Internet Mathematics Vol. 5, No. 3: 227–250

JumpNet: Improving Connectivity and Robustness in Unstructured P2P Networks by Randomness

J. Zich, Y. Kohayakawa, V. Rödl, and V. Sunderam

Abstract. We propose a self-organizing algorithm that maintains a high degree of connectivity and robustness in unstructured P2P networks. The algorithm is based on the "jump" primitive that periodically replaces overlay links at each node. The algorithm produces and maintains P2P networks that closely approximate sparse random graphs. Although the sparse random graphs have low degree, they are well connected and possess low diameter. This makes them good candidates for common P2P networks. Through a series of simulation experiments, we confirm that the jump algorithm leads to a network topology with these desirable properties if we start with an ad-hoc network. P2P networks conditioned through jumps also accomplish search and broadcast operations very effectively as compared to both simulated and real-life systems such as the Gnutella ultrapeer network.

1. Introduction

Peer-to-peer (P2P) networks and protocols have received considerable attention in recent years, and many instances of such networks are in widespread use. By far the most popular are implementations for file sharing; real and perceived deficiencies in each manifestation have led to a slew of alternatives that often incorporate architectural extensions. Well-known examples include Gnutella [Kirk 03], Gnutella2 [Gnutella 09], and Kazaa [Sharman 06].[1] In these

[1] Although other file sharing networks, like eDonkey, Overnet, and DirectConnect, are possibly more popular [Slyck 08], they cannot be considered peer-to-peer solutions because they are server based.

1542-7951/08 $0.50 per page

and other networks, file sharing protocols are built almost solely on one communication primitive: *search*. In popular P2P networks, this primitive is based on flooding queries, sometimes augmented by heuristics. Substantial work on more effective search has been undertaken by the research community, primarily on distributed hash tables (DHT) and related algorithms. The best-known DHT-based networks are Chord [Stoica et al. 01], Pastry [Rowstron and Druschel 01], Tapestry [Zhao et al. 01], Kademlia [Maymounkov and Mazieres 02], and Symphony [Bawa et al. 03]. However, such innovations have not been widely adopted in popular software systems—possibly due to skepticism towards overly complicated solutions and the fact that DHT-based systems are well suited only for a limited number of applications [Freedman and Mazieres 03, Ganesan et al. 03, Harvey et al. 03].

Another problem of the current P2P networks is that for reasons of pragmatics many of them, if not most, are less than "pure." For example, it is common for P2P nodes to be designated as "special" (e.g., ultra-peers or super-nodes). Moreover, P2P servent software often includes hard-coded bootstrap information, or some other indirect dependence on infrastructural support. Both these mechanisms, which are deviations from the P2P model, are used in popular systems, including Gnutella [Kirk 03], Gnutella2 [Gnutella 09], and Kazaa [Sharman 06]. While improving usability and performance, such deviations from strict adherence to the definition of P2P (a) dilute some of its advantages (e.g., vulnerability to localized failures or attacks) and (b) may not be possible in emerging application scenarios (e.g., spontaneous crowd collaborations or peer-assisted streaming multimedia distribution). Moreover, it was observed [Karbhari et al. 03] that the choice of the bootstapping method in the Gnutella network may significantly influence the subsequent performance of the peer.

Indeed, recently, there has been a surge of interest in the latter types of ad-hoc P2P networks. Vehicular ad-hoc networks are one instance [Holfelder et al. 07]; in certain settings, P2P communications can be used autonomously (without any associated infrastructure or fixed support) to enable a wide range of functions such as accident avoidance, alternate route planning in traffic jams, heuristic fleet scheduling among delivery trucks, and so on. Another example is the 2007 use, by the Swedish company TerraNet AB, of a mesh network of mobile phones permitting the routing of calls and data between participating handsets, without the need to involve cell towers [TerraNet 08, Draves et al. 04]. In these and other emerging situations, generating and maintaining a P2P network with "good" properties can be extremely beneficial to its operation.

In this paper, our main concern is connectivity and robustness of "pure" P2P networks targeted towards applications where there is little or no landline or in-

frastructural support. Traditional search-oriented P2P frameworks do not consider any broader meaning of "well connectedness," and connectivity in DHT-based architectures is constrained by the requirement of nodes to link to other according to a prescribed pattern. To enable the deployment of self-organizing P2P networks, we propose an algorithm that can be used *to create and maintain a well-connected and robust network topology without using any external support*. In particular, the P2P network should be robust and well-connected in the presence of high dynamicity, i.e., nodes joining and leaving frequently, and should possess low degree and small diameter. Our algorithm is based on the so-called *jump operation*, which periodically replaces overlay links at each node in a random fashion. We argue that the resulting network topology shares many properties with *random graphs*—i.e., "typical" graphs that are randomly generated. Random graphs have been thoroughly studied in graph theory (see, e.g., [Bollobás 01] and [Janson et al. 00]) and exhibit many useful properties: low average distance, low diameter, resilience to random failures, good connectivity, etc. The problem of building and maintaining an unstructured P2P network with provably good topology was tackled, for instance, in [Pandurangan et al. 01]; however, the method presented in that paper supposes the existence of a special node (a *host server*).

In order to validate these claims, besides presenting an analysis of a simplified model, we have performed a number of simulation experiments. The main network properties that we monitored were *average pairwise distance* between nodes and *resilience to random failures*. We observed that, after the execution of a few jump operations per node, not only does the average distance drop but it also has significantly smaller variance. Resilience to failures was measured via the average component size and the largest component size after the random removal of a specified number of connections. In order to show that typical network operations will also benefit from the jump operations, we have tested the performance of a lookup and a broadcast primitive. In the case of lookup, we observed a significant increase in hits, and in the case of broadcast, the total time to reach a target number of nodes also rapidly decreased.

This paper is organized as follows. In Section 2, we present the formal description of the jump operation. Section 3 presents a mathematical analysis of a simplified model that supports the claim that, by repeating jump operations, the starting graph converges to a random-looking graph in time proportional to the logarithm of the number of nodes in the network. The simulations and experimental results supporting the theoretical claims are described in Section 4. Finally, in Section 5, we briefly discuss directions for future work.

2. The Jump Algorithm

We start with an informal description of our algorithm. We suppose that we have an overlay network of connected nodes and that each pair of connected nodes can symmetrically communicate over their connection. Nodes maintain lists of first and second immediate overlay neighborhoods. Since the network is dynamic (because nodes join and leave, and because of the nature of the jump operation), we assume that each node maintains its list of first and second neighbors or updates the list before each jump. We also suppose that each node has the ability to drop any of its overlay connections and to establish new overlay connections based upon the identity of a target node. In what follows, we use standard graph theoretic terms to model our networks.

The jump operation is performed by each node independently. In general, the purpose of the jump performed by a node v is to replace as many of its current neighbors as possible by a subset of its second-order neighbors. Clearly, the candidates for becoming neighbors of v must be willing to accept the new connection to v. In fact, an important feature that we require of our network is that all nodes have bounded degree, and this requirement may prevent a vertex from accepting such a new connection. Therefore, first v asks the second-order neighbors that have the lowest degrees whether they are willing to accept a connection. If all of them are already saturated, v asks a random second-order neighbor to drop a connection. The connection for removal is chosen such that no node gets disconnected from the network. The jumping node v can assure this, because it keeps the list of the first and second neighbors, and therefore it is able to identify connections that form cycles among them. By eliminating short cycles we improve the throughput of the network and simultaneously keep the degrees bounded.

2.1. The Algorithm

The formal description of the jump operation is as follows. Suppose we have a network G. If H is a subnetwork of G and v is a node in H, then let $N_H(v)$ be the neighborhood of v in H.

1. *Collecting phase.* Node v collects all its second-order neighbors into a temporary list L. If $|L| < d$, node v also appends to L its third-order neighbors up until $|L| = d$ (if $|L| < d$ even after the third-order neighbors are added, the algorithms proceeds with this shorter list L). While building the list L, node v also associates to any $v' \in L$ the following two pieces of information:

(a) An arbitrary node, adjacent to v, lying on a shortest path joining v and v'; we denote this node by *first*(v').

(b) Let T be a spanning tree of the subgraph F induced by $\{v\} \cup N(v) \cup L$, containing all edges incident to v. Then v associates with v' the subset of neighbors of v' that are not adjacent to v' in T; formally, we define this set as $C(v') = N_F(v') \setminus N_T(v')$.

2. *Jumping phase.* Node v repeats the following steps:

(a) Select uniformly at random a node $v' \in L$.

(b) If $\deg v' \geq d$, then do:

i. If $C(v') = \emptyset$, go to step (e).

ii. Select any $u \in C(v')$.

iii. Drop the edge between u and v'.

iv. Remove u from $C(v')$.

v. If $\deg v' \geq d$, go to step (i).

(c) Establish an edge between v and v'.

(d) Drop the edge between v and *first*(v').

(e) Remove v' from L.

(f) If $L = \emptyset$ or $\deg v \geq d$, quit the loop.

For our experiments presented in Section 4, we used a discrete-event simulator and we treated the jump operation as a single event. However, in real-world scenarios, the algorithm must ensure that no node is disconnected from the network. Therefore, for instance, all disconnections should be preceded by establishing new connections. However, we believe that collisions would rarely lead to a separation of a node if one chooses a proper timing for jumps. To illustrate this, let us assume that the average degree in the network is d and that Δt is the time needed to accomplish a single jump. Since the jump operates only in the first and second neighborhoods, in order to avoid "overlapping jumps," a node should not jump more often than once per a time interval of length $d^2 \Delta t$.

Besides the synchronization issue of the jumps themselves mentioned in the previous paragraph, the jumps must also be synchronized with the other network operations. A more detailed discussion about this issue is presented in Section 4.6, after we outline some other network primitives.

2.2. Motivation behind the Algorithm

The motivation of the jump algorithm is derived from the theory of random walks on graphs. We observed that we can utilize the theory of random walks from two different perspectives. Both justify our theoretical hypothesis. The first one is explained in the remaining few paragraphs in this section while the second one is used for a more rigorous analysis in Section 3. The first approach considers random walks directly on the given network and gives a fairly intuitive explanation of why the jumps transform any given starting graph to a random-looking graph. We show that the jumps can be roughly interpreted as "boosted" multiple random walks. However it does not provide good bounds on the number of jumps needed to achieve this "randomness." The second approach studies random walks on a "supergraph" of all graphs. At the cost of certain simplifications of the considered model, this allows us to prove that one reaches the desired state of randomness much faster. Now we discuss the first theoretical justification.

Given a graph G and a vertex v in G, the *random walk on G started at v* is a sequence of vertices of G corresponding to a path of a walker who starts at v and, at every time step, randomly chooses, among the neighbors of the currently visited vertex, the next vertex to visit.[2] Given vertices v and u, we let $P_t(v,u)$ be the probability that a random walk started at v will be at u at time t.

In the context of Markov chains, a random walk on a non-bipartite and connected graph is ergodic, i.e., there exists a stationary distribution of probability to which the random walk (starting at any vertex) converges. The stationary distribution is $\pi(u) = d_u/\operatorname{vol} G$, where $\operatorname{vol} G$ is the sum of the degrees d_v ($v \in V(G)$) of G. This is a classical result, previously proven (for instance, in [Lovász 96] or [Chung 97]). Therefore, for a random walk starting at v and $u \in V(G)$, we have

$$\lim_{t\to\infty} P_t(v,u) = \pi(u) = \frac{d_u}{\operatorname{vol} G}. \tag{2.1}$$

In particular, when the graph is regular (all the vertices have the same degree), every node u is, asymptotically as $t \to \infty$, equally likely to be visited at time t.

This result can be used to obtain a random graph by sending d random walkers from each node v, stopping them after some time, and connecting v to the nodes at which they have stopped. Therefore, if G is regular, formula (2.1) guarantees that these d new nodes are chosen essentially uniformly, provided that t is sufficiently large. We thus essentially obtain a random graph from the so-called d-out model [Bollobás 01].

[2]For technical reasons, the walker may also stay at the currently visited vertex with some fixed, positive probability.

In general, many advanced techniques are known for estimating the speed of convergence of (2.1) [Diaconis and Saloff-Coste 96, Chung 97, Aldous and Fill 02], but they require additional information about the given graph. Since, in fact, we target any starting network topology (it may be a wireless network, local area network, etc.), we cannot directly apply any of these techniques. However, it is known that, for an arbitrary regular graph G on n nodes, one needs to let t be of order n^3 in order to at least hit every node of the graph [Feige 95]. Since n can be of the order of thousands or more and the network may constantly change, this prevents this approach from being of practical use.

To obtain a randomly behaving graph more quickly, we let all nodes cooperate in such a way that several nodes participate on the construction of a "boosted-up" random walk. One may describe the "boost-up" mechanism in the following simplified way: suppose we let a random walker start at each node of the network at time $t = 0$, and suppose that our walkers take their steps in a synchronized fashion (one step per unit of time). At time $t = 2$, most of the walkers will have moved two steps away from their starting point (some returned to their starting point). Suppose that we add to our graph the edges joining the starting point of the walkers to their current location. Note that such an edge is "worth two edges" (such an edge replaces a walk of length two). If we iterate this process, when a walker takes a (single) step along a "new edge," she will actually be taking several steps in the original graph. The cascading effect that this process gives is what the jumps exploit.

3. Jump Algorithm Analysis

In this section we present two mathematical analyses of the jump algorithm, and we show that jumps lead to a network topology with good characteristics—i.e., low average distance, low diameter, resilience to random failures, and good connectivity. In the introduction we noted that a graph generated randomly will have almost surely the desired properties. Suppose that our starting graph before jumps is G_0 and has r edges, and let G be *any graph* with r edges. In this section we show that the probability of turning G_0 into G by a sequence of jumps is approximately the same for any such G if the number of jumps is reasonably large. This will mean that jumps will generate the uniform probability distribution on the space of all graphs with r edges, and since almost all of them carry the required properties, the process will likely end up with a good network topology.

Therefore, the crucial question is, "how many jumps are needed to achieve this?" The first result states that $O(\log n)$ jumps per node is enough. In this result,

we are content with achieving a distribution that is statistically close to the uniform distribution. The second result gives $O\big((\log n)^2\big)$ jumps per node, but the deviation from the uniform distribution is controlled pointwise (and hence the most likely and the least likely graphs have about the same probability).

In order to make the analyses feasible, we simplify the jump operation. One of the consequences of this simplification is that the graph before and after each jump has the same number of edges; let this be r and let d be the average degree, i.e., $d = 2r/n$. A *simplified jump* is defined as follows: during a simplified jump operation, an edge and a non-edge are selected, and the edge is deleted and the non-edge becomes an edge. Notice that the normal jump corresponds to several simplified jumps (typically d). Moreover, if several jumps occur at about the same time in different locations of the graph, then several new edges are introduced and several edges are dropped independently. Therefore, we can view this group of jumps as a sequence of simplified jumps. The key simplifying assumption that we shall make from now on is that, *in our simplified jumps, the edge and the non-edge are chosen uniformly at random.*

The analyses are based on identifying our problem with a random walk on r-sets (sets having r elements), known as the *Bernoulli-Laplace diffusion model* [Diaconis and Saloff-Coste 96, Diaconis and Shahshahani 87]. Under this identification we view our graphs as sets chosen from the set of all possible $\binom{n}{2}$ edges. The process of swapping edges and non-edges in the simplified jump operation corresponds exactly to the evolution of the r-sets in the Bernoulli-Laplace diffusion model. In the rest of Section 3 we keep the notation $m = \binom{n}{2}$ and $r = dn/2$.

3.1. Total Variation Distance

Diaconis and Shahshahani studied the speed of convergence of our random walk in terms of the *variation distance* [Diaconis and Shahshahani 87]. Let $P_k(R, S)$ be the probability of going from an r-set $R \subset [m] = \{1, \ldots, m\}$ to an r-set $S \subset [m]$ in k steps. Let π be the uniform probability distribution on $\binom{[m]}{r}$. The variation distance between $P_k(R, \cdot\,)$ and π is defined as

$$\Delta_{\mathrm{VD}}(k, R) = \max_{\mathcal{E} \subseteq \binom{[m]}{r}} |P_k(R, \mathcal{E}) - \pi(\mathcal{E})|.$$

A standard argument shows that the variation distance is just half of the l_1 distance, i.e.,

$$\Delta_{\mathrm{VD}}(k, R) = \frac{1}{2} \sum_{S \in \binom{[m]}{r}} |P_k(R, S) - \pi(S)|, \tag{3.1}$$

where the maximum and the sum are taken over all r-sets. In the context of our setup, the main theorem of Diaconis and Shahshahani states the following.

Theorem 3.1. [Diaconis and Shahshahani 87] *There is an absolute constant a such that*

$$\text{if } k \geq \frac{dn}{2}(\log n + c), \text{ then } \Delta_{VD}(k, R) \leq ae^{-c}, \tag{3.2}$$

where R is any starting r-set.

Suppose now that we execute a total of k simplified jumps, starting from an arbitrary graph. How large does k have to be so that we are close to the uniform distribution? Let us see what a direct application of Theorem 3.1 gives. The main problem is that the variation distance in (3.1) is a sum, and thus it does not allow any direct control over each particular summand. It could happen that one of the summands is significantly larger than the others, which would mean that one particular r-set is substantially more likely to be reached than the others. For our purposes, this is, of course, undesirable. In order to avoid this problem, one has at least to choose c such that the right-hand side of (3.1) is smaller than $1/\binom{m}{r}$ (the probability of an element in the uniform space). Using (3.2), we obtain c of order $r \log m$. In our model of simplified jumps, this would mean that we need $k = \Omega(n^2 \log n)$ simplified jumps in total, yielding the bound of $\Omega(n \log n)$ simplified jumps per node.

The result below, which follows from Theorem 3.1, tells us that if we give up on a small proportion of the space, then we can guarantee small relative error after $O(d \log n)$ simplified jumps per node.

Proposition 3.2. *After a total of $k = \Omega(dn \log n)$ simplified jumps, we have*

$$\frac{P_k(R, S)}{\pi(S)} = 1 + o(1) \tag{3.3}$$

for $(1 - o(1))\binom{m}{r}$ r-sets S.

The following technical lemma is enough to prove Proposition 3.2.

Lemma 3.3. *Let $l \geq 1$ and d be real constants, $k = d\,ln \log n$, $j_0 = l\frac{\log n}{\log \log n}$, and $j \leq j_0$. Then, for large enough n, the probability that, after k steps, the random walk is at a set that intersects R on j elements is $p_j = \binom{r}{j}\binom{m-r}{r-j}/\binom{m}{r}$, up to an additive error of at most $O(1/n^l)$. Moreover, $n^l p_j \to \infty$ as $n \to \infty$, and hence this additive error is negligible in comparison with p_j.*

Let us briefly sketch how to obtain Proposition 3.2 from Lemma 3.3.

Proof of Proposition 3.2 (Sketch). First notice that there are exactly $\binom{r}{j}\binom{m-r}{r-j}$ r-sets that share j elements with a given r-set. Since our random walk does not distinguish

among the r-sets that have the same intersection size (with the given r-set R), we desire to prove that the probability of hitting a set with j common elements with R is close to $\binom{r}{j}\binom{m-r}{r-j}/\binom{m}{r}$, with a reasonably small *relative* error. This is exactly the claim in Lemma 3.3. However, this cannot be proven for all possible values of j and thus the upper bound j_0 for j is introduced. Note that $j_0 \to \infty$ as $n \to \infty$, but the expected size of the intersection of a random r-set with R is $r^2/m = \Theta(d^2/2)$, which is a constant for our setup. Since the majority of r-sets intersect R in a number of elements that is close to the expectation, we obtain Proposition 3.2. □

Now we proceed to the proof of Lemma 3.3. Let us recall that the main purpose of the proof is to show that the deviation from the uniform distribution after k steps is small for j in the range given in the statement.

Proof of Lemma 3.3. First, we group the terms in the sum of the variation distance in (3.1) according to the size of the intersection with the starting set R. In that way, (3.1) becomes

$$\begin{aligned}\Delta_{\mathrm{VD}}(k,R) &= \frac{1}{2}\sum_{j=0}^{r}\sum_{\substack{S\in\binom{[m]}{r}\\ |S\cap R|=j}} |P_k(R,S)-\pi(S)| \\ &= \frac{1}{2}\sum_{j=0}^{r}\left|\widetilde{P_k}(j) - \binom{r}{j}\binom{m-r}{r-j}\Big/\binom{m}{r}\right|,\end{aligned}$$

where $\widetilde{P_k}(j)$ is the probability of being at a set that has intersection of size j with R after k steps. We first show that, for $j \le j_0 = l\frac{\log n}{\log\log n}$, the hypergeometric term $p_j = \binom{r}{j}\binom{m-r}{r-j}/\binom{m}{r}$ is asymptotically larger than $1/n^l$, that is,

$$p_j n^l \to \infty \text{ as } n \to \infty. \tag{3.4}$$

To prove (3.4), we observe that

$$p_j = \binom{r}{j}\binom{m-r}{r-j}\Big/\binom{m}{r} > \binom{r}{j}\left(p-\frac{j}{m}\right)^j\left(q-\frac{r-j}{m}\right)^{r-j} \stackrel{\text{def.}}{=} E_1,$$

where $p = r/m$ and $q = 1-p$. The right-hand side E_1 can be further rewritten as

$$\begin{aligned}E_1 &= \binom{r}{j}\left(\frac{r-j}{m-2r+j}\right)^j\left(1+\frac{-2r+j}{m}\right)^r \\ &\ge \binom{r}{j}\left(\frac{r-j}{m-2r+j}\right)^j\left(1-\frac{2r}{m}\right)^r \stackrel{\text{def.}}{=} E_2.\end{aligned} \tag{3.5}$$

Since in our setup $r/m = \Theta(1/r)$, the last term in (3.5) is bounded from below by a constant, say B. In the next step we also replace r and m by the corresponding expressions in terms of n and d:

$$E_2 \geq B \binom{\frac{dn}{2}}{j} \left(\frac{\frac{dn}{2} - j}{\binom{n}{2} - dn + j} \right)^j \geq B \left(\frac{dn}{2j} \right)^j \left(\frac{\frac{d}{2} - \frac{j}{n}}{\frac{n-1}{2} - d + \frac{j}{n}} \right)^j$$
$$\geq B \left(\frac{dn}{2j} \right)^j \left(\frac{d}{n} \right)^j \left(1 - \frac{2j}{dn} \right)^j \geq B \left(\frac{d^2}{2j} \right)^j \left(1 - \frac{2j^2}{dn} \right) \stackrel{\text{def.}}{=} E_3.$$

Finally, we show that for $j \leq j_0 = l \frac{\log n}{\log \log n}$ the quantity E_3 is asymptotically larger than $1/n^l$. Since we have $j \leq j_0$, the last expression in E_3 is of the form $1 - o(1)$. Hence, in order to prove our claim, we need to verify that

$$B \left(\frac{d^2}{2j} \right)^j (1 - o(1)) \gg \frac{1}{n^l}.$$

Since B and d are constants, this is equivalent to showing that $n^l \gg j^j$, which is the same as showing that

$$l \log n - l \frac{\log n}{\log \log n} (\log l + \log \log n - \log \log \log n) \to \infty.$$

Clearly, this is the same as

$$l \frac{\log n}{\log \log n} (\log \log \log n - \log l) \to \infty,$$

which is true, and hence (3.4) is proved.

It now remains to show that, after $k = dln \log n$ steps, the probability that the random walk is at a set that intersects R on j elements is p_j, up to an additive error of at most $O(1/n^l)$. Clearly, it suffices to check that $\Delta_{\text{VD}}(k, R) = O(1/n^l)$. However, for such a value of k, we have $c = (2l - 1) \log n$ in (3.2). We obtain from (3.2) and the fact that $l \geq 1$ that

$$\Delta_{\text{VD}}(k, R) \leq ae^{-c} = \frac{a}{n^{2l-1}} = O\left(\frac{1}{n^l} \right).$$

This completes the proof of Lemma 3.3. □

We conclude this section with an illustrative numerical example. From the proof of Theorem 3.1 [Diaconis and Shahshahani 87], it can be deduced that for $n \geq 1000$ the constant a in (3.2) is essentially 2 (more precisely, it is not greater than $2 + 0.5 \times 10^{-9}$). We consider the setup $n = 1000$ and $d = 5$. Table 1 shows

j	% of graphs	j	% of graphs
5	0.921	13	10.947
6	1.933	14	9.780
7	3.474	15	8.149
8	5.457	16	6.360
9	7.614	17	4.668
10	9.554	18	3.233
11	10.890	19	2.120
12	11.369	20	1.319

Table 1. Number of average degree-5 graphs ($n = 1000$) grouped by number of intersecting edges with the starting graph, j.

the number of graphs with $r = dn/2 = 2500$ edges grouped according to j, the number of common edges with the starting graph R. Thus, approximately 98% of the graphs have from 5 to 20 common edges with R. Setting $c = \log n$ in (3.2) yields $k \geq 1000 \times 33$, while the error in (3.2) is $ae^{-c} \doteq 0.002$. Therefore, according to Table 1, we can conclude that, for instance, the graphs with 12 common edges, which form 11.369% of all graphs with 2500 edges, are reached after 33 jumps per node with probability $(11.369 \pm 0.2)\%$.

3.2. Relative Pointwise Distance

Another bound can be obtained directly from a paper by Diaconis and Saloff-Coste in which they derive a general upper bound for the relative *pointwise distance* involving the *log-Sobolev constant* and the *spectral gap* [Diaconis and Saloff-Coste 96]. The relative pointwise distance after the kth step of the walk is defined as

$$\Delta_{\mathrm{RPD}}(k) = \max_{R,S \in X} \left| \frac{P_k(R,S)}{\pi(S)} - 1 \right|.$$

Let $\lambda_\star$ be the spectral gap of this random walk, i.e.,

$$\lambda_\star = \min\{1 - \beta_{\max}, \beta_{\min} + 1\},$$

where $\beta_{\min}$ and $\beta_{\max}$ are the smallest and the largest eigenvalues of the *transition matrix* P, respectively, i.e., the matrix of dimension $\binom{m}{r} \times \binom{m}{r}$ with entries $P_1(R,S)$ for all pairs of r-sets (R,S). The eigenvalues of P for Bernoulli-Laplace diffusion can be explicitly computed—it is well known (see, for instance, [Diaconis and Shahshahani 87]) that

$$\lambda_\star = \frac{m}{r(m-r)}. \tag{3.6}$$

The definition of the log-Sobolev constant is easier to introduce in the more general setup of an arbitrary graph. Let G be a graph, let d_x be the degree of

a vertex x, let $P_1(x,y) = 1/d_x$ be the probability of going from x to y (in one step), and let $\pi(x) = d_x/\operatorname{vol} G$ be the stationary distribution of the random walk on G. Then, the log-Sobolev constant α of G is defined as

$$\alpha = \inf_f \frac{\mathcal{E}(f)}{\mathcal{L}(f)},$$

where the infimum is taken over all functions $f : V(G) \to \mathbb{R}$ with $\mathcal{L}(f) \neq 0$, and

$$\mathcal{E}(f) = \sum_{P_1(x,y)>0} (f(x) - f(y))^2, \qquad \mathcal{L}(f) = \sum_{x \in V(G)} f(x)^2 \pi(x) \log \frac{f(x)^2 \operatorname{vol} G}{||f||_2^2},$$

with

$$||f||_2 = \left(\sum_{y \in V(G)} f(y)^2 \pi(y) \right)^{1/2}.$$

In general, it is hard to find bounds for log-Sobolev constants, even for special graphs. According to the next result, lower bounds are more important for estimating the mixing rate of random walks.

Theorem 3.4. [Diaconis and Saloff-Coste 96] *Let P be the transition matrix of a random walk on a graph H with the stationary probability π. Let $\pi_\star = \min_{x \in V(H)} \pi(x)$. Then if $c > 0$ and*

$$k \geq \frac{1}{4\alpha} \log \log \frac{1}{\pi_\star} + \frac{c}{\lambda_\star} + 1, \tag{3.7}$$

then

$$\Delta_{\mathrm{RPD}}(k) \leq (1 + 2e^2)\, e^{-c}. \tag{3.8}$$

Diaconis and Saloff-Coste also obtained a lower bound for the log-Sobolev constant of the random walk on r-sets [Diaconis and Saloff-Coste 96].

Theorem 3.5. [Diaconis and Saloff-Coste 96] *The log-Sobolev constant of the Bernoulli-Laplace random walk satisfies*

$$\alpha \geq \frac{m}{3r(m-r)\log m}. \tag{3.9}$$

Now we apply the previous two theorems in order to obtain a bound on the relative pointwise distance on our random walk. Formula (2.1) gives the general

formula for the stationary distibution of a random walk on a graph. Since the graph $\mathcal{G}$ is regular with degree $r(m-r)$, we have

$$\pi_\star = \pi(G) = \frac{1}{\binom{m}{r}}, \text{ for any } G \in V(\mathcal{G}). \tag{3.10}$$

Now we are ready to estimate the number of steps given by (3.9) just in terms of m and r. For this, we substitute (3.9), (3.6), and (3.10) into the right-hand side of (3.7), which yields

$$\begin{aligned}\frac{1}{4\alpha}\log\log\frac{1}{\pi_\star}+\frac{c}{\lambda_\star}+1 &\le \frac{3r\log m}{4}\left(\frac{m-r}{m}\right)\log\log\binom{m}{r}+cr\,\frac{m-r}{m}+1\\ &\le \frac{3r\log m}{4}\log\log\binom{m}{r}+cr+1\\ &\le \frac{3r\log m}{4}\log(r\log m)+cr+1\\ &= \frac{3}{4}\,r\log m\log r+\frac{3}{4}\,r\log m\log\log m+cr+1, \end{aligned}\tag{3.11}$$

where we have used the fact that $(m-r)/m \le 1$ and $\binom{m}{r} \le m^r$. Using $m \le n^2$ and $r = dn/2$, the right-hand side of (3.11) can be further bounded from above by

$$\begin{aligned}3r(\log n)^2+\frac{3}{2}r\log n\log(2\log n)+cr+1 &\le 3r(\log n)^2+3r\log n\log\log n+cr+1\\ &= \frac{dn}{2}\big(3(\log n)^2+o((\log n)^2)+c\big).\end{aligned}$$

Therefore, if we choose $k \ge 2dn\big((\log n)^2+c/4\big)$, then k will satisfy (3.7) and the relative pointwise error will be at most $(1+2e^2)\,e^{-c}$, according to (3.8). This means that $2d\big((\log n)^2+c/4\big)$ simplified jumps *per node* ensure that every average degree-d graph is reached equally likely up to that relative error. We may thus obtain (3.3) for *every* r-set S with $O(d(\log n)^2)$ simplified jumps per node.

4. Experimental Results

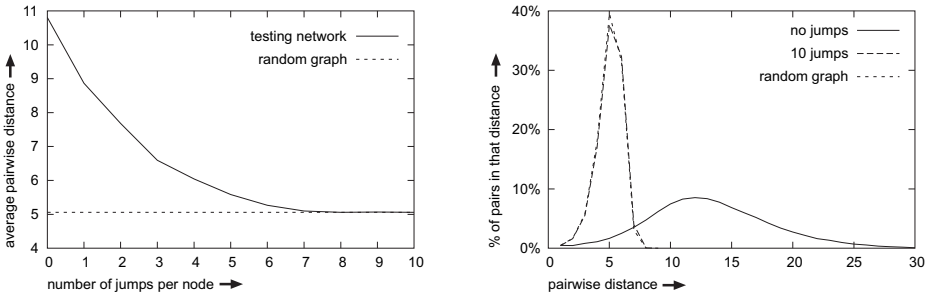
In order to validate our claims we performed a number of discrete-event simulation experiments on an overlay network $\mathcal{O}_\Delta$ of maximum degree $\Delta = 5$, which was built on top of an underlying network $\mathcal{U}$ with 1000 nodes. The network $\mathcal{U}$ was obtained using the following algorithm. First, we generated a random tree on 100 nodes. To this tree 10 edges were randomly added to form a few

cycles. Finally, 900 pendant edges were added to the resulting graph at locations selected randomly from the original 100 nodes, where the choices are made independently and uniformly. This underlying network $\mathcal{U}$ is representative of typical, fixed wireline computer networks. Nodes enter the overlay network $\mathcal{O}_\Delta$ in random order. When a node v wishes to join (or "wakes up"), it browses its first, second, and possibly third neighborhoods in the underlying network $\mathcal{U}$ (closely resembling the well-known expanding ring multicast method, with time to live (TTL) = 3). It looks for nodes that are already part of $\mathcal{O}_\Delta$ but are not yet saturated (i.e., their degrees in $\mathcal{O}_\Delta$ are strictly less than Δ). Among these nodes, v selects as many as possible, but not more than Δ, and connects to them. In that process, the closer nodes have preference.

In our experiments, this procedure typically resulted in an overlay network $\mathcal{O}_\Delta$ that was almost Δ-regular. Since the experiments with jumps were performed on the overlay network with $\Delta = 5$, the parameter d in the jump algorithm was also set to 5. Note that the jump algorithm will therefore retain the average degree.

4.1. Average Distance

The results of the simulations concerning average distance are summarized in Figure 1. The distance between a pair of nodes is defined as the length of the shortest path between them. In Figure 1(a), we show a chart that illustrates how the average distance in $\mathcal{O}_\Delta$ decreases as more jumps are performed. The horizontal axis denotes the number of jumps per node. For comparison, the chart also includes the expected average distance in the random graph with the same average degree. It is clearly visible that the average distance of our graph approaches the expected average distance of the random graph.



(a) (b)

Figure 1. Measuring the average distance before and after the jumps: (a) the average distance versus jumps and (b) histogram of distances.

Figure 1(b) shows the histograms of distances before and after ten jumps per node. Again, for comparison, the histogram for the random graph is presented. The random graph histogram is so similar to the histogram after jumps that they are barely distinguishable.

From these results we observed that for our experimental setup (degree 5 and 1000 nodes) approximately after ten jumps per node, the average distance stabilizes very near the expected value for the random graph. Because of this similarity, one may expect that the graph after jumps shares other properties of the random graph. Therefore, all other subsequent experiments compare the measured parameters before and after ten jumps per node. Note that this assumption is also supported (at least in order of magnitude) by the result in Section 3, which claims that after 33 jumps per node we are close to having a genuine random graph (see the last paragraph of Section 3.1).

4.2. Resilience to Failures

In the following set of experiments we observed whether the jumps lead to better connectivity by measuring the number and the sizes of the components after removing a certain portion of connections.

The steps of the experiments were as follows. First, after the overlay network $\mathcal{O}_\Delta$ is formed, the prescribed number of edges is randomly removed. In addition, we remember the removed edges for a later step. Next, we find the largest component. This captures the situation without jumps. After that, we return the removed edges to achieve the same configuration as in the beginning. Finally, each node performs jumps (ten per node), and we remove randomly the same number of edges as before and count the components. Note that the jumps change the network in a random fashion. This implies that it is not possible to remove exactly the same edges as before. In order to obtain statistically sound results, we repeat the experiment 10,000 times and take the average.

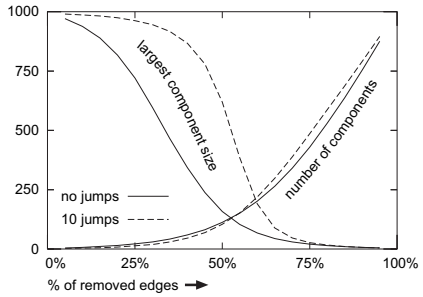


Figure 2. Simulation of random failures, showing resilience to failures.

The results are presented in Figure 2. The horizontal axis shows the percentage of removed edges: we removed 5%, 10%, 15%, ... , 95%. The edges were not gradually removed by 5%; instead, each time we started from $\mathcal{O}_\Delta$. The decreasing curves represent the largest component sizes, while the increasing ones show the number of components.

The experiment clearly demonstrates the better resilience of the network after the jumps, which, for example, even after the deletion of 50% of the edges maintained the largest component of size 700. The removal of the same number of edges from the original $\mathcal{O}_\Delta$ led to a disconnected graph whose largest component had size 200. Although the jumps kept the largest component big, they did not succeeded in decreasing the number of components, but the difference is not significant.

4.3. Searching

One of the most popular network operations in today's real-world P2P networks is searching. Therefore, we implemented a simple searching primitive and compared its performance in the network $\mathcal{O}_\Delta$ before and after jumps. Our implementation closely resembles the original Gnutella search, which is based on the flooding paradigm. The node initiating a search contacts its neighbors with the search query, they in turn contact their neighbors, and this process continues until it reaches a prescribed number of hops (or TTL hits zero). In order to avoid looping, duplicate search queries are dropped.

The experiment depended on two parameters: k, called proliferation, and TTL. After $\mathcal{O}_\Delta$ is formed, 100 random subsets of nodes, each of size k were generated. The nodes of the ith subset were assigned the same object—in our case the integer $i \in \{1, \ldots, 100\}$. Since the sets were not necessarily disjoint,

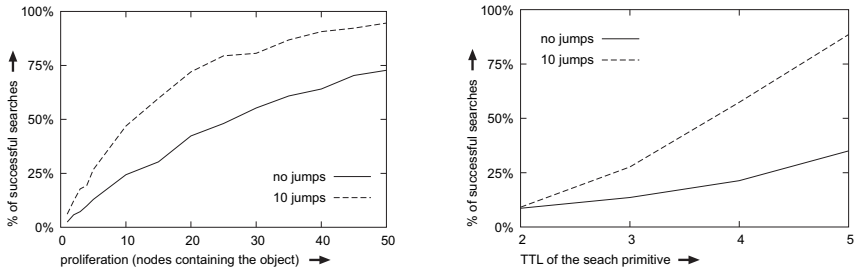


Figure 3. Simulation of a search primitive based on flooding: (a) dependence on proliferation (with TTL = 3) and (b) dependence on TTL (with $k = 5$).

one node could have been assigned several different integers. After this initial procedure, each node tried to find all 100 objects with a prescribed TTL. We measured the *hitrate*—the ratio of searches when a node locates at least one occurrence of an object to the total number of attempts. Finally, each node jumped ten times and the search and measurements were repeated.

Figure 3 summarizes the results of this experiment. Figure 3(a) shows the dependance of the hitrate on the proliferation. The parameter TTL was set to 3. Figure 3(b) shows the dependance of the hitrate on TTL. The proliferation k was set to 5, i.e., 0.5% of nodes contained the object. Both results demonstrate a visible improvement in the network after the jumps. From the first figure it follows that if an object is contained by 20 nodes (i.e., 2% of nodes), the hitrate of searches increases by jumps from 40% to 70%. The second figure demonstrates that the diameter of the graph after the jumps shrinks to nearly 5, which is also close to the expected diameter of the random graph with the same parameters.

4.4. Broadcasting

To measure the throughput of a network modified by jumps, we designed and developed a simple broadcast primitive called *numcast*. The technical details of numcast resemble the search primitive presented in Section 4.3; however, the goals of numcast and search differ. The purpose of numcast is to deliver a copy of a given message to a given number of nodes. The number of messages is called the *quota*.

If a node v initiates a numcast, it distributes the quota evenly among its neighbors. The neighbors continue recursively. If a node receives the request to forward a certain quota of messages, it distributes them evenly between all its neighbors, except the one that sent the request. To avoid looping, each node remembers for each message the first node that requested to forward that message. This node is called the *parent node* with respect to the given message m and node v and is denoted by $p(v, m)$. In this way each node fulfills only requests issued by its parent and refuses requests from other nodes. If a node $w \neq p(v, m)$ is refused by v, it uses the remaining neighbors or eventually routes the quota back to its own parent. This process builds virtually a tree (corresponding to one message); therefore, it is guaranteed that all copies are eventually delivered (if there are enough nodes). The root of this tree is the node initiating the numcast, and the length of a path traveled by a copy of the message from the root is called its *number of hops*.

We were mainly interested in the speed in which the message spreads. We measured this by recording the number of delivered copies of one message in each hop (from the node that originally disseminated the message). The result

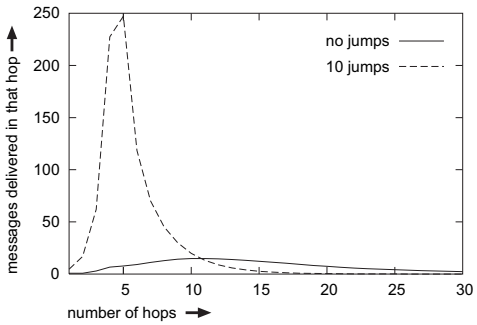


Figure 4. Simulation of a broadcast primitive (to 500 nodes).

of this measurement is depicted in Figure 4. The horizontal axis is the number of hops, while the vertical is the number of newly delivered copies. In order to exclude possible deviations, each node in the experiment initiated ten numcasts. The figure shows that the messages spread significantly fast and that it took ten hops to deliver the "slowest" message. Although the results in Section 4.3 demonstrated that the diameter of the graph decreases close to five after jumps, the then hops needed to complete the numcast is caused by returning and redistributing the messages.

4.5. Simulations on Gnutella

By an analysis of the modern Gnutella protocol, we realized that the network of Gnutella ultrapeers may already be random enough. We were led to this conclusion, because the Gnutella nodes used during the bootstrap procedure a large cache of nodes built in the previous live period. From this list, which may contain several thousands of nodes, a node typically selects a random subset of 20 to 30 nodes and connects to them.

In order to perform experiments on the Gnutella ultrapeer network, we developed a network crawler. We used the crawler several times with the following results. The crawler always reached about 100,000 nodes in about 20 minutes, and, in addition, as it approached this number of nodes it really stopped. This led us to believe that we reached most of the Gnutella ultrapeers.

We compared the average distance in the collected Gnutella ultrapeer graph with that of the graph obtained from it by 100 jumps per a node. The parameter d in the jump algorithm was set equal to the average degree in the Gnutella ultrapeer graph, which was 20. The comparison of the histograms of degrees and distances before and after the jumps are presented in Figure 5. It is clearly visible that the histogram of the degrees shrunk after the jumps, which is a

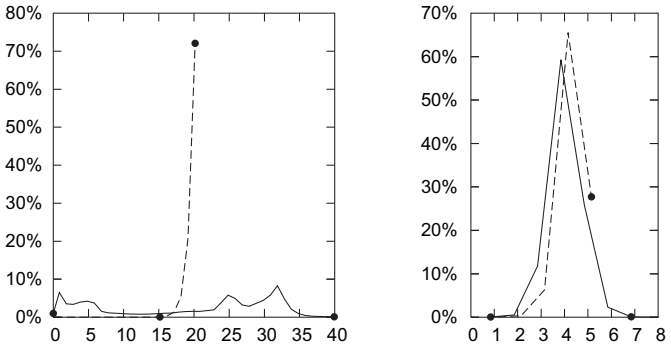


Figure 5. Results of jumps performed on a snapshot of 100,000 Gnutella ultrapeers (solid lines: original network; dashed lines: after jumps): (a) degrees and (b) distances.

consequence of the nature of the jumps algorithm. While the average degree stayed the same, the average distance slighly increased (from 4.1794 to 4.207).

During the initial analysis of the Gnutella ultrapeers network, we noticed 20 nodes that had degree higher than 100 (there were even two nodes with degree over 2000). In order to rule out the possibility that these nodes might have been responsible for the low average distance, we removed them before the experiment.

Although the execution of jumps on the Gnutella ultrapeer network may seem useless, it shows two important results. First, according to the results, the Gnutella ultrapeer network appears to be already random enough; at least, it attains an average distance similar to that of a random graph with the same parameters. Second, it shows that one may obtain a topology similar to the current Gnutella ultrapeer network just by using jumps, *without using the preliminary cache of nodes*. This becomes even more important when one realizes that the very first boostrapping procedure of a Gnutella node is completely based only on advertised web servers containing known lists of running nodes. According to the results presented in the previous paragraph, we believe that the jumps would be able to build a Gnutella-like topology while avoiding the current bootstrap procedure and replacing it by the expanding ring multicast.

4.6. Synchronizing Jumps with Other Operations

The original version of the jump algorithm described in Section 2 may not be suitable for all applications. For instance, in the case of numcast or the search operation described in Sections 4.4 and 4.3, respectively, it is necessary to collect the appropriate results back along the same paths. If a node, starting the search

primitive, performed a jump immediately after the search, he would not be able to receive any response. Moreover, even a node serving at some time as a broker for others may not simply drop its connections via ajump.

Therefore, it is necessary for these cases to implement a mechanism into the jump algorithm that will prevent such collisions. At best, the mechanism should be application independent (it should work for numcast, search, and possibly for new and yet-nonexistent operations) and transparent (so that the applications should not be responsible for providing extra logic for synchronization with jumps).

We propose the following mechanism. The main issue of the synchronization is that the jump may simply drop a connection that is currently indispensable or important for some reason. Usually such connection is needed because an incoming message is awaited along it. Hence, each application should be responsible for flagging chosen connections as "important" for some time interval. If a connection is flagged as important, it cannot be removed during the jumps. Since it is not permissible to lock connections indefinitely, after the given time interval, the connection looses the flag and can be removed during the jumps. Moreover, the application should also be able to remove the flag as soon as it knows that the connection is not needed. For that, it is necessary to keep a list of bookings for each connection instead of just by a single flag. Each booking should contain two parameters: the unique identifier (returned to the application at the time of booking) and the expiration time.

Marking the edges may seem to be an extra burden for the applications, but we believe that in most cases it could be built flawlessly into the application logic so that the users do not have to worry about it. Let us discuss, for instance, the search primitive. If a node is about to forward a search request, it knows the TTL of the request. Based on TTL and the average degree in the network, the node can estimate (or slightly overestimate) the time needed to forward the request and receive responses. It should then flag all of the edges that it uses to forward the search request and set the expiration time to that computed value. If the given time interval of some edge elapses before all the responses are collected and the node decides to jump, all remaining undelivered responses are lost. In the case of search, it does not constitute a big loss, because the desired information could have been localized on many other nodes.

We conclude this section with a formal description of the proposed modification. In fact, the only change in the jump algorithm description presented in Section 2 is related to the first phase. The choice of the node $first(v')$ in step 1(a) should be restricted to connections that are not flagged, and in step 1(b) the spanning T should be made only from non-flagged edges.

5. Conclusion and Future Work

Network formation and maintenance is very important in P2P applications. Networks that possess high connectivity, low degree, and small diameter are much more efficient and robust. In this paper, we offer a low-cost scheme that attains and maintains these properties in P2P network, with little external support. Our approach, based on the jump algorithm, rapidly confers these attributes on arbitrary graphs, leading to significant improvements in the efficiency of common P2P network operations such as searching and broadcasting. We note that this algorithm may even be useful in networks like Gnutella, by obviating the need for preliminary node lists.

The jump algorithm may be enhanced in some ways. One current limitation is that the continual topology changes caused by jumps might disrupt other ongoing operations. One should explore the idea of temporarily locking connections to preclude such disruptions. The applicability of the jump algorithm on highly dynamic networks, with a high rate of joins and leaves, should be investigated in detail.

In a more theoretical direction, we mention the problem of investigating the mixing rate of the actual jump algorithm: we have argued that the network topology achieved by jumps shares essential characteristics with classical random graphs on the basis of a simplified model and on simulation results. Analyzing the actual jump dynamics theoretically seems to us to be quite a challenging task.

Acknowledgments. The second author was partially supported by FAPESP and CNPq through a Temático-ProNEx project (Proc. FAPESP 2003/09925-5) and supported by CNPq (Proc. 308509/2007-2, 485671/2007-7, and 486124/2007-0). Part of this work was done while this author was visiting Emory University, on leave from the University of São Paulo. The third author was supported by NSF grant DMS 0300529.

References

[Aldous and Fill 02] D. Aldous and J. Fill. *Reversible Markov Chains and Random Walks on Graphs*. Manuscript, 2002. Available at http://www.stat.berkeley.edu/~aldous/RWG/book.html.

[Bawa et al. 03] M. Bawa, G. S. Manku, and P. Raghavan. "Symphony: Distributed Hashing in a Small World." Paper presented at the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, WA, March 26-28, 2003.

[Bollobás 01] Béla Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics 73, second edition. Cambridge, UK: Cambridge University Press, 2001.

[Chung 97] Fan R. K. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics 92. Washington, DC: American Mathematical Society, 1997.

[Diaconis and Saloff-Coste 96] P. Diaconis and L. Saloff-Coste. "Logarithmic Sobolev Inequalities for Finite Markov Chains." *Ann. Appl. Probab.* 6:3 (1996), 695–750.

[Diaconis and Shahshahani 87] Persi Diaconis and Mehrdad Shahshahani. "Time to Reach Stationarity in the Bernoulli-Laplace Diffusion Model." *SIAM J. Math. Anal.* 18:1 (1987), 208–218.

[Draves et al. 04] Richard Draves, Jitendra Padhye, and Brian Zill. "Routing in Multi-radio, Multi-hop Wireless Mesh Networks." In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pp. 144–128. New York: ACM Press, 2004.

[Feige 95] Uriel Feige. "A Tight Upper Bound on the Cover Time for Random Walks on Graphs." *Random Structures Algorithms* 6:1 (1995), 51–54.

[Freedman and Mazieres 03] M. J. Freedman and D. Mazieres. "Sloppy Hashing and Self-Organizing Clusters." In *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21–22, 2003, Revised Papers*, Lecture Notes in Computer Science 2735, pp. 45–55. Berlin: Springer, 2003.

[Ganesan et al. 03] P. Ganesan, Q. Sun, and H. Garcia-Molina. "YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology." In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 2, pp. 1250–1260. Los Alamitos, CA: IEEE Press, 2003.

[Gnutella 09] "Gnutella2 Developer Network." Available at http://g2.trillinux.org, 2009.

[Harvey et al. 03] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. "Skipnet: A Scalable Overlay Network with Practical Locality Properties." Paper presented at the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, WA, March 26-28, 2003.

[Holfelder et al. 07] Wieland Holfelder, Paolo Santi, Yih-Chun Hu, and Jean-Pierre Hubaux (editors). *Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks, VANET 2007*. New York: ACM Press, 2007.

[Janson et al. 00] Svante Janson, Tomasz Łuczak, and Andrzej Ruciński. *Random Graphs*. New York: Wiley-Interscience, 2000.

[Karbhari et al. 03] P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. Riley, and E. Zegura. "Bootstrapping in Gnutella: A Preliminary Measurement Study." Technical report, Georgia Institute of Technology, 2003.

[Kirk 03] Patrick Kirk. "Gnutella Protocol Development." Available at http://rfc-gnutella.sourceforge.net, 2003.

[Lovász 96] L. Lovász. "Random Walks on Graphs: A Survey." In *Combinatorics, Paul Erdős is Eighty*, Vol. 2, Bolyai Society Mathematical Studies, pp. 353–397. Budapest: János Bolyai Mathematical Society, 1996.

[Maymounkov and Mazieres 02] P. Maymounkov and D. Mazieres. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric." In *Peer-to-Peer Systems:*

First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7–8, 2002, Revised Papers, Lecture Notes in Computer Science 2429, pp. 53–65. Berlin: Springer-Verlag, 2002.

[Pandurangan et al. 01] G. Pandurangan, P. Raghavan, and E. Upfal. "Building Low-Diameter P2P Networks." In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pp. 492–499. Los Alamitos, CA: IEEE Press, 2001.

[Rowstron and Druschel 01] A. I. T. Rowstron and P. Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems." In *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany, November 12–16, 2001, Proceedings*, Lecture Notes in Computer Science 2218, pp. 329–350. Berlin: Springer-Verlag, 2001.

[Sharman 06] Sharman Networks. KaZaA homepage. Available at http://www.kazaa.com, 2006.

[Slyck 08] Slyck homepage (internet magazine). Available at http://www.slyck.com, 2008.

[Stoica et al. 01] 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 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 149–160. New York: ACM Press, 2001.

[TerraNet 08] "TerraNet AB." *Wikipedia*. Available at http://en.wikipedia.org/wiki/TerraNet_AB, 2008.

[Zhao et al. 01] B.Y. Zhao, J.D. Kubiatowicz, and A.D. Joseph. "Tapestry: An Infrastructure for Fault-Tolerant Wide-area Location and Routing." Technical report, UC Berkeley, 2001.

J. Zich, Department of Mathematics and Computer Science, Emory University, 400 Dowman Dr., W401, Atlanta, GA 30322 (jan@ajz.cz)

Y. Kohayakawa, Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, São Paulo, SP 05508-090, Brazil (yoshi@ime.usp.br)

V. Rödl, Department of Mathematics and Computer Science, Emory University, 400 Dowman Dr., W401, Atlanta, GA 30322 (rodl@mathcs.emory.edu)

V. Sunderam, Department of Mathematics and Computer Science, Emory University, 400 Dowman Dr., W401, Atlanta, GA 30322 (vss@emory.edu)

Received March 29, 2006; accepted June 25, 2008.