters. In particular, both filters of the nodes in the lower levels of the tree hierarchy of a group and update filters are sparse. Updates to filters can be sent as deltas: indices of which bits to turn on or off, instead of sending the whole filter. For a relatively balanced tree a node at level $i-1$ has approximately $2^{\log_2 n - i} \approx \frac{n}{2^i}$ subnodes, therefore assuming that each node has roughly the same number of distinct files, the sparseness of the child filters of a node in the tree increases exponentially with the level of the node in the tree.

We have presented a brief description of our P2P system which uses hierarchical groups to prune searches and take advantage of node heterogeneity to improve system stability. Now we return to looking at reliability in Chord and in our system.

5. Reliability in Chord

We evaluated Chord by determining how it mapped into our three characteristics of decentralized system reliability and by modifying a pre-existing simulator [21] to test for these characteristics.

Chord, in its current form, supports probe (existence) queries: if an object exists in the system, it (hopefully) returns a pointer to the node storing that object. Although the primary Chord papers [23, 6] do not explicitly discuss node characteristics, it is designed to run on a P2P network, inferring that the nodes exhibit the previously outlined characteristics [20, 18, 1, 25], in particular, that average node lifetime is on the order of one to several hours. As previously mentioned, the objects chosen to query are chosen randomly from those currently existing on live nodes.

We modified a Chord simulator to count the number of messages used for search, object relocation, and stabilization. The stabilization process provides a mechanism for nodes to confirm they are the predecessors of their successors and to repair their finger and successor tables. This operation keeps the ring intact and the more frequently nodes join and exit, the more frequently `stabilize` needs to be run to keep failure rates level.

We show the results from two sets of experiments in Figures 1 and 2. Both experiments were run with 1000 nodes performing one search per minute on average (all average events for the Chord simulator follow a Poisson distribution). The number of fingers was 40 and the number of successors was 20. The average message delay for Chord and for hierarchical groups was 50 milliseconds. Both figures average five separate experiments (errorbars were omitted from Figure 2 for visual clarity). Figure 1 shows the gradual improvement in the failed lookup rate as the average lifetime increases from 15 minutes to 3 hours. The stabilize

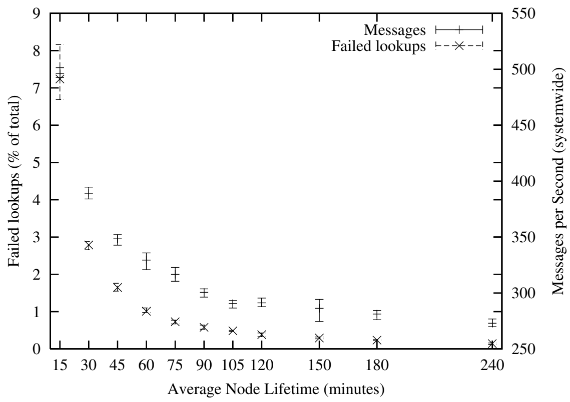

Figure 1. Effect of varying average lifetimes in Chord. Stabilization rate is 2 msg per min per node.

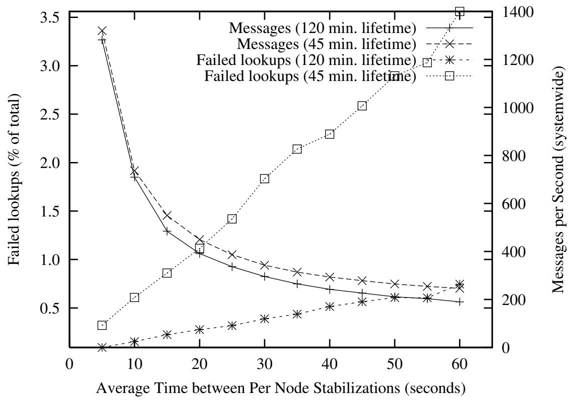

Figure 2. Effect of rate at which stabilizing process is run in Chord.

procedure runs once every 30 seconds per node on average, updating two pointers per run: the successor and a randomly chosen other node from the successor table, with nearby successors chosen with higher probability. This appears to be the default used in most of the experiments in the Technical Report [23], except the one on lookup failure, where the simulator updates all finger entries on every invocation. Each experiment ran for 3 or more hours of virtual time; we found that shorter experiments yielded results with higher variability.

Figure 2 shows Chord as the rate of stabilization changes with average lifetimes of 45 minutes and 120 minutes. Increasing message rates to ~ 1.25 messages per node per second allows a reduction in failed lookups to $< 0.4\%$.

This means from a reliability standpoint that reaching the levels of availability expected by most file system users, say

$> 99.99\%$, would be difficult in Chord unless (a) all nodes exhibit higher than previously observed levels of uptime or (b) all nodes have the bandwidth capacity to run stabilize many times per second. The preferential availability of some data of others is impossible in Chord due to its DHT design; of course, using a higher layer for redundancy would partially ameliorate the availability problem (as CFS does [6]).

6. Reliability in Hierarchical Groups

We have designed and implemented a simulator prototype whose nodes follow the steps outlined in Sections 3 and 4 to form groups and perform searches. Although we ran tests with larger numbers of nodes, we present results with 1000 nodes up on average and with nodes performing one search per minute on average. Because the parameter space for hierarchical groups is large (there are more than 20 separate parameters once we include different possible network configurations), we have been limited in the variety of experiments we could examine prior to this publication.

The primary simplification made in the current simulator is that groups are formed only virtually. That is, nodes do not walk a branch of the tree and reach a final destination; instead, they all exist directly under the root. The death of a node, however, dislocates nodes following a distribution which is based upon failure of nodes in a tree shaped topology with $\log_2 n$ nodes at level n, such that approximately half of the failures are assumed to be leaf failures, then half of those remaining parents with one child, an so on. This has prevented us from evaluating the effectiveness of intragroup filters and seeing the benefits resulting from the compressed bloom filters at the bottom of the tree.

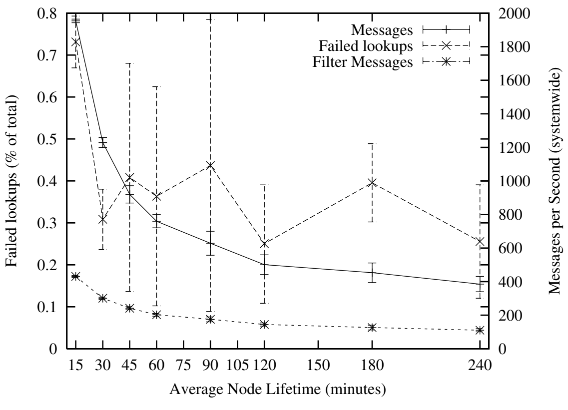

Figure 3. Effect of varying average lifetimes in Hierarchical Groups.

In our simulations we set $\alpha = 0, \beta = 1$, and $\gamma = 1/2$, with a maximum filter-message bandwidth usage $\epsilon = 0.1$ and no activation cost. We hope to emphasize quality of searches over bandwidth usage beyond the hard limit of ϵ. In the long-lifetime limit (average node uptimes beyond ten hours) within 10 minutes a 1000 node system stabilized to a better than a 0.1 % search failure system rate. We also ran four simulations per average lifetime using a uniform distribution of nodes with average lifetimes between 15 minutes and 4 hours and 56k modem-like bandwidths. Filter sizes were set to 5 bits per file to minimize bandwidth consumption for filters. One node-cacher was used, disseminating five nodes to every joining node. These results are shown in Figure 3. We see a much lower failure rate and higher message rate than Chord for similar node lifetimes, doing better than 1% even with average lifetimes of 15 minutes. The large fluctuation of results is most likely due to digitization effects from small node number near 0.1% failure rates. Comparing to Figure 2, we see that a similar number of messages are required for both simulators to get similar failure rates at 45 minutes. However, the bulk of these messages are group join queries, followed by filter messages and then all others (search, other intra-group), and that of these the vast majority are generated by node failure leading to child nodes regrouping. In particular, a $k-$tree leads to deaths creating order $\log_k \text{size}(g)/k$ times the number of nodes disseminated by the node-cacher messages. For the simulations with average lifetimes of 45 minutes, this corresponds to $\sim$ 600 msg/min. We expect that widening the tree or adding grandparent to child links as discussed previously should mitigate this cost. The extra links would require two additional messages per join and k additional messages per death, which is $\sim$ 120 msg/min for an average lifetime of 45 minutes and 1000 nodes. Then, the filter message rates as shown in Figure 3 should dominate.

We also ran simulations with a subset of nodes with high-bandwidth and long-uptimes, as might be expected in an actual network, to look for the effects of these nodes in the trees. They were given 10 times the bandwidth and lifetime of the regular nodes. However, with only 5% of nodes the high-bandwidth type, we see little improvement at the level of 1000 nodes, though larger systems may demonstrate this more effectively. Furthermore, as group rearrangement and root replacement have yet to be implemented, the benefit of these extra nodes may be marginal.

By introducing these refinements and in addition considering sparse filter compression, we can make the limiting factor for group overhead filter messages, and reduce the current bandwidth consumption by filters by a factor of 80% for groups of size 200 and 96% for groups of size 1000. Thus we can scale from a maximum group size of 200 in our simulation for $\epsilon = 0.1$ and MTTF of 45 minutes to well beyond that for even smaller ϵ. It seems that root node overusage will become the limiting factor for large numbers of nodes. As shown earlier, this will allow for scaling the system to order 10^6 nodes.

7. Conclusion

This paper makes three contributions. First, we examine how the implicit goals and assumptions about a particular decentralized system affect measures of its reliability. Second, we introduce a self-organizing hierarchically-based P2P system. Third, we take the assumptions implicit in current P2P filesharing systems and evaluate the reliability of Chord and the hierarchical grouping system. In simulation experiments, both systems perform adequately as long as there exist a $0.5 - 3\%$ tolerance for failure under normal conditions. This failure rate is probably acceptable for a file sharing situation but would need to be tampered by a higher-level application that would provide redundancy in more rigorous file system-like scenarios. Both systems utilize self-configuration — `stabilize` and local-information-based group formation — to maintain an adequate degree of reliability even under high fluctuation. In particular, our model enables the formation of local points of stability and high bandwidth, and we show how self-configuration can create many local foci to which the rest of the more dynamic system can attach.

We would like to thank M. Mitzenmacher for helpful discussions.

References

[1] E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 2000.

[2] P. Bak and K. Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71:4083, 1993.

[3] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. The asilomar report on database research. *ACM SIGMOD Record*, December 1998.

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

[5] I. Clarke. Freenet: A distributed anonymous information storage and retrieval system. `http://freenetproject.org/cgi-bin/twiki/view/Main/ICSI`, 2001.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001.

[7] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001.

[8] Gnutella. Gnutella protocol specification v0.4. `http://www.clip2.com/GnutellaProtocol04.pdf`, 2001.

[9] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001.

[10] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[11] J. Ledlie, L. Serban, and D. Toncheva. Scaling filename queries in a large-scale distributed file system. Research Report TR-03-02, Harvard University, January 2002.

[12] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Observations on the dynamic evolution of peer-to-peer networks. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.

[13] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make Gnutella scalable? In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.

[14] M. Mitzenmacher. Compressed Bloom filters. In *Twentieth ACM Symposium on Principles of Distributed Computing (PODC 2001)*, 2001.

[15] C. Perez, A. Corral, A. Diaz-Guilera, K. Christensen, and A. Arenas. On self-organized criticality and synchronization in lattice models of coupled dynamical systems. *International Journal of Modern Physics B*, 10:1111, 1996.

[16] M. Ramakrishna. Practical performance of Bloom filters and parallel free-text searching. *Communications of the ACM*, 32(10):1237–1239, October 1989.

[17] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *INFOCOM 2002*, January 2002.

[18] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of International Conference on Peer-to-peer Computing*, August 2001.

[19] J. Ritter. Why Gnutella can't scale. `http://www.darkridge.com/~jpr5/doc/gnutella.html`, 2001.

[20] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2002.

[21] I. Stoica. Chord simulator. `http://www.fs.net/cvs/sfsnet/simulator/?cvsroot=CFS-CVS`, 2001.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, August 2001.

[23] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Research report, MIT, January 2002.

[24] P. Weichman, A. Harter, and D. Goodstein. Colloquium: Criticality and superfluidity in liquid 4he under nonequilibrium conditions. *Reviews of Modern Physics*, 73:1, 2001.

[25] B. Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.

[26] B. Y. Zhao, Y. Duan, L. Huang, A. D. Joseph, and J. D. Kubiatowicz. Brocade: Landmark routing on overlay networks. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.