Spinneret: A Log Random Substrate for P2P Networks

Jeff Rose[1], Cyrus Hall[1], and Antonio Carzaniga[1,2]

[1]University of Lugano
Faculty of Informatics
6900 Lugano, Switzerland
{jeffrey.rose,cyrus.hall}@lu.unisi.ch
antonio.carzaniga@unisi.ch

[2]University of Colorado
Dept. of Computer Science
Boulder, CO 80309-0430 USA

Abstract

Until now, structured and unstructured networks have been considered in absentia of each other. We believe that next-generation P2P services will require both structured and unstructured algorithms, and that it therefore makes sense to consider a unified substrate that provides good service for both. In this paper we argue for the creation of a semi-structured overlay substrate, called Spinneret, which can serve as the base layer for a variety of structured and unstructured search algorithms. In order to validate that this structure forms a good foundation for various services, we present two algorithms simulated on top of the Spinneret substrate: an unstructured k-walker random walk search as well as a logarithmic DHT search. Further, we argue that such a substrate strikes a balance between the resilience and reliability of unstructured networks and the efficiency of structured networks.

1 Introduction

The initial widely deployed peer-to-peer (P2P) systems used simple network flooding to search for objects on other peers. This was shown to not scale well due to the explosion of the number of messages in typical, well-connected networks. A fundamental improvement over unstructured P2P systems was achieved through the development of *structured* P2P systems commonly referred to as *distributed hash tables* (DHT).

While substantially more efficient in terms of both query latency and bandwidth usage, DHTs are limited to queries with exact matches. Further, most DHTs require that the network maintain a strict set of peer connections, which, like the pointers in an in-memory data structure, must stay valid in order for the data structure to be traversed. Churn, or the constant leaving and joining of nodes in the network, can wreak havoc on an algorithm which requires such a specific link structure [10]. Recent work in P2P structures has focused on improving DHTs in these two directions. For example, randomized graph structures have been proposed instead of rigid structures. These structures have superior resiliency in the face of failure and network churn [5]. Randomized graph structures can also achieve slightly better performance with high probability, routing in $\Theta(\log n / \log \log n)$ hops [8].

Our research starts from the observation that, despite the fundamentally superior theoretical performance of structured P2P systems, unstructured algorithms remain important tools in building most of the widely deployed P2P services. This choice is attributable to the two major shortcomings of structured P2P systems: exact match search and churn vulnerability. Therefore, in this paper we argue for an engineering solution that attempts to combine the best features of structured DHTs with the flexibility and resilience of algorithms like random walks and epidemic flood typical of unstructured P2P systems.

Specifically, we envision *a peer-to-peer substrate* based on a randomized semi-structured graph, amenable to both exact-match queries, using DHT-style routing, and generic queries using random peer sampling. We call this substrate *Spinneret*.

Spinneret is intended to support DHT-style queries by means of a greedy algorithm.In order to support generic queries, we propose a two-pronged approach: First, we propose to develop random-walk algorithms that visit a random sampling of peers. These algorithms provide a basis for

generic (probabilistic) query matching, and can be used individually or in combination with a greedy search. Second, we propose to make the substrate parametric with respect to the distance functions used to identify and compare objects and peers. These functions could embody semantic-rich conditions to be exploited through greedy or randomized routing algorithms.

As for churn and maintenance, we propose to exploit the randomness of the data structure as well as the presence of query traffic to carry out opportunistic repair of the overlay. A similar approach was proposed by Zhang et al. [12] to find latency-minimal routes in a DHT. However, our goal here is to reduce maintenance overhead and to reduce the effect of link failures through proactive repair, even in the presence of dynamic and unstable networks. Finally, we argue that a common peer-to-peer substrate would be beneficial from the viewpoint of the engineering of P2P applications, as it would modularize the maintenance and scheduling of network resources.

In this paper we present our initial effort and results towards the goal of a shared peer-to-peer substrate. As an initial validation of the proposed structure and its provided interface, both a greedy DHT-style search and a k-random walk are implemented on top of Spinneret. Although discussed here, future work will look at network bootstrap and maintenance under churn in more depth.

We continue in Section 2 with a discussion of the techniques that have been applied to both unstructured and structured networks. Next, Section 3 lays out the design goals and the architecture of Spinneret, including a formal description of the network structure and its interface to higher layers. In Section 4 we present the two example protocols, and then finally we conclude in Section 5 with a discussion of ongoing development and directions for future research.

2   Related Work

In order to build a substrate that can be utilized by both unstructured and structured networks, we must start by understanding these two classes of P2P network algorithms.

2.1   Unstructured Networks

In an unstructured network the links maintained by each node are random. A node joins at any point in the graph, and then finds as many neighbors as it wants to maintain connections with. On top of these random structures a number of search techniques have been utilized. In the original Gnutella [6], a node would flood a search message to all of its neighbors, who then forward it on to all of their neighbors and so on, until a time-to-live (TTL) expires. This type of algorithm results in many duplicate messages due

to peers that share neighbors to whom they both forward the same message. Furthermore, superfluous messages continue to be transmitted even after a query has successfully been answered. Improving on this, a number of other algorithms for unstructured networks have been put forward by the community.

The k-walker random walk query strategy [7] sends out k messages, each of which is forwarded to a single random neighbor at each hop. This type of technique has been shown to find objects with the same probability as flooding [3]; however, the total number of network messages can be reduced by one to two orders of magnitude. Such reduced duplication can however lead to longer search latencies because of the additional number of message iterations.

2.2   Structured Networks

Like regular data structures, structured P2P systems must maintain specific links to peers at certain regions in the network. These links allow for deterministic traversal of the distributed structure. Chord [11] uses a set of logarithmic connections around a circular address space, as well as additional connections to immediate neighbors in both directions (successors and predecessors). CAN [9] assigns each node to a region in a multi-dimensional torus, and each peer maintains a connection to neighbors in every dimension. Many other DHTs also provide $O(\log n)$ queries, and neighbor-of-neighbor greedy routing further reduces this to $\Theta(\log n / \log \log n)$ [8].

These algorithms should be used for applications using exact match queries with relatively stable peers; however, in this work our goal is to support massively scalable networks for applications where peers are expected to have high churn rates. In a large network running a DHT, maintenance algorithms can be expensive, requiring $O(n^2)$ or even $O(n^3)$ rounds of communication [11]. In large part because of the difficulty of maintenance, these systems are not widely deployed in active peer-to-peer networks, and as of yet they are unproven in their ability to continue operation under the presence of many transient and unreliable nodes.

Last, and even more important for many applications, these types of algorithms have no ability to handle partial-match queries. Just like a hash table, DHT algorithms require that search queries are expressed by a reference identifier, and for routing it must be in the same address space as that of the nodes themselves. This means that any type of range query, best guess, or fuzzy matching is inappropriate, which greatly limits the applicability for these techniques.

2.3   Improvements

There have been a number of attempts to improve upon the standard flooding scheme, some of which take advan-

tage of the natural structure that arises in a typical network, and some which try to impose additional state or structure.

In some recently deployed systems [4, 6] a concept of super-peers has been introduced, where nodes with higher than average resources form a P2P network that is then utilized by lower class nodes which act as clients. In this work we wish to avoid the constraint of assuming ample quantities of high resource nodes.

Lv et al. [7] show that a power law graph distribution is undesirable because it leads to a high message duplication rate when using any variation of flooding. Reliance on a small subset of high resource nodes can also lead to scalability and attack vulnerability issues.

Spinneret bears some resemblance to recent work published using Chord as a base. G. Manku et al. [8] and Zhang et al. [12] define Chord structures very similar to our own. The first analyzes the theoretical path length of neighbor of neighbor greedy routing, where each node keeps track of its neighbor's neighbors so that it has a larger table to use when making next-hop route decisions, a scheme they call NoN-greedy routing. They find that NoN-greedy routing results in paths of length $\Theta(\log n / \log \log n)$. The second paper discusses using latency to refine fingers in a Chord to decrease lookup latency. This, in effect, creates a finger table where each finger is chosen from within $[2^{i-1}, 2^i)$, where $1 \leq i \leq m$. However, neither paper analyzes these semi-structured Chord networks in the context of unstructured search, nor considers the implication of churn to their performance.

3 The Spinneret Substrate

Spinneret is a peer-to-peer network substrate that is designed to support both structured and unstructured P2P algorithms. We start by discussing the numerous goals we have for such a shared substrate. These goals apply to both practical and algorithmic concerns. Finally, we define the network structure and the application interface that characterize Spinneret.

3.1 Design Goals

The primary goal for virtually any P2P network is to offer a service on top of the resources provided by the individual nodes in the network. This typically requires some form of query being injected into the network, which should result in one or more node ids, satisfying the query, being returned. In Spinneret we do not provide a query mechanism. Instead we leave the semantics of the upper layers open for specialization on top of the common network substrate.

Any type of P2P network that exhibits unbalanced resource usage, such as super-peer structured networks or
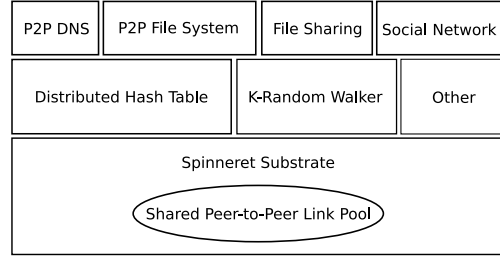

Figure 1. The Spinneret Stack: An example Peer-to-Peer stack on top of the Spinneret substrate.

those that utilize power-law connectivity, are vulnerable to directed attacks that can disproportionally effect large numbers of nodes. For example, in super-peer networks, disabling only a subset of super-peers can severely cripple or even partition the entire network [2]. By maintaining balance in the network, the Spinneret system should deemphasize the importance of any specific node, thus minimizing the effect of this form of attack.

As peer-to-peer networks proliferate and an increasing number of applications are P2P aware, the overhead of simply maintaining connections for multiple networks becomes prohibitive for a typical desktop computer. If mobile devices and other resource constrained computers are to be connected to P2P networks, reduced resource usage is necessary. To that end, a shared network infrastructure that can be used simultaneously by multiple P2P protocols would vastly reduce the strain on both local operating system resources as well as global network resources. Figure 1 shows an example architecture that could be implemented on top of the Spinneret substrate.

Currently, P2P application developers must spend a considerable effort engineering their basic network-level structures. For example, maintaining active links to a large number of neighbors requires constant heartbeat messages and discovery of new peers to take the place of those that have left. Separating this aspect of P2P protocol development from the algorithmic design should ease the process of P2P development. Furthermore, as is suggested by Gummadi et al. [5], such a separation would allow research to focus on the best designs for each component and progress along parallel tracks. Research at both the substrate layer and the higher algorithmic layer would no longer be intertwined.

3.2 Network Structure

Spinneret is designed to support graph traversal to any node in as close to logarithmic time as possible, while not requiring the strict maintenance of links between any specific pairs of nodes. Intuitively, our solution is to organize

the network into an approximation of a DHT, but keep the exact structure flexible enough so that maintenance can be done in a very light-weight fashion. Here we describe the initial design of the of the Spinneret substrate.

We describe the structure of the Spinneret substrate within the reference model proposed by Aberer et al. [1]. We model a P2P network as a directed graph $G = (P, E)$ where P is a set of $n = |P|$ peers, E the set of edges, and an edge $(i, j) \in E$ represents the connection between peers i and j. We also denote with $N(i)$ the neighbor function, the set of peers to which i is connected, so $N(i) = \{j \mid (i, j) \in E\}$. The properties of the network are completely defined by the N function, which is used to derive the link table maintained by each peer in a Spinneret network.

Specifically, the N function defines the *network geometry* as discussed by Gummadi et al. [5]. The geometry pertains to the pattern of links connecting the P2P network. In practice, the geometry is defined by an addressing scheme and by a *distance function* d, which is a design parameter of Spinneret. The addressing scheme is based on a closed and uniform address space A. In practice, we can assume that A consists of all B-bit binary strings. Each participant in the network is assigned an address a from A. Assignments from A must be uniformly random so that no part of the address space is more densely populated than another. This is very similar to the way addresses are assigned in many DHTs. (In the following text we simplify the notation a bit by denoting with $i \in P$ both a peer and its address.)

The Spinneret geometry is then defined according to the distance function $d : A \times A \to \mathbb{R}$. From a network-wide perspective, the geometry consists of a random graph where the probability that link $(i, j) \in E$ is proportional to $2^{-d(i,j)}$. From a local perspective, each peer i maintains a set of links to peers whose addresses are chosen uniformly at random within regions of the address space that are at exponentially-increasing distances from i. In other words, $N(i)$ contains addresses that are distributed uniformly at random over a $\log d$ scale.

This results in a geometry that is akin to a loosely defined Chord [11], but rather than mandating specific neighbors at certain addresses it will accept any node within a given address range. Such "randomized" geometries have been suggested before [8, 12], but while our formalization differs only slightly, none of the previous work has studied this type of geometry as the basis for both structured and unstructured algorithms.

As is shown in Figure 2, the link table maintained by the substrate at a peer i is represented as a set of *bins*, where each bin contains nodes twice as distant from i as the previous. In order to adjust the size of this link table each bin has a given number of *slots* that define the maximum number of links that a bin may hold. The number S of slots in each bin is a design parameter of Spinneret that can be used to adjust
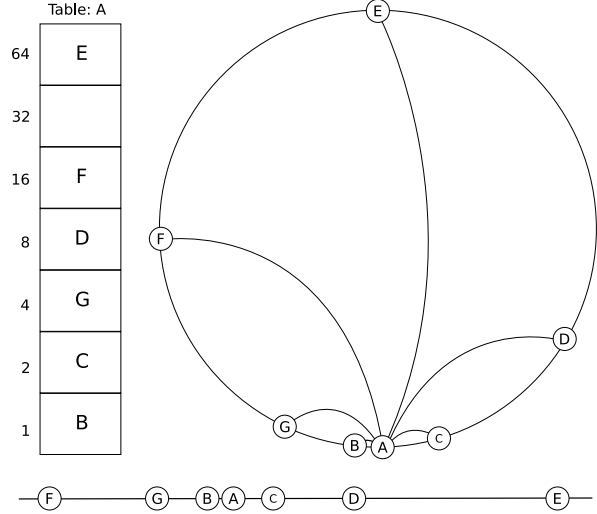

Figure 2. Log-Bin Structure: The log-bin structure is shown for node A.

overall network connectivity.

3.3 Substrate Parameters

The Spinneret substrate is parameterized by three main components: an address, a distance function, which takes two addresses and returns the distance between them, and the number of slots in each address bin, which determines the number of links maintained by each node.

- **Address**: The address assigned to each node is used to place that node in relation to other nodes in the network. We leave the specifics of the address space and properties of distribution up to the higher layers, but for most systems this will probably be a uniformly random value such as the result of a hash function over an object or address. The dimensionality of the address space also directly affects the choice of efficient algorithms for a given P2P network. Although the distance value will always be used to determine which bin a remote peer should be placed in, the substrate can not know in which direction in multi-dimensional space this peer resides. Therefore, there is no guarantee that a given bucket will indeed hold a node in the appropriate direction in order to answer a query. However, as long as the dimensionality remains low, and the number of slots for each bin is reasonable in relation to this dimensionality, the probability of having a useful next hop remains high.

- **Distance Function**: The choice of distance function in structured P2P networks has a direct impact on the algorithms used to build routing tables, search for data,

and maintain the network. As the distance function is what is used to build the logarithmic link table, all nodes wanting to cooperate to provide a P2P service will need to share their distance function. We imagine that two networks with different distance functions could run side-by-side, each using their own routing tables.

- **Connectivity:** In order to be confident that the network is connected, all nodes maintain links to other nodes through at least some minimum number of neighbors. Erdös and Rényi show that with very high probability a random graph with more than $\frac{n \ln n}{2}$ edges will be connected. In Spinneret each node tries to maintain order $O(\log |A|)$ links to peers, for order $O(n \log |A|)$ total links in the entire network. The level of connectivity can be further increased on a node by node basis by adjusting the number of bin slots

3.4 Maintenance

In order to determine when replacement of a node entry in the link table is necessary, Spinneret needs a way to detect failing or failed peers. A typical heartbeat mechanism that pings all peers in the link table once per time t_h can be used to determine liveness. Ideally, these pings would be spread out over t_h, and t_h could be relatively large, in order to reduce network traffic. If links are symmetric, nodes can send one ping every $2t_h$, as receiving a ping indicates the sender is alive, alleviating the need to send a ping to the sender for another t_h. As such, the total failure-detection traffic generated in the network amounts to $nS \log |A|/2t_h$ messages per time unit, or $nS \log |A|/t_h$ in the case of asymmetric links.

The primary role of this shared substrate is to keep itself informed about the status of remote nodes in the network. For the purposes of firewall traversal and low overhead keep-alive and routing, most P2P systems utilize UDP datagrams at the socket layer [10]. Thus connections are not held open, but up to date status information about each of a node's peers is maintained through periodic messaging.

While we can imagine many possible maintenance schemes for Spinneret, there are two main classifications: opportunistic, and proactive. Opportunistic algorithms use the naturally occurring message traffic in the network to refresh bins, replacing failed or poorly performing nodes. Proactive schemes would actively search for new nodes, possible in response to under-full bins or failed neighbors.

Opportunistic schemes should have zero additional cost over the basic heartbeats. As remote peers make requests via the Spinneret interface, their routing information is cached in a pool of nodes available to be used for replacement. When a failure is detected the maintenance algorithms find an appropriate replacement. If no appropriate

replacement is found then a proactive scheme can be utilized.

Proactive schemes require nodes to perform extra work to find replacements for failed links detected using heartbeats. Instead of, or in combination with, using the opportunistic algorithm, proactive recovery would attempt to find nodes in specific distance ranges in order to keep bins full. Ideally, a proactive approach would maintain full bins at all times, or at least keep each bin as full as possible given a specific distribution of nodes in the network.

Preliminary work, not presented in this paper, has pointed toward hybrid schemes consisting of an initial proactive bootstrap phase using a push-pull mechanism [10], and a second phase that is opportunistic, with fall-back to push-pull under network stress.

3.5 Substrate Interface

The interface to Spinneret is structured around the address space. Two methods are used to configure the node and two methods are used to query the underlying substrate. Specifically, *set_address*(**address**) sets the address of the current node; *set_distance_function*(**distance_func**) sets the distance function to be used for building the link table; *get_random_node*() returns a random neighbor, uniformly chosen across all bins in the link table; and *get_closest_node*(**address**) returns the closest node in the link table to the given address. Many higher level protocols, both structured and unstructured, can be implemented using only these four methods. We give examples of two such protocols in Section 4.

4 Analysis

Here we describe two typical P2P algorithms implemented on top of the Spinneret log-random substrate: a Chord like DHT and a k-walker random walk. We also believe that various types of range queries, semantic searches, and other P2P search algorithms would work well on top of Spinneret, but lack of space presents us from presenting them here. Currently, we use a simple euclidean distance function ($d(i, j) = |j - i|$) in a single dimensional address space, which will probably suffice for most applications, but it is possible that for some systems a different distance function might be appropriate.

4.1 K-walker

The ability to make efficient queries in an unstructured network reduces to a simple task: visit the most nodes in the network in the shortest amount of time, while using the smallest number of messages. Lv et al. [7] present a k-walker random walk algorithm which produces results sim-

Algorithm 1 K-walker random walk search

```
 1: procedure RANDOM_WALK_SEARCH(o, htl)
 2:     if have_object?(o) then
 3:         return self
 4:     else if htl = 0 then
 5:         return none
 6:     else
 7:         next ← spinneret.get_random_node()
 8:         return next.random_walk_search(o, htl − 1)
 9:     end if
10: end procedure
11:
12: procedure K_RANDOM_WALK_SEARCH(k, o, htl)
13:     results ← [ ]
14:     for i ∈ (1..k) do
15:         results[i] ← random_walk_search(o, htl)
16:     end for
17:     return first_result(results)
18: end procedure
```


Figure 3. Random Walker Success Rate: The probability of success of the random walker vs. the number of replicas placed in the network.

ilar to that of the original Gnutella-style flooding, but they cut traffic by two orders of magnitude by reducing duplicate messages. Furthermore, Lv et al. also found that random graphs are the best network structure for supporting random walks, as they result in better load distribution among nodes. For these reasons the k-walker random walk algorithm is a good fit on top of the Spinneret substrate.

The k-walker algorithm starts a query by issuing k requests to a set of k random nodes from the underlying substrate. At each node the message is forwarded on to a random neighbor until the hops-to-live (HTL) has expired. In this simulation we did not implement the checking described by Lv et al., which would result in additional benefits in terms of saved network messages. We did however simulate their state-saving technique, which helps to further minimize duplicate messages by not forwarding random walkers for a given query to the same neighbor twice. Additionally we test the success of queries where multiple nodes can satisfy the query request. This test models the replication of a given object onto multiple random nodes in the network.

In Algorithm 1 we present pseudo-code for the basic random walk search on top of the Spinneret substrate.

4.1.1   Results

In order to validate our simulation we compare our results with an analytical solution. A random walk can be thought of as a method for efficiently taking an approximate random sample from a graph structure, thus we use uniform sampling to model our ideal performance [3]. The probability of finding the desired node in a uniformly sampled network is the sum of the hypergeometric distribution:
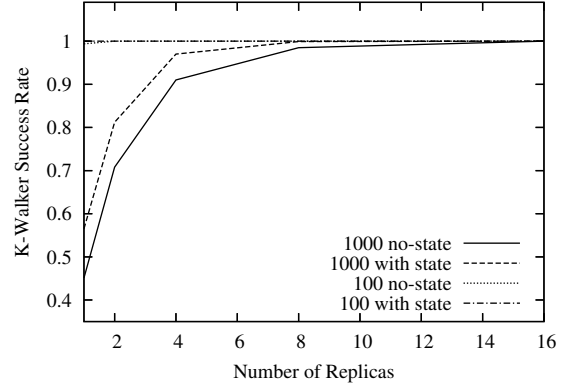
$$f(x|R, F, s) = \sum_{x}^{R} \frac{\binom{R}{x}\binom{F}{s-x}}{\binom{R+F}{s}} \quad (1)$$

Here R is the number of nodes that hold replicas of the desired data, F the number of the nodes that do not, and s is the sample size, which is the number nodes sampled. This reduces to a binomial distribution in the case where $R = 1$. x is the minimum number of nodes that we would like to find as the result of a given query, so typically $x = 1$, because we only need one copy to be successful.

The primary metric for testing the k-walker random walk algorithm is the percentage of successful queries which are achieved. In the following experiments, 100 queries are made over 10 different networks for a total of 1,000 queries per data point. From each randomly selected node we launched 32 random walkers with an HTL of 20, and searched for another randomly selected node, or in the case of replication, a set of nodes. In Figure 3 the k-walker algorithm is run for increasing levels of replication in both a 100 and 1000 node networks. Our results show the stateless walkers succeeding at around 45% while the stateful walkers improved to an average of right around 56%. The theoretical probability given by Equation 1 for this configuration is ∼64%, which implies that the stateful walkers are closely approximating a uniform random sampling. We also found that the number of slots per bin had almost no effect on the performance of the k-walkers.

Figure 3 also shows that a small amount of replication results in considerable gains in the query success rate. These numbers are consistent with both our analytical model in Equation 1, as well as the results found by Lv et al.

Figure 4 shows the number of times nodes are visited more than once during a run of the k-walker algorithm with
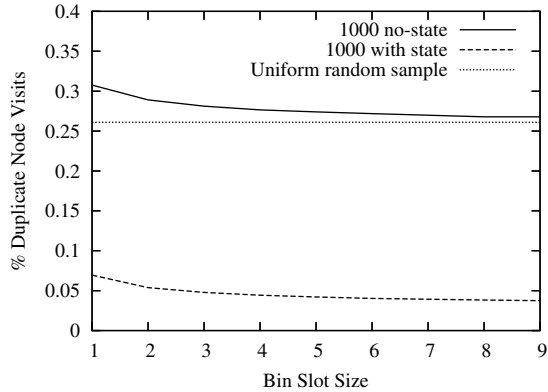
Figure 4. Nodes in Multiple Walker Paths: The percent of node visits that are at nodes already seen by a walker.

Algorithm 2 DHT search
1: **procedure** DHT_SEARCH(a)
2:    $closest \leftarrow spinneret.get\_closest\_node(a)$
3:    **if** $closest = $ none **then**
4:        $closest \leftarrow close\_immediate(a)$
5:    **end if**
6:    **if** $closest = $ none **then**
7:        **return** $self$
8:    **else**
9:        **return** $closest.dht\_search(a)$
10:    **end if**
11: **end procedure**

32 walkers, each set to an HTL of 20. The uniform random sample line represents duplicate visitations in 640 samples across 1000 nodes. This represents the amount of duplication one would expect from a k-walker random walk if each node visitation was independent of the last, (i.e, a truly random sampling). Figure 4 suggests that Spinneret offers a near-random sample of the nodes in the network, even though there is a bias toward close nodes in the link table, with the number of duplicate visits approaching the level seen in truly random sampling. Adding state to the walker further reduces duplication, although some remain from multiple path pairs to the same destination peer.

4.2   DHT

We have chosen a ring structure similar to the one introduced in the Chord system [11] as the basic geometry for our DHT implementation [5]. Each node chooses a uniformly random identifier $a \in A$, where A is an B-bit identifier space which wraps around at the ends of the number line. Each node in the DHT is responsible for the region of address space at its own identifier as well as half the distance to each of its immediate neighbors, the predecessor and the successor.

Sitting on top of the Spinneret substrate gives the DHT access to a table of nodes which are already placed in bins of logarithmically increasing distance. This flexibility in route selection for bins does not effect path distance. As is shown by Gummadi et al. [5], the ith neighbor used for logarithmic routing on a ring can be any node in the range $[(d + 2^i), (d + 2^{i+1})]$. This means that for each bin there are 2^i possible neighbors.

To route a query packet which is looking for a given id, at each hop the routing node requests the closest node to the id from the Spinneret link table (using *get_closest_node*(**a**)), and then forwards on to that node. If no nodes in the link-table are closer than the nodes in the neighbor list, the neighbor list is used instead. Routing can be done either iteratively, where the requesting node does successive queries to nodes, or recursively where each node forwards the packet without communicating to the requesting node.

4.2.1   Results

Figure 5 shows the average number of hops taken from one node to another during a DHT style search in a 1000 node Spinneret network. 10 trials were performed for each bin size, with each trial consisting of 100 searches performed between randomly selected node pairs. The error-bars depict the minimum and maximum number of hops seen across the 10 trials. The trials with bins of slot size 0 are a special case, as only the successor and predecessor links are used, resulting in ∼222 hops on average. This is not a realistic case, and is therefore not show on the graph. Adding a single node per bin reduces this hop count to ∼6.0 hops, with additional slots further reducing the hop count. All trials successfully found their search target because we are not simulating failure in these experiments.

We also looked at the use of predecessor and successor links during a DHT-style lookup. While these links are necessary for correctness, their use is less than optimal and indicates that the bins did not hold a useful link. For slot size 0, every next hop is necessarily either a predecessor or successor, so they are used for every hop. For the other numbers of slots the usage quickly drops, reaching 0.21 uses per search with slot size 1, and 0.0 at slot size 5.

Finally, we looked at the density of the usage of addresses in the address space. The density was changed by holding the number of nodes steady while increasing the size of the address space. This should have no effect on the performance of DHT-style search, and our experiments run over a 1000 node topology using a bin slot size of 3 show that this is in fact the case.
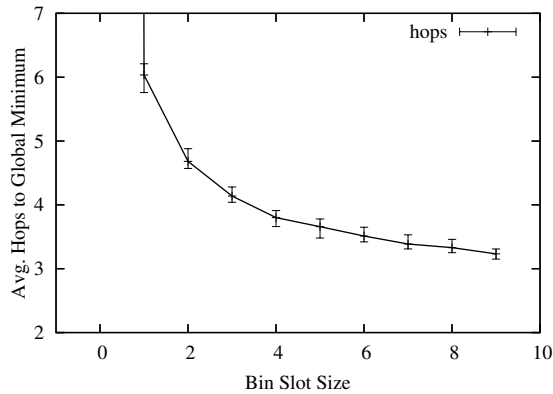
Figure 5. Avg. Hops to Search Target: For bin sizes 0–9 the graph shows the average number of hops taken to find a target node. Bin size 0 forces the DHT to use only predecessor and successor links.

5 Conclusions and Future Work

We have presented Spinneret, which is a low-level substrate for developing peer-to-peer networks. By maintaining a log-random network structure, higher level algorithms can make use of both structured and unstructured style search techniques.

The algorithms presented are two of the most common search techniques for P2P networks, but in the future we would like to experiment with a wide variety of algorithms on top of Spinneret. One family of search that is missing is a query mechanism for multi-dimensional or fuzzy data. A distributed clustering algorithm, for example, would be complementary to the current set.

One of the primary goals in creating a two-level design, where the substrate maintains the network structure while higher layers utilize the set of shared links, is to separate network maintenance from the algorithmic aspects of P2P systems. The two case studies we presented are an initial validation of the network structure, but our current work is now focused on efficient network construction and maintenance algorithms. Additionally, we plan to consider the implications of using different distance functions and address assignment schemes.

An additional unexplored aspect of this work is looking at the interaction of multiple P2P algorithms running side by side. Although the substrate can ameliorate problems related to local resource usage, such as open network connections, it can not deal with distributed issues such as network congestion from one service effecting the performance of another service. Looking at bandwidth usage, latency and other issues in specific examples of P2P services will be necessary for achieving the overall goals of this work.

Finally, another key characteristic of this work is to develop a structure that does not require unfair usage of resources. Towards that end, we would like to study equilibria in the management interactions taking place within the substrate, with the purpose of structuring intra-peer communications in a way that does not allow free-riders.

References

[1] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The essence of P2P: A reference architecture for overlay networks. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 11–20, Konstanz, Germany, Aug. 2005.

[2] N. Daswani and H. Garcia-Molina. Query-flood DoS attacks in gnutella. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 181–192, Washington, DC, USA, Oct. 2002.

[3] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proc. of the 23rd Conference of the IEEE Communications Society (INFOCOM)*, Hong Kong, China, Mar. 2004.

[4] L. Gong. Project JXTA: A technology overview. technical report, SUN microsystems, Apr. 2001.

[5] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '03)*, pages 381–394, Karlsruhe, Germany, Aug. 2003.

[6] P. Kirk. Gnutella protocol rfc, 2002.

[7] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95, June 2002.

[8] G. Manku, M. Naor, and U. Wieder. Know thy neighbors neighbor: the power of lookahead in randomized P2P networks. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing (STOC 04)*, 2004.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, pages 161–172, Aug. 2001.

[10] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of USENIX'04*, pages 127–140, Boston, Massachusetts, USA, June 2004.

[11] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. A scalable peer-to-peer lookup protocol for Internet applications. Technical Report TR-819, Lab. Computer Science, Massachusetts Institute of Technology, 2001.

[12] H. Zhang, A. Goel, and R. Govindan. Incrementally improving lookup latency in distributed hash table systems. *SIGMETRICS Performance Evaluation Review*, 31(1):114–125, 2003.