Constructing a Balanced, (log(N)/loglog(N))-Diameter Super-Peer Topology for Scalable P2P Systems

Young June Pyun
Department of Computer Science
North Carolina State University
Raleigh, North Carolina
yjpyun@unity.ncsu.edu

Douglas S. Reeves
Departments of Computer Science and
Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina
reeves@eos.ncsu.edu

Abstract

Current peer-to-peer (P2P) file sharing applications are remarkably simple and robust, but their inefficiency can produce very high network loads. The use of super-peers *has been proposed to improve the performance of* unstructured *P2P systems. These have the potential to approach the performance and scalability of structured systems, while retaining the benefits of unstructured P2P systems. There has, however, been little consensus on the best topology for connecting these super-peers, or how to construct the topology in a distributed, robust way.*

In this paper we propose a Scalable Unstructured P2P System (SUPS). The unique aspect of SUPS is a protocol for the distributed construction of a super-peer topology that has highly desirable performance characteristics. The protocol is inspired by the theory of random graphs. We describe the protocol, and demonstrate experimentally that it produces a balanced and low-diameter super-peer topology at low cost. We show that the method is very robust to super-peer failures and inconsistent information, and compare it with other approaches.

1. Introduction

Peer-to-peer (P2P) file sharing, where files are searched and downloaded between hosts or peers without the need for central servers, has emerged as a major component of Internet traffic over the last few years, The growth rate of P2P has been phenomenal. As a consequence, P2P systems have reached their limit on the performance as their traffic overwhelms the available bandwidth capacity of the underlying Internet [11].

P2P systems generally form, at the application level, a decentralized overlay network with its own routing mechanism. Due to its decentralized nature, the topology of the overlay network and it's routing mechanism determine such system properties as performance, robustness, and scalability. So far, two major categories of P2P systems have been introduced.

The most common P2P systems are the simple and practical *unstructured* P2P systems where data are shared among peers in a naive fashion. In these systems, data are stored anywhere in the system and are located by broadcasting queries to all peers within a specified distance. These methods are simple, practical, and highly robust to changes in the overlay network topology. However, the inefficiency of broadcasting raises doubts about their scalability.

For Internet-scale applications, scalability is a primary issue, perhaps *the* primary issue. Many ways have been proposed to address this problem of unstructured P2P methods. A particularly important class of proposals for improving scalability is *structured* P2P systems. In a structured P2P system, network topology and data placement are carefully designed in order to support efficient and scalable searching. The drawbacks of such methods are their additional complexity, lower robustness and adaptability, and limited search capabilities (e.g., for range and keyword matching queries). For these reasons, structured P2P systems are not considered further in this paper.

Another proposal for unstructured methods has been the addition of *super-peers*. Super-peers are selected for their larger capacity and greater capabilities from among the set of peers. This approach essentially creates a hierarchical overlay network, where the top layer consists of the super-peers, and the bottom layer consists of the peers. While retaining most of the simplicity and flexibility of unstructured systems, the addition of super-peers has the potential to substantially improve their efficiency and scalability. To date, however, there has not been a thorough examination of how to organize the super-peers (i.e., what topology they should have) to maximize their benefits to the fullest extent.

In this paper we propose a new P2P system entitled the "Scalable Unstructured P2P System" (SUPS). SUPS is an unstructured P2P system in which the interconnections between super-peers are selected to approximate a random graph. The super-peers organize in a fully distributed way, such that the resulting overlay network will have a balanced load and a logarithmic diameter, with minimum node degree. Attention is paid to recovery from super-peer failure, and the method rapidly adapts to changes in the set of peers and super-peers. The super-peer topology construction and maintenance protocols are described. We investigate the performance by means of simulation, and conclude by comparing with two other approaches.

2. P2P Systems

In this section we briefly review some previously-proposed P2P systems.

In P2P systems, file locations are stored in a distributed index. The major design issues are the topology of the P2P network, the organization of the distributed index, and the strategy for querying that index.

Unstructured P2P systems have little or no relationship between the network topology and index placement. Generally, these peers self-configure into an overlay network with no particular intended topology. Since these systems have no coupling between the network topology and data placement, locating a desired file is not easy. A blind search that probes the whole network (i.e., flooding) is the typical query method ([14], [12]). In many unstructured P2P systems, files are replicated at multiple peers. The result is that excessive bandwidth is consumed both for flooding the queries and for receiving the responses. To mitigate the waste of bandwidth, most unstructured systems sacrifice data availability by bounding the scope of the flooding process. The appeal of such methods is their simplicity, their flexibility, their robustness, the scalability of searching (which is conducted in parallel by many peers), and their adaptability to frequent changes in the set of available peers and files. Because of these attractive qualities and the undoubted success of such approaches, considerable research has been conducted to analyze these systems ([11], [17], [19], [8]), and to propose methods of reducing their bandwidth requirements ([20], [12], [16], [15], [10]).

A popular example of an unstructured P2P system is Gnutella ([2], [5]). Each Gnutella peer independently creates connections to a set of neighbors known from past experience (i.e., cached), or learned from a well-known server. The actual selection process is random. Once connected, peers use "ping" and "pong" messages to maintain the overlay network. Querying in Gnutella is handled by flooding; any peer having a matching index responds by routing the response back on the reverse path to that taken by the query. Gnutella bounds the scope of flooding to a maximum path length of 7 peers.

To make unstructured systems more scalable, the concept of *super-peers* has been suggested. The use of super-peers creates a two-layer hierarchy in the P2P system. Popular examples include KaZaA [6] (based on the proprietary Fasttrack [1] technology), and a new version of Gnutella [3]. In these systems, super-peers are specially designated nodes with higher bandwidth connectivity. The super-peers connect to each other, forming the upper level in the overlay network hierarchy. Each super-peer acts as a server on behalf of a set of client peers, who form the lower level of the network hierarchy. The super-peers normally construct a topology and propagate queries in the same fashion as unstructured P2P systems. Super-peer systems thus represent a balance between the inherent efficiency of centralized search and the autonomy, load balancing, and robustness of distributed search methods [21]. In addition, they exploit the heterogeneous peer capabilities (e.g., bandwidth and processing power) which are normal in most P2P systems ([19], [15], [10]).

In Gnutella v0.6 [3], super-peers are termed *ultrapeers*, and were introduced to overcome scalability problems. The specification of the ultrapeer topology has not been described in detail, and there is little experience yet with deployment, so the impact on actual performance has not been reported. It is recommended that client peers connect to two or three ultrapeers for reliability purposes. The proposed number of connections each ultrapeer should maintain is 32 (normally 6), while the proposed maximum number of clients in a cluster is 30 [7], without detailed justification. We believe the addition of super-peers can substantially improve the scalability of P2P systems. A question that remains unanswered, however, is what the optimum topology for the super-peer network is, and how that topology can be constructed. In the next section we present the basis of our approach for constructing this topology, and in the section following that, we present the method itself.

3. Basis For The Method

We believe the most important requirement for a scalable P2P network should be to construct a topology with a low diameter. We focus on the topology of the super-peers, since they are responsible for storing and searching the index. One consequence of a low diameter is that every query can be flooded to all super-peers over relatively short paths, ensuring that data will

be found if it exists. In addition to low diameter, it is desirable to construct a topology that is mostly regular, for load balancing purposes and avoidance of hot spots. Finally, minimizing the number of links or connections in the network is important for efficiency.

3.1. Random Graph Theory

Random graphs were originally investigated by Erdös and Rényi ([13], [9]), and shown to have a number of highly desirable properties. We summarize below some of the theory of random graphs for the model proposed by Erdös and Rényi, which we call the ER model. This theory inspired our method for dynamically constructing the super-peer network.

$\mathscr{G}(n, M)$, denoted as G_M, is an ER model that consists of all graphs with vertex set $V = \{1, 2, \cdots, n\}$ having M edges, in which the graphs have the same probability, and M is a function of n. A random graph G_M is generated by a *random graph process* (or simply *graph process*); a Markov chain $\widetilde{G} = (G_t)_0^\infty$, whose states are graphs on V. The process starts with the empty graph and for $1 \leq t \leq \binom{n}{2}$ the graph G_t is obtained from G_{t-1} by the addition of a single edge, all new edges being equiprobable. Then G_t has exactly t edges and maps to G_M at time M. During the process, most of the properties of the random graphs appear rather suddenly, that is, for M below a certain value almost no G_M has the properties, while for M above this value, almost every G_M has the property.[1] Suppose Q is a certain property of graphs, then the value of M at which Q suddenly appears is called the *hitting time* or *threshold* of Q.

Among many properties of the random graphs, some properties are closely relevant to our objective. The first of them is the property of *strong homogeneity*. That is, a G_M is an *almost regular graph* such that its vertices have about the same degrees. This property is relevant to the *load balancing* of our objective. While random, totally regular graphs are too complicated to be generated, simple random graphs which are almost regular are quite simple to generate.

The second property of concern is graph *connectivity*, which is certainly desired for P2P systems. Erdös and Rényi showed that $(n/2)\ln n$ is a sharp threshold function for connectedness. That is, as long as at least $(n/2)\ln n$ edges are maintained, the network will be connected with high probability. In addition, it can also be proved that almost every graph becomes k-connected at the same time its minimum degree becomes k.

[1]Note that the term *'almost every'* in this context means that the probability tends to 1 as $n \to \infty$.

The most important property for our purposes is the *diameter* of random graphs (the diameter of a graph G, denoted $diam(G)$, is the maximal shortest distance between any pair of vertices of G). We are especially interested in the diameter of the sparsest connected random graphs G_M, which is the diameter at the hitting time of connectedness in a graph process. If the hitting time of connectedness is τ, then Erdös and Rényi proved that almost every graph process $\widetilde{G}$ is such that

$$\left\lfloor \frac{\ln n - \ln 2}{\ln \ln n} \right\rfloor + 1 \leq diam(G_\tau) \leq \left\lceil \frac{\ln n + 6}{\ln \ln n} \right\rceil + 3.$$

In other words, almost every random graph G_M with $M \geq (n/2)\ln n$ has a diameter of $\Theta(\ln n / \ln \ln n)$.

These properties of the ER model are very attractive for P2P systems. Unfortunately, it is not possible to directly apply the ER model to P2P systems due to their dynamic nature. In particular, the ER model assumes the graph has a fixed set of vertices; only edges are added by the random graph process. In unstructured P2P systems, in contrast, the set of peers and super-peers may be continually changing, and their number may grow and shrink in unpredictable ways. It is not possible, therefore, to directly construct a random graph in a way consistent with the ER model.

3.2. A Random δ-Process

We now present a method of approximating the ER model for graphs where the number of vertices is allowed to change, as will be the case for P2P systems.

As discussed above, for the ER model the hitting time of connectedness is $(n/2)\ln n$, where n is the number of vertices. This means the average vertex degree is $\ln n$. While the ER model does not guarantee a non-trivial bound on vertex degree, in our method we require that the minimum vertex degree be $\lceil \ln n \rceil$. The motivation for this is to produce networks that are connected and almost regular. We will denote the minimum required vertex degree as $\delta(n)$ (when n is understood, we will simply refer to this as δ).

For a graph with a fixed set of n vertices, we propose to modify the random graph process in the following way. Rather than add edges between all vertices equiprobably, at each step we add an edge between a vertex with degree less than δ and a vertex of minimum degree. These vertices are selected randomly from among the vertices that qualify. The process terminates when all vertices have degree at least equal to δ. We call this a *random δ-process*. It is somewhat similar to the random d-process of [18], where the degree is bounded above by a maximum value. It is a simple and fast way to generate random graphs with degree restrictions. The resulting graphs are not

guaranteed, however, to have the properties of the ER model.

We now extend the δ-process to the case of graphs with varying numbers of vertices. Suppose a graph has been constructed as above, for a fixed set of vertices of size n, and a new vertex is added to this set. The process of adding this vertex is just as above: add edges, one at a time, between a vertex with degree less than $\delta(n+1)$ and a vertex of minimum degree, until the degree of all vertices is at least $\delta(n+1)$. Conversely, when a vertex is removed from a graph of size n that has been constructed as above, of course all edges connected to this vertex are also removed. From among the remaining vertices, the process is as before: add edges, one at a time, between a vertex with degree less than $\delta(n-1)$ and a vertex of minimum degree, until the degree of all vertices is at least $\delta(n-1)$.

While the size of the graph may thus vary, the desired minimum degree, which is a function of the graph size, is maintained by the above process. Each time the graph size changes, given sufficient time the δ-process will terminate with the required minimum degree.

Figure 1 shows an example of the δ-process, starting with 6 vertices and $\delta = 2$. When a new vertex 7 joins in (step a), δ is still 2, 2 edges must be added to the graph. First an edge is added to a randomly selected vertex with minimal degree, which is vertex 6, followed by an edge to randomly-selected minimal-degree vertex 4. When a new vertex 8 joins (step b), δ becomes 3; 3 edges between vertex 8 and other vertices must be added. Vertex 7 is first selected (since it has minimum degree), and then two more vertices are randomly selected. When a new vertex 9 joins (step c), several candidates with the same minimal degree are randomly selected to have an edge with vertex 9. When an existing vertex 6 leaves (step d), its neighbors 1,5,7, and 8 lose one edge each. However, only vertex 8 now has a degree less then $\delta = 3$ and must add an edge to a random vertex with minimal degree.

3.3. Comparison of Random δ-Process and ER model

The key factor in the graph process of the ER model lies in its pure randomness in creating edges, with all new edges equiprobable and independent. The problem introduced when the number of vertices changes over time is that the vertices have different *durations*. Ignoring this when adding edges would produce a bias such that vertices with longer durations would tend to have higher degree than vertices with shorter durations.

Our random δ-process restricts the vertex degree to correct this bias. Although it can not be claimed that the result is a random graph consistent with the ER model, our experimental results (section 5) indicate the δ-process produces graphs with properties very similar to random graphs. Even under highly dynamic conditions, with unreliable peers, and using approximations required for distributed execution, the graphs produced are always connected, have logarithmic diameters, are almost regular, and have low degree. The next section describes a protocol based on the δ-process described above.

4. Implementation

In this section, we present a "Scalable Unstructured P2P System" (abbreviated SUPS), which is a protocol for constructing the super-peer topology of an unstructured P2P system. The concept of super-peers and a two-level hierarchy is a natural way to exploit the heterogeneity of peers. The goal of SUPS is to significantly improve the performance of super-peer overlay networks at low cost. We substitute the terms "networks" and "nodes" (super-peers) in this discussion of implementation, in place of the terms "graphs" and "vertices" from the previous discussion of the basic algorithm.

SUPS is designed for dynamic P2P systems in which peers join and leave the network at a very high rate, which is normally the case. The disruption rate (number of nodes affected) by a node joining or leaving is very low, and the method of accomplishing joins and leaves is fully distributed.

The protocol implements the *random δ-process* described in the previous section. With a simple degree-bounded process, the resulting topology provides the most efficient underlying structure for systems requiring scalability and high performance. This process relies on a single invariant, which is the minimum degree of the system, δ. SUPS estimates δ in a distributed manner, eliminating the need for super-peers to have knowledge of the global topology. The estimation mechanism provides not only the correct value to the δ-process, but also tolerates failures and concurrent joins and leaves, an important consideration for P2P systems.

The SUPS protocol is intended only to create the super-peer topology. We assume that normal (non-super) peers are each connected to 2 super-peers for reliability. We also assume that super-peers are selected from normal peers that have high bandwidth, high computing power, a long residence time, and a low likelihood of failure, as suggested in [4]. The detailed choice of super-peers remains as a separate research topic not addressed in this paper.

Control messages required for topology construction are "piggybacked" onto normal query messages, which

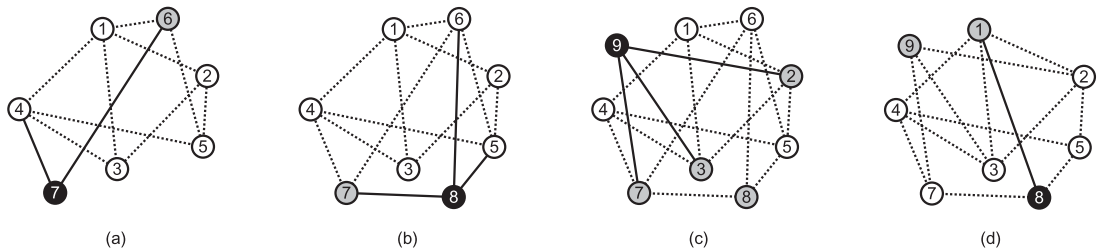

Figure 1. An example of δ-process, starting with 6 vertices (δ=2): (a) after a vertex 7 joins (δ=2), (b) after a vertex 8 joins (δ=3), (c) after a vertex 9 joins (δ=3), (d) after a vertex 6 leaves (δ=3). [white circle=existing vertex, grey circle=minimal degree vertex, black circle=vertex requiring more neighbors, dashed line=existing edge, solid line=new edge]

are flooded throughout the network. Since each super-peer forwards requests on behalf of a large number of peers, intervals when there are no queries to be sent are likely to be quite short.[2]

4.1. Distributed Random δ-Process

SUPS uses the random δ-process that we presented in section 3.2 for generating its super-peer topology. It is therefore important to have an accurate estimate of the network size N, so that the appropriate value for δ is used. The estimation method is fully distributed, and each node i computes its estimate N_i independently, as described in section 4.3.

Using the local estimate N_i, the local value of δ_i is computed as $\lceil \ln N_i \rceil + 1$. This value for δ_i is one greater than that explained in the previous section, to provide a greater degree of fault tolerance in the face of inaccuracies in the estimates N_i.

A node computing a new value for N_i invokes the δ_i-process whenever its number of neighbors ($d(i)$) becomes less than δ_i. This can occur as the network size grows, or as connections are terminated when previous neighbors leave the system. The δ_i-process adds connections to new neighbors until the node has δ_i neighbors.

A node wishing to connect to a new neighbor attempts to choose the super-peer with the lowest degree among all super-peers. Each query message originated by node i includes its ID, its current local estimate of the system size N_i, and its current number of neighbors $d(i)$. Since queries are flooded to all super-peers, every node has an estimate of the degree of every other node in the system at all times. Each node i maintains a sorted list, called $MinList(i)$, of the $\ln N_i$ lowest-degree nodes in the system, according to its estimates.[3] Neighbors are added in increasing order of their estimated degree, with ties broken randomly (as required by the δ-process). Maintaining these lists at each node provides robustness in the event of failures, and/or a high degree of concurrency (rapid changes in the number of nodes and connections in the network). For each node in the list, the node ID, current degree, and its most recent estimate of N is stored by i.

4.2. Node Joins and Leaves

In a dynamic P2P system, new nodes will join and need to establish connections with existing nodes, and existing nodes will leave, requiring other existing nodes to establish new connections. The challenge for SUPS is to maintain a nearly-correct estimate N_i of the current size of the network at all nodes.

As stated above, a normal peer is connected to two super-peers. A normal peer selected by some process to be promoted to a super-peer inherits the $MinList$s and estimated values of N from each of its parents, for purposes of reliability. It computes its estimate of N_i as the ceiling of the mean of the estimates it gets from its parents. Node i also creates its own $MinList(i)$ from the $\ln N_i$ lowest-degree nodes from either of its parents' $MinList$s (ties broken randomly). It then creates connections to other super-peers in increasing order of their estimated degree, as described previously.

When a new node joins, some of the existing nodes may have to invoke the δ-process, if their new values of δ become greater than their degrees. Our experimental results (in section 5) shows that only a constant number of nodes are affected by a join in this way. The reason is that the function $\lceil \ln N \rceil + 1$ changes only if N increases or decreases by a factor of e from the value it had when δ was last revised.

In case of a node leave or failure, its neighbors learn of this event through the query timeout method; if a

[2] In the unlikely event this occurs, each super-peer can maintain a timer to generate an empty query, just for the purposes of transmitting a control message throughout the network.

[3] Before connecting to this node, i confirms that its estimate of the node's degree is correct by contacting that node.

node has not generated any queries for a time interval of T, it is automatically removed from the system. Some of the former neighbors may have to invoke the δ-process as their degrees decrease, possibly below their estimated value for δ. As explained above (and shown by the experiments), this tends to affect only a small number of nodes.

4.3. Local Estimation

Proper function of the δ-process requires an accurate estimate of the global system size. This estimation consists of two separate procedures. First, an *update* procedure explicitly increments/decrements the local N_i of each node i in response to join/leave events. Second, an implicit *synchronize* procedure is used to diminish the variance of these estimates between the nodes.

We define the *primary neighbors* of a node j, noted as $Prim(j)$, as neighbors of j with an additional function used for estimation purposes. $Prim(j)$ is chosen when node j joins the network. Specifically, the first 3 neighbors to which node j connects when it joins the network are designated as the primary neighbors of j. If one of these 3 primary neighbors fails or leaves the system, an existing non-primary neighbor of j is randomly selected to replace it in $Prim(j)$.

The purpose of these 3 primary neighbors is to notify other nodes when node j joins or leaves the network. Each broadcasts a Join(j) or Leave(j) message in such an event; the triple redundancy provides resilience to faults in the network. A node i performs an *update* procedure, which increments or decrements its local estimate N_i, upon receiving at least 2 out of these 3 Join(j)/Leave(j) messages within a time interval of length T.

Although the *update* procedure would produce a correct estimate, it may not work in an unreliable network where node failures occur. That is, messages may be received from less than 2 of the primary neighbors for a variety of reasons. To enhance the accuracy of estimation, after performing an *update*, node i collects information from queries received from the nodes on $MinList(i)$ (as described above) for a period of time of length T. Node i then replaces its estimate N_i with the median of these values. The use of the median will tend to exclude estimates that are skewed by transient or local failures.

4.4. Costs

In some respects, SUPS has lower cost than other approaches to constructing super-peer topology. This is because fewer connections need to be established and maintained to achieve a low diameter network. The main overhead that is necessary is the additional information that has to be distributed with each query, i.e., the estimate of current system size, and the current number of neighbors. This requires no more than 3-4 bytes of overhead per query, which seems very reasonable given that Gnutella message headers are 23 bytes in length.

5. Experimental Results

In this section, we use simulation to evaluate the actual performance of the SUPS protocol. We investigate whether the protocol produces a topology with properties similar to that of random graphs, as well as some other important properties for P2P systems.

5.1. System Model for Simulation

Our simulation of SUPS focuses on the properties of the super-peer topology as it is being generated and maintained by the SUPS protocol. Note that all queries are flooded to all super-peers, and path lengths are fixed and almost regular. It is therefore not necessary to simulate the actual propagation of queries to determine network or processing load. The only effect of queries that is simulated is their affect on the maintenance of MinList at each node. For the sake of simplicity, we also assumed that sufficient time elapses after each join or leave for all super-peers to be informed, before another join or leave occurs. In most experiments the system was assumed to be reliable (i.e., no super-peer or connection failures); in one experiment, however, we measured the effect of failures on the protocol.

Our model of super-peer joins follows an M/M/∞ system with Poisson arrivals, exponential service time, and no waiting time, assuming an unlimited system size. Each simulation began with 4 fully-connected nodes. The arrival rate of super-peers is chosen to produce a desired average system size. This allows us to measure the performance of SUPS as a function of the system size. Simulation data are collected in steady state only after processing 10,000 join/leave events. Each data point is the average of 30 runs, using different initial random number seeds. In most of the simulations, we do not show the confidence intervals because they are less than 1% of the value measured.

5.2. Metrics

To evaluate the properties of the topology generated by the SUPS protocol, we measured the following five values:

- The *diameter* of the topology at any point of system duration is the longest shortest-path length between any pair of nodes in the system. This represents an upper bound on the search path length,

which is the primary factor for the scalability.[4] A small diameter means that query responses will be received quickly.

- The *eccentricity* of a node n is the longest shortest-path from n to any other node in the system; the maximum eccentricity over all nodes is the same as the network diameter. We measure the *average eccentricity* of the nodes to observe the possible variation in the search path length. Low eccentricity means that response times to queries due to path lengths will not vary greatly.
- The node *degree* represents the number of connections that must be maintained by that node. A small *average* degree results in a low network load, since each query message is flooded over each connection. A low degree *variance* results in better load balancing and an almost regular topology. That is, no super-peer or connection in the network will be significantly more congested than another.
- The *disruption rate* of the system is the number of nodes that are affected by a node leaving or joining. This reflects the maintenance cost caused by the dynamic behavior of the system.
- Finally, the *fault resilience* is measured by observing the difference between estimated values and actual values of N and δ. The estimation error of N and the estimation error of δ shows how well the system tolerates any failures.

5.3. Results

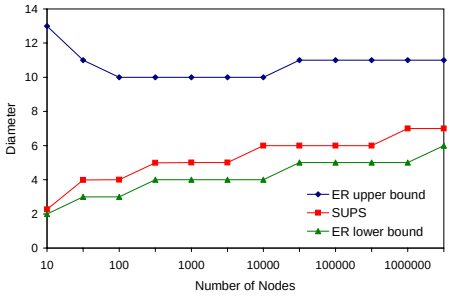

Figure 2. Diameter of the SUPS topology and the theoretical bounds of ER.

Figure 2 depicts the results of an experiment investigating the diameter of the topologies produced by SUPS, as a function of the system size.[5] These results are bracketed by the theoretical bounds of the ER model. The measured performance of SUPS is well within the bounds of the ER random graph model, and approaches the lower of those bounds. The network diameter grows sub-logarithmically as a function of the system size. The topologies produced by SUPS are clearly very scalable, since the network diameter is an upper bound on the search path length to reach *all* nodes in the system. In addition, all topologies produced were always connected, which is essential for correct function.

[4] In some other work, this distance would be called the maximum hop-count.

[5] Since this experiment investigated very large system sizes, there is only one measurement for each data point, and no confidence intervals are available.

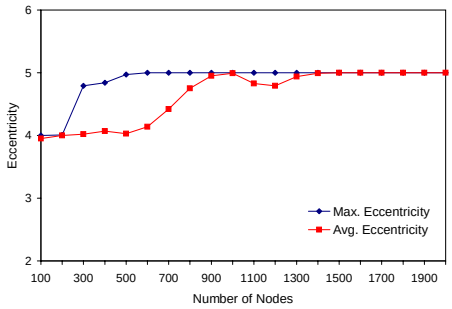

Figure 3. Max. eccentricity (Diameter) and avg. eccentricity of nodes in SUPS.

Figure 3 shows the average node eccentricity and maximum eccentricity (diameter) as a function of system size, for a much smaller range of sizes. This experiment demonstrates that the difference between average and maximum eccentricity is always less than 1. As a result, search path length from anywhere in the system is highly uniform.

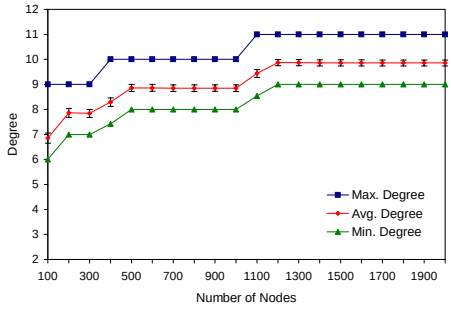

Figure 4. Degree distribution of SUPS.

Figure 4 shows the minimum, average, and maximum node degree, as a function of system size. The variance is also shown by the error bars. The very small variance implies that the topologies produced were very regular, as for random graphs in the ER model, and therefore network loads should be very evenly distributed. The logarithmic growth in degree means the topologies constructed attain a low diameter very efficiently.

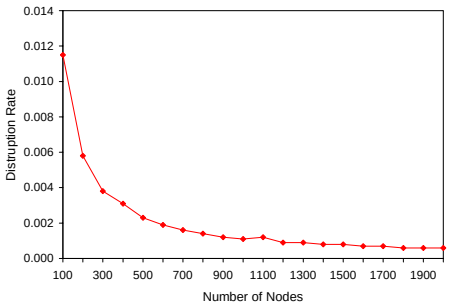

Figure 5. Disruption rate of SUPS. % of nodes affected by a node join/leave.

Figure 5 shows the disruption rate as a function of system size. The disruption rate is a measure of the cost of maintaining the network topology (i.e., the fraction of nodes that have to adjust their connections) as nodes join and leave the P2P system. The resulting rate is approximately equal to $1.14/N$. That is, an average of 1.14 nodes are affected by each node joining or leaving, regardless of the system size. This constant maintenance cost is extremely favorable for highly dynamic systems like P2P; scalable, efficient search performance is achieved with a low maintenance cost, rather than the high cost associated with structured P2P approaches.

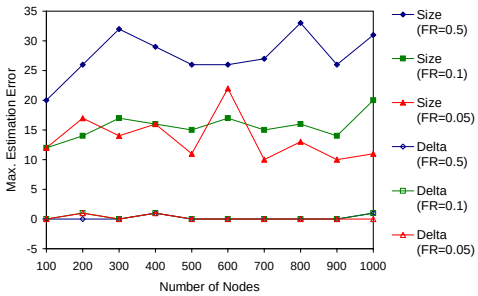

Figure 6. Fault resilience of SUPS. Max. estimation error of N and δ with varying fault rate of nodes.

The final experiment investigated the fault tolerance of the SUPS protocol; the results are shown in Figure 6. In this experiment, the primary nodes responsible for broadcasting information about node joins and leaves fail independently with probability FR, for FR = .05, .1, and .5 (5%, 10%, and 50%, respectively). The results show that the maximum error in the estimated value of N is low; more importantly, the maximum error in the estimated value of δ never exceeds 1. These results confirm that distributed estimation of N and δ works very well despite conditions that are highly challenging.

6. Comparison with Other Approaches

In this section we compare the performance of SUPS with two other P2P systems: Gnutella v0.6 and Gia. Each represents a different tradeoff of scalability, efficiency, simplicity, and robustness.

Gnutella v0.6 ([3], [4]) is a widely used unstructured P2P system with support for super-peers (termed "ultrapeers" in Gnutella). We compared with two versions of the Gnutella ultrapeer topology, one with 6 connections per ultrapeer, and one with 32 connections per ultrapeer, as proposed in [7]. The topology is constructed by a joining ultrapeer filling up $1/3$ of the maximum connections with the outgoing connections and the rest with the incoming connections. Each node then maintains the given number of connections using ping/pong when necessary. The difference in the diameter of these two choices, compared with SUPS, is shown in Figure 7 for varying network sizes. This figure shows that SUPS for a 10,000-node network has a diameter that is 50% smaller than Gnutella with 6 connections per ultrapeer, and with approximately twice number of connections. Compared with the Gnutella with 32 connections per ultrapeer, the diameter of SUPS is about the same, but with 33% fewer connection.

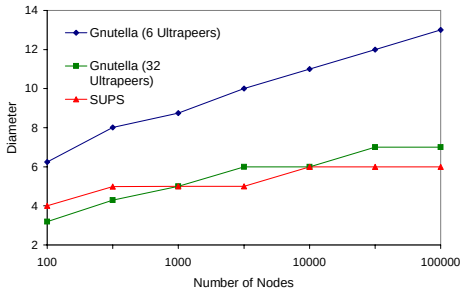

Figure 7. A comparison of SUPS and Gnutella.

Gia [10] is another recent approach to improve the performance of unstructured P2P systems. Gia advocates multiple levels of hierarchy, topology adaptation, flow control, index replication, and a biased random walk search procedure. It is an elegant and well-reasoned proposal. Gia was compared with a randomly-connected unstructured super-peer network, and shown to support a higher query load (by several orders of magnitude under some conditions). This is accomplished primarily due to the use of random walks (rather than flooding), and by exploiting the capacity differences of different nodes.

The advantages of SUPS versus GIA, however, include the following:

- Gia is much more complex than SUPS.

- SUPS is a proposal for a small change to existing P2P systems, affecting only the super-peer topology construction. Gia is much less compatible with existing systems and is therefore less likely to be adopted.
- Gia's performance depends on whether the matching data is found quickly. In the worst case, the random walk either (a) will give up without finding a match, or (b) may have to traverse a very long path (1-2 orders of magnitude longer than SUPS). Searches for data that don't exist will always trigger this behavior. SUPS guarantees data will be found if it exists, with very low maximum search path length (diameter) whether the data exists or not.

7. Conclusion

We presented a novel protocol for constructing a super-peer topology for unstructured P2P systems. This protocol is based upon randomly adding edges between super-peers in a way that maintains a nearly-uniform low node degree. We evaluated the performance of the method and showed that it approximates well the behavior of ER model random graphs. The resulting topology has the lowest diameter (maximum search path length) of any method presented to date ($\Theta(\ln N / \ln \ln N)$). At the same time, the topologies produced are low cost and almost regular. SUPS has a low maintenance cost for super-peers who join or leave the network. Our protocol is fully distributed and was shown to be robust to (a) rapid changes in the set of super-peers, and (b) failures in the super-peers.[6] Perhaps most importantly, our approach is relatively simple, compatible with currently-deployed P2P systems, and has all of the advantages of unstructured P2P systems.

Many ideas have been proposed to improve the performance and scalability of unstructured P2P systems. Some of these ideas are orthogonal to our approach and could be combined with the approach of SUPS.

We speculate that our method of constructing "random-like" graphs when the set of vertices is constantly changing may be useful for other applications that depend upon the properties of random graphs. We are investigating the generalization of our method for any such self-organized distributed systems.

[6]While it is desirable and normal to select super-peers that are highly reliable, we believe attention to robustness and fault tolerance in the face of failures or changes in network size will continue to be an important requirement for P2P systems, including super-peer overlays.

References

[1] Fasttrack. http://www.fasttrackdcn.net.

[2] Gnutella. http://www.gnutella.com.

[3] Gnutella Protocol Development. RFC-Gnutella 0.6. http://rfc-gnutella.sourceforge.net.

[4] Gnutella Protocol Development. Ultrapeers: Another Step Towards Gnutella Scalability. http://rfc-gnutella.sourceforge.net.

[5] Gnutella: The Gnutella Protocol Specification v0.4. http://www.limewire.com.

[6] KaZaA. http://www.kazaa.com.

[7] LimeWire. http://www.limewire.com.

[8] E. Adar and B. A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), October 2000. http://firstmonday.org/issues/issue5_10/adar/index.html.

[9] B. Bollobás. *Random Graphs*. Academic Press, London, 1985.

[10] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. In *Proceedings of the ACM SIGCOMM 2003 Conference*, 2003.

[11] Clip2 DSS. Gnutella: To the Bandwidth Barrier and Beyond. Technical report, Clip2 DSS, November 2000. http://lambda.cs.yale.edu/cs425/doc/gnutella.html.

[12] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of the ACM SIGCOMM 2002 Conference*, 2002.

[13] P. Erdös and A. Rényi. On Random Graphs I. *Publ. Math. Debrecen*, 6:290–297, 1959.

[14] 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*, 2002.

[15] 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)*, 2002.

[16] G. Pandurangan, P. Raghavan, and E. Upfal. Building Low-Diameter P2P Networks. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 2001.

[17] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. *IEEE Internet Computing Journal*, 6(1), 2002.

[18] A. Ruciński and N. C. Wormald. Random Graph Process with Degree Restrictions. *Combinatorics, Probability and Computing*, 1:169–180, 1992.

[19] 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)*, 2002.

[20] S. Vuong and J. Li. Efa: An Efficient Content Routing Algorithm in Large Peer-to-peer Overlay Networks. In *Proceedings of the3rd International Conference on Peer-to-Peer Computing (P2P 2003)*, 2003.

[21] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *Proceedings of the 19th International Conference on Data Engineering*, 2003.