Percolation Search in Power Law Networks: Making Unstructured Peer-To-Peer Networks Scalable

Nima Sarshar
Department of Electrical Engineering, UCLA
nima@ee.ucla.edu

P. Oscar Boykin
Department of Electrical Engineering, UCLA
boykin@ee.ucla.edu

Vwani P. Roychowdhury
Department of Electrical Engineering, UCLA
vwani@ee.ucla.edu

Abstract

We introduce a scalable searching protocol for locating and retrieving content in random networks with Power-Law (PL) and heavy-tailed degree distributions. The proposed algorithm is capable of finding any content *in the network with* probability one *in time* $O(\log N)$*, with a total traffic that provably scales sub-linearly with the network size,* N*. Unlike other proposed solutions, there is no need to assume that the network has multiple copies of contents; the protocol finds all contents reliably, even if every node in the network starts with a unique content. The scaling behavior of the size of the giant connected component of a random graph with heavy tailed degree distributions under bond percolation is at the heart of our results. The percolation search algorithm can be directly applied to make unstructured Peer-to-Peer (P2P) networks, such as Gnutella, Limewire and other file-sharing systems (which naturally display heavy-tailed degree distributions and scale-free network structures), scalable. For example, simulations of the protocol on the limewire crawl number 5 network[12], consisting of over 65,000 links and 10,000 nodes, shows that even for this snapshot network, the traffic can be reduced by a factor of at least 100, and yet achieve a hit-rate greater than 90%.*

1 Introduction and Motivation

Peer-to-peer (P2P) networking systems consist of a large number of nodes or computers that operate in a decentralized manner to provide reliable global services, such as query resolutions (i.e., database searches), ad hoc point-to-point communications, and cluster or P2P computing. The existing P2P schemes can be broadly categorized into two types: (1) *Unstructured P2P Networks:* Such networks include the popular music and video download services, such as Gnutella[8], Limewire[13], Kazaa[1], Morpheus[2], and Imesh [3]. They together account for millions of users dynamically connected in an ad hoc fashion, and creating a giant federated data base. The salient feature of such networks is that *the data objects do not have global unique ids*, and *queries are done via a set of key words*. (2) *Structured P2P Networks*: These include systems under development, including Tapestry [19], Chord[18], PAST [14, 7], and Viceroy[10], and are *characterized by the fact that each content/item has a unique identification tag or* key; e.g., an m-bit hash of the content is a common choice, leading to the popular characterization of such networks as DHT (Distributed Hash Table) P2P systems.

As opposed to the unstructured networks, which are already being used by millions of users, most of the structured systems are in various stages of development, and it is not clear at all which system is best suited to provide a reliable, load-balanced, and fault-tolerant network. Moreover, unstructured searches using key-words constitute a dominant mechanism for locating content and resources, and for merging/mining already existing heterogeneous sets of data-bases. Thus, unstructured P2P networking will continue to remain an important application domain.

In spite of the great popularity of the unstructured P2P networks, systematic designs of provably robust and scalable networks have not been proposed, and most of the networks currently being used are still ad hoc (even though ingenious) in their designs. The three major problems in designing unstructured P2P systems are: (i) *Lack of systematic protocols for generating global networks with predictable topological properties*: A number of recent studies [15, 12] have shown that the *structure of the existing networks has complex network characteristics, including approximate power law degree distributions*[1]*, small diameter, tolerance to node deletions* etc. However, client-based protocols that guarantee the global emergence of scale-free networks with tunable properties have not been implemented.

[1] A distribution is said to be a power law (PL) distribution, if $P(k) \sim k^{-\gamma}$, where $\gamma > 0$ is called the exponent of the distribution.

Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)
0-7695-2156-8/04 $20.00 © 2004 IEEE

(ii) *Traffic scalability problems:* In a straightforward approach to query resolution, in order to find an object, all the nodes in the network need to be addressed, leading to $O(N)$ total queries in the network for every single query. This results in significant scaling problems and Ripeanu et.al.[12] estimated that in December of 2000 Gnutella traffic accounted for 1.7% of Internet backbone traffic. As reviewed later, a number of ad hoc measures, ranging from forcing an ultra-peer structure on the network to random walk protocols for searching content, have been proposed. But none of these measures provides a true solution to the underlying scalability problem. (iii) *Vulnerability to targeted Attacks:* It is well known that one can crawl such networks and identify the high-degree nodes quite quickly, and thus can potentially disrupt the network, by attacking these high degree nodes. Protocols for identifying or compensating for such attacks, or even recovering efficiently after such an attack has disrupted the network are yet to be designed.

In this paper *we provide a systematic solution to the scalability problem for unstructured P2P networks*. We first show how to perform scalable parallel search in random Power-law networks with exponents between 2 and 3 (and other heavy-tailed degree distribution networks, where the variance is much larger than the average degree), *when each node starts with a unique content, and queries are made randomly for any of these contents from any of the nodes* (Section III). The key steps in our search algorithm are, (i) an initial one-time-only replication of a node's content list or directory in the nodes visited via a short random walk, and (ii) a probabilistic broadcast scheme for propagating queries, which in graph theoretic terms is an implementation of bond percolation on the underlying networks. We would like to note that, while the design of the protocol is based on theoretical concepts, **the final protocol is straightforward and is very easy to implement**. For example, for a PL network with exponent, $\tau = 2$, and maximum degree k_{max}, we show that any content in the network can be found with probability one in time $O(\log N)$, while generating only $O(N \times \frac{2 \log k_{max}}{k_{max}})$ traffic per query. Thus if $k_{max} = cN$ (as is the case for a random PL network) then the overall traffic scales as $O(\log^2 N)$ per query, and if $k_{max} = \sqrt{N}$ (as is the case for most grown graphs) then the overall traffic scales as $O(\sqrt{N} \log^2 N)$ per query. *The reason we consider PL networks is because unstructured P2P networks, both existing and proposed ones [17, 12], are characterized by random PL and heavy-tailed degree distributions*. We provide both simulation and analytical studies of the improvements to be accrued from the percolation search algorithms when implemented on Gnutella crawl networks (Section IV).

2 The Traffic Scaling Problem and Prior Work

The scaling problem associated with key-word based searches in P2P networks can be described in terms of Gnutella, which is a real-world network that employs a broadcast search mechanism to allow searching for computer files. Various additions have been made[8] to the original protocol; however, the model is essentially the following: each query has a unique identifier and a time-to-live (TTL). As a node receives a query it checks the identifier to verify that it has not already processed this query. If the node has not previously processed the query, it checks its local files and responds to the query if it finds a match. Finally, if the TTL is greater than 0, it decrements the TTL and passes the query to all nodes it is connected to (except the node from which it received the query). The unique identifier prevents loops in query routing. If the TTL is greater than the diameter of the network, each query passes each link exactly once, and all nodes receive the query. This means that each node would send or receive each query a number of times equal to the average degree of the network, $\langle k \rangle$, *which means that total communication cost per query is* $\langle k \rangle N$. Thus, every node[2] must process all queries. This problem manifests itself in two important ways. First, low capacity nodes are very quickly overloaded and fragment the network[3]. Second, total traffic per node increases at least linearly with the size of the network. To give an idea of the magnitude of communication required, Ripeanu et.al.[12] estimated that in December of 2000 Gnutella traffic accounted for 1.7% of Internet backbone traffic.

Following is an account of a few important attempts at mitigating the traffic scaling problem:

1. Ultra-peer Structures and Cluster-Based Designs: A non-uniform architecture with an explicit hierarchy seems to be the quickest fix. This was motivated by the fact that the nodes in the network are not homogeneous; a very large fraction of the nodes have small capacity (e.g. dial-up modems) and a small fraction with virtually infinite capacity. The idea is to assign a large number of low capacity nodes to one or more Ultra-peers. The Ultra-peer knows the contents of its leaf nodes and sends them the relevant queries only. Among the Ultra-peers they perform the usual broadcast search.

The Ultra-peer solution helps shield low bandwidth users; however, the design is non-uniform, and an explicit hierarchy is imposed on nodes. In fact, the two-level hierarchy is not scalable in the strict sense. After more growth of the network, the same problem will start to appear among

[2] This has been mitigated somewhat with the introduction of Ultra-peers as we discuss later in the section.

[3] this is believed to have happened to the original Gnutella in August 2000.

the Ultra-peers, and the protocol should be augmented to accommodate a third level in the hierarchy, and so on. In a more strict theoretical sense, the traffic still scales linearly, but is always a constant factor (determined by the average number of nodes per ultra-peer) less than the original Gnutella system.

The *results of this paper might be considered as an alternative to artificially imposing a hierarchical structure*: in our percolation search algorithm, each search automatically distills an Ultra-peer-like subnetwork, and no external hierarchy needs to be imposed.

2. Random Walk Searches with Content Replication: Lv et.al.[9] analyze random walk searches with file replication. The random walk search idea is simple: for each query, a random walker starts from the initiator and asks the nodes on the way for the file until it finds a match. If there are enough replicas of every file on the network, each query would be successfully answered after a few steps. In [9] it is assumed that a fraction λ_i of all nodes have the file i. They consider the case where λ_i might depend on the probability (q_i) of requesting content i. They show that under their assumptions, performance is optimal when $\lambda_i \propto \sqrt{q_i}$.

This scheme has several disadvantages. Since high connectivity nodes have more incoming edges, random walks gravitate towards high connectivity nodes. *A rare item on a low connectivity node will almost never be found.* To mitigate these problems, [9] suggests avoiding high degree nodes in the topology. Moreover, this scheme is not scalable in a strict sense either: even with the uniform caching assumption satisfied, the design requires $O(N)$ replications per content, and thus, assuming that each node has a unique content, it will require a total of $O(N^2)$ replications and an average $O(N)$ cache size. The above scaling differs only by a constant factor from the straightforward scheme of all nodes caching all files.

3. Random Walk on Power-Law Graphs: In contrast to the above approach, Adamic et. al.[4] proposed an algorithm that takes advantage of the existence of high degree nodes: A random walker starts from a node to resolve a query. At each step of the walk, it scans the neighbors of the node it visits for hits. For a power-law graph, the random walk quickly converges towards high-degree nodes. These nodes are expected to have many neighbors; hence, they can answer many queries. *This work is among the few that attempt to derive the scaling behavior of the search time with the size of the network N analytically*. Their scheme, however, suffers from a serious issue: the walk very quickly finds a finite fraction of all queries (e.g. 50% of the network), but their simulations show it takes time $O(N^{.79})$ to query all nodes, much longer than their analytical prediction of $O(N^{.15})$.

Scanning neighbors is equivalent to getting each node to cache its directory (or content list) on each of its neighbors. Thus, an advantageous characteristic of their work is having an average cache size of $O(\log N)$, in contrast to $O(N)$ for the previous scheme. This is because for a power-law network with exponent -2, the average degree of a node is $O(\log N)$. One should also note that the traffic on a highly connected node is much more than low connectivity ones, because almost all walks pass through them. *For the very same reason, the contents of low connectivity nodes are very hard to find*.

As noted in the introduction, in this paper we show that *as long as the network topology is random and has an appropriately heavy-tailed degree distribution* (e.g., PL networks with exponent between 2 and 3), then one can design a search protocol with the following characteristics: (i) *Each node can start with a unique content* and no assumption on the relative abundance of any content is necessary for the search algorithm to succeed. (ii) A parallel probabilistic broadcast search can be performed (unlike the sequential random-walk based search protocols discussed above) that *finds any content*, whether residing in a low degree or a high-degree node, with probability one in time that is logarithmic in the size of the network. (iii) The algorithm is *truly decentralized and is carried out only via local decisions*, and (iv) No explicit hierarchy or central control needs to be imposed during the operation of the network.

3 The Percolation Search Algorithm and Its Scaling Properties

The percolation search algorithm can be described as follows:

(i) *Content List Implantation:* Each node in a network of size N duplicates its content list (or directory) through a random walk of size $L(N,\tau)$ starting from itself. The exact form of $L(N,\tau)$ depends on the topology of the network (i.e., τ for PL networks), and is in general a sub-linear function of N. Thus the total amount of directory storage space required in the network is $NL(N,\tau)$, and the average cache size is $L(N,\tau)$. Note that, borrowing a terminology from the Gnutella protocol, *the length of these implantation random walks will be also referred to as the TTL (Time To Live)*.

(ii) *Query Implantation:* To start a query, a query request is *implanted* through a random walk of size $L(N,\tau)$ starting from the requester.

(iii) *Bond Percolation:* When the search begins, each node with a query implantation starts a *probabilistic broadcast search*, where it sends a query to each of its neighbors with probability q, with $q = q_c/\gamma$ where q_c is the percolation threshold [16].

We next derive scaling and performance measures of the above algorithm. Our derivations will follow the following steps:

- First we define high degree nodes and compute the number of high degree nodes in a given network.
- Second, we show that after the probabilistic broadcast step (i.e., after performing a bond percolation in the query routing step), a query is received by all members of connected component to which an implant of that query belongs. We also see that the diameter of all connected components is $O(\log N)$, and thus the query propagates through it quickly.
- Third, we show that *a random walk of length* $L(N, \tau)$ starting from any node will *pass through a highly connected node, with probability approaching one*. This will ensure that (i) a pointer to any content is owned by at least one highly connected node, and (ii) at least one implant of any query is at one of the high degree nodes.
- Finally, we examine the scaling of the maximum degree of the network k_{max} and give the scaling of query costs and cache sizes in terms of the size of the entire network N. We show that both cache size and query cost scale sublinearly for all $2 \leq \tau < 3$, and indeed can be made to scale $O(\log^2 N)$ with the proper choice of τ and k_{max}.

3.1 High Degree Nodes

In this section we define the notion of a high degree node. For any node with degree k, we say it is a high degree node if $k \geq k_{max}/2$. We assume that we deal with random power-law graphs which have a degree distribution:

$$\begin{aligned} p_k &= Ak^{-\tau}, \\ \text{where } A^{-1} &= \sum_{k=2}^{k_{max}} k^{-\tau} \approx \zeta(\tau) - 1, \end{aligned}$$

and $\zeta(\cdot)$ is the Riemann zeta function. A approaches the approximate value quickly as k_{max} gets large, and thus can be considered constant. Thus the number of high degree nodes, H is given by:

$$H = N\left(A\sum_{k=k_{max}/2}^{k_{max}} k^{-\tau}\right).$$

Since for all decreasing, positive, $f(k)$ we have $\sum_{k=a}^{b} f(k) > \int_a^{b+1} f(k)dk > \int_a^b f(k)dk$ and $\sum_{k=a}^{b} f(k) < \int_{a-1}^{b} f(k)dk$, we can bound H from above and below:

$$H > \frac{A}{\tau-1}\left(\frac{1}{(\frac{1}{2})^{\tau-1}} - 1\right)\frac{N}{k_{max}^{\tau-1}}, \text{ and}$$

$$H < \frac{A}{\tau-1}\left(\frac{1}{(\frac{1}{2})^{\tau-1}(1-\frac{1}{k_{max}/2})} - 1\right)\frac{N}{k_{max}^{\tau-1}}.$$

For $k_{max} \to \infty$ we have that $\frac{1}{k_{max}/2} \to 0$ thus:

$$H \approx \frac{A}{\tau-1}\left(2^{\tau-1} - 1\right)\frac{N}{k_{max}^{\tau-1}}.$$

We have shown that $H = O(\frac{N}{k_{max}^{\tau-1}})$. As we discuss in section 3.5, there are two choices for scaling of k_{max}. If we put no prior limit on k_{max} it will scale like $O(N^{1/(\tau-1)})$. As we will discuss, we may also consider $k_{max} = O(N^{1/\tau})$. We should note that the first scaling law gives $H = O(1)$, or a constant number of high degree nodes as the system scales. The second gives $H = O(N^{1/\tau})$. For all $\tau \geq 2$, we have H scaling sublinearly in N.

In the next sections we will show that without explicitly identifying or arranging the high degree nodes in the network, we can still access them and make use of their resources to make the network efficiently searchable.

3.2 High Degree Nodes are in the Giant Component

In conventional percolation studies, one is guaranteed that as long as $q - q_c = \epsilon > 0$, where ϵ is a constant independent of the size of the network, then there will be a giant connected component in the percolated graph. However, in our case, i.e., PL networks with $2 \leq \tau \leq 3$, $\lim_{N\to\infty} q_c = 0$ (for example, $q_c = \frac{\log(k_{max})}{k_{max}}$ for a PL network with exponent $\tau = 2$ [6]), and since the traffic (i.e., the number of edges traversed) scales as $O(\langle k\rangle Nq)$, we cannot afford to have a constant $\epsilon > 0$ such that $q = \epsilon + q_c$: the traffic will then scale linearly.

Hence, we will percolate not at a constant above the threshold, but at a multiple above the threshold: $q = q_c/\gamma$. We consider this problem in detail in a separate work[16]. The result is that if we *follow a random edge in the graph, the probability it reaches an infinite component is* $\delta = z/k_{max}$ *for a constant* z *which depends only on* τ *and* γ*, but not* k_{max}.

Thus, since each high degree node has at least $k_{max}/2$ degree, the average number of edges of a high degree node that connect to the infinite component (k_{inf}) is at least:

$$k_{inf} \geq \delta\frac{k_{max}}{2} = \frac{z}{k_{max}}k_{max}/2 = \frac{z}{2}. \qquad (1)$$

The probability that a high degree node has at least one link to the infinite component is at least:

$$\begin{aligned} P &\geq 1 - (1-\delta)^{k_{max}/2} \\ &= 1 - (1 - \frac{z}{k_{max}})^{k_{max}/2} \\ &\geq 1 - e^{-z/2}. \end{aligned}$$

Thus both the average number of degrees that a high degree node has to the giant component, and the probability that a

high degree node has at least one edge to the giant component are independent of k_{max}. So as we scale up k_{max}, we can expect that the high degree nodes stay connected to the giant component. We can make z larger by decreasing γ, particularly, if $1/\gamma > 2/(3-\tau)$ we have $z > 1$ [16].

It remains to be shown that the diameter of the connected component is on the order of $O(\log N)$. To see this, we use the approximate formula $l \approx \frac{\log M}{\log d}$ [11] of the diameter of a random graph with size M and average degree d. We know that the size of the percolated graph is $\frac{Nz}{k_{max}}\langle k\rangle$ and that the average degree is approximately 2[16]. Thus the diameter of the giant component is:

$$\begin{aligned} l &= \frac{\log(\frac{Nz}{k_{max}}\langle k\rangle)}{\log(2)} \\ &= \frac{\log\frac{N}{k_{max}} + \log z + \log\langle k\rangle}{\log(2)} = O(\log N). \end{aligned}$$

At this point we have presented the main result. If we can cache content on high degree nodes, and query by percolation *starting from* a high degree node, we will always find the content we are looking for. We have not yet addressed how each node can find a high degree node. In the next section we show that by taking a short random walk through the network we will reach a high degree node with high probability, and this gives us the final piece we need to make the network searchable by all nodes.

3.3 Random Walks Reach High Degree Nodes

Consider a random PL network of size N and with maximum node degree k_{max}. We want to compute the probability that following a randomly chosen link one arrives at a high degree node. To find this probability, consider the generating function $G_1(x)$[16] of the degree of the nodes arrived at by following a random link:

$$G_1(x) = \frac{\sum_{k=2}^{k_{max}} k^{-\tau+1}x^{k-1}}{C}, \tag{2}$$

where $C = \sum_{k=2}^{k_{max}} k^{-\tau+1}$. This results in the probability of arriving at a node with degree greater than $\frac{k_{max}}{2}$ to be:

$$P_\tau = \frac{\sum_{k=k_{max}/2}^{k_{max}} k^{-\tau+1}}{C}. \tag{3}$$

Since the degrees of the nodes in the network are independent, each step of the random walk is an independent sample of the same trial. The probability of reaching a high degree node within $\frac{\alpha}{P_\tau}$ steps is:

$$1-(1-P_\tau)^{\alpha/P_\tau} \quad \geq \quad 1-e^{-\alpha}.$$

Therefore, after $O(1/P_\tau)$ steps, a high degree node will be encountered in the random walk path with high (constant) probability. Now we need to compute P_τ for $\tau = 2$ and $2 < \tau < 3$. Since for all decreasing, positive, $f(k)$ we have $\sum_{k=a}^{b} f(k) > \int_a^{b+1} f(k)dk > \int_a^b f(k)dk$ and $\sum_{k=a}^{b} f(k) < \int_{a-1}^{b} f(k)dk$, we can bound the following sums.

If $\tau = 2$, we have the probability of arriving at a node with degree greater than $\frac{k_{max}}{2}$ is:

$$\begin{aligned} P_2 &= \frac{\sum_{k=k_{max}/2}^{k_{max}} k^{-1}}{C} \\ &> \frac{\log(k_{max}) - \log(k_{max}/2)}{C} = \frac{-\log 2}{C}, \end{aligned}$$

and $C = \sum_{k=2}^{k_{max}} k^{-1} < \log(k_{max})$. We finally get:

$$P_2 > \frac{-\log 2}{\log(k_{max})}. \tag{4}$$

For $\tau = 2$, then in $O(1/P_2) = O(\log k_{max})$ steps we have reached a high degree node.

If $2 < \tau < 3$, we have the probability of arriving at a node with degree greater than $\frac{k_{max}}{2}$ is:

$$\begin{aligned} P_\tau &= \frac{\sum_{k=k_{max}/2}^{k_{max}} k^{-\tau+1}}{C} \\ &> \frac{1}{\tau-2}(2^{\tau-2}-1)\frac{1}{Ck_{max}^{\tau-2}}, \end{aligned}$$

and $C = \sum_{k=2}^{k_{max}} k^{-\tau+1} < \frac{1}{\tau-2}(1-\frac{1}{k^{\tau-2}})$. We finally get:

$$P_\tau > \frac{2^{\tau-2}-1}{k_{max}^{\tau-2}-1}. \tag{5}$$

For $2 < \tau < 3$, then in $O(1/P_\tau) = O(k_{max}^{\tau-2})$ steps we have reached a high degree node, which is polynomially large in k_{max} rather than logarithmically large, as in the case of $\tau = 2$.

A sequential random walk requires $O(k_{max}^{\tau-2})$ time steps to traverse $O(k_{max}^{\tau-2})$ edges, and hence, the query implantation time will dominate the search time, making the whole search time scale faster than $O(\log N)$. Recall that the percolation search step will only require $O(\log N)$ time, irrespective of the value of τ. A simple parallel query implantation process can solve the problem. To implement $k_{max}^{\tau-2}$ query seeds for example, a random walker with time to live (TTL) of $K = \log_2 k_{max}^{\tau-2}$ will initiate a walk from the node in question and at each step of the walk it implants a query seed, and also initiates a second random walker with time to live $K-1$. This process will continue recursively until the time to live of all walkers are exhausted. The number of

links traversed by all the walkers is easily seen to be:

$$\sum_{i=0}^{K-1} 2^i = 2^K - 1 = k_{max}^{\tau-2} - 1.$$

In practice, for values of τ close to two, the quality of search is fairly insensitive to how the number of query implants are scaled.

3.4 Communication Cost or Traffic Scaling

Each time we want to cache a content, we send it on a random walk across $L(N,\tau) = O(1/P_\tau)$ edges. When we make a query, if we reach the giant component, each edge passes it with probability q (if we don't reach a giant component only a constant number of edges pass the query). Thus, the total communications traffic scales as $qE = q_c\langle k\rangle N/\gamma$. Since $q_c = \langle k\rangle/\langle k^2\rangle$ we have $\mathcal{C}_\tau = O(\frac{\langle k\rangle^2 N}{\langle k^2\rangle})$. For all $2 \le \tau < 3$, $\langle k^2\rangle = O(k_{max}^{3-\tau})$. For $\tau = 2$, $\langle k\rangle = \log k_{max}$ which gives

$$\mathcal{C}_2 = O\left(\frac{\log^2 k_{max} N}{k_{max}}\right) \quad (6)$$

For $2 < \tau < 3$, $\langle k\rangle$ is constant which gives

$$\mathcal{C}_\tau = O\left(k_{max}^{\tau-3} N\right) \quad (7)$$

In section 3.1, we showed that the number of high degree nodes $H = O(N/k_{max}^{\tau-1})$. We also know that $L(N,\tau) = \alpha/P_\tau$ and $P_2 = O(1/\log k_{max})$ and $P_\tau = O(1/k_{max}^{\tau-2})$. Thus we can rewrite the communication scaling in terms of the high degree nodes, $\mathcal{C}_\tau = O(L(N,\tau)^2 H)$. So we see that communication costs scales linearly in H, but as the square of the length of the walk to the high degree nodes. This meets with our intuition since the high degree nodes are the nodes that store the cache and answer the queries.

In the next section we discuss explicit scaling of k_{max} to get communication cost scaling as a function of N. Table 1 shows the scaling of the cache and communication cost in N. We see that for all $\tau < 3$, we have sublinear communication cost scaling in N.

3.5 On Maximum Degree k_{max}

There are two ways to generate a random PL network:
(i) Fix a k_{max} and normalize the distribution, i.e.,

$$p_k = Ak^{-\tau}, 0 < k \le k_{max}\,, \quad (8)$$

$$\text{where } A^{-1} = \sum_{k=1}^{k_{max}} k^{-\tau}\,. \quad (9)$$

To construct the random PL graphs, N samples are then drawn from this distribution. For several reasons, the choice $k_{max} = O(N^{1/\tau})$ is recommended in the literature [5], and in our scaling calculations (e.g., Table 1) we follow this upper bound.
(ii) No a priori bound on the maximum is placed, and N samples are drawn from the distribution $p_k = Ak^{-\tau}$, where $A^{-1} = \sum_{k=1}^{\infty} k^{-\tau}$. It is quite straightforward to show that almost surely, $k_{max} = O(N^{\frac{1}{\tau-1}})$. Thus, when $\tau = 2$, $k_{max} = cN$ $(1 > c > 0)$ in this method of generating a random PL graphs.

A potential problem with using the larger values of k_{max}, as given by method (ii), is that the assumption that the links are chosen independently might be violated. Random graph assumptions can be shown to still hold when the maximum degree of a power-law random graph is $k_{max} = O(N^{1/\tau})$ [5]. This however does not necessarily mean, that the scaling calculations presented in the previous section do not hold for $k_{max} = O(N^{\frac{1}{\tau-1}})$. In fact, extensive large-scale simulations (see Section 4) suggest that one can indeed get close to poly-logarithmic scaling of traffic (i.e., $O(\log^2 N)$), as predicted by the scaling calculations in this section.

There are several practical reasons for bounding k_{max}, as well. First, in most grown random graphs, k_{max} scales as $N^{1/\tau}$. While grown random graphs display inherent correlations, we would like to compare our scaling predictions with performance of the search algorithm when implemented on grown graphs. Hence, the scaling laws that would be relevant for such P2P systems correspond to the case of bounded k_{max}. Second, since the high degree nodes end up handling the bulk of the query traffic, it might be preferable to keep the maximum degree low. For example, for $\tau = 2$, the traffic generated is of the same order as the maximum degree, when $k_{max} = c\sqrt{N}$, thus providing a balance between the overall traffic and the traffic handled by the high degree nodes individually.

$\mathbf{k_{max} = O(N^{1/(\tau-1)})}$	Cache Size	Query Cost
$\tau = 2$	$O(\log N)$	$O(\log^2 N)$
$2 < \tau < 3$	$O(N^{\frac{\tau-2}{\tau-1}})$	$O(N^{\frac{2\tau-4}{\tau-1}})$
$\mathbf{k_{max} = O(N^{1/\tau})}$	Cache Size	Query Cost
$\tau = 2$	$O(\log N)$	$O(\log^2(N)N^{1/2})$
$2 < \tau < 3$	$O(N^{1-2/\tau})$	$O(N^{2-3/\tau})$

Table 1. The scaling properties of the proposed algorithm when $k_{max} = O(N^{\frac{1}{\tau-1}})$ (top) and $k_{max} = O(N^{1/\tau})$ (bottom).

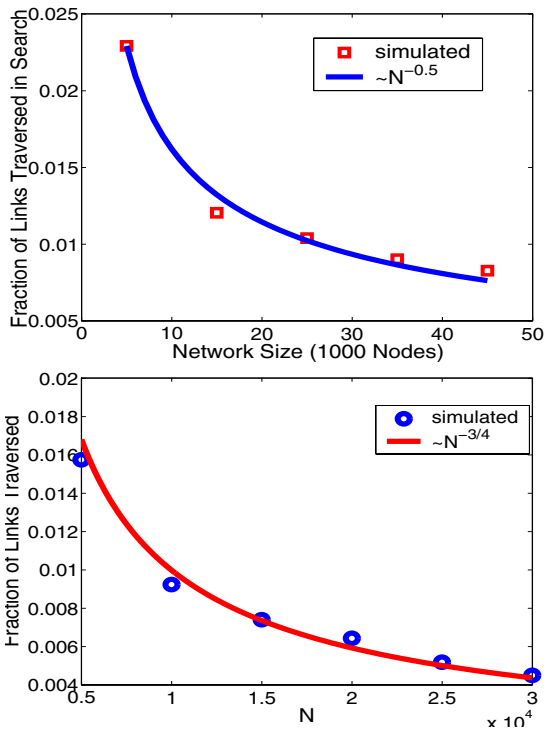

Figure 1. Scaling behavior of the percolation probability required for a fixed hit-rate of 95% as a function of the network size for a network with $\tau = 2$. For the top figure, the maximum degree is scaled as $N^{1/2}$ while it is scaled as $N^{3/4}$ in the bottom figure. The scalings predicted in the paper are also plotted for comparison.

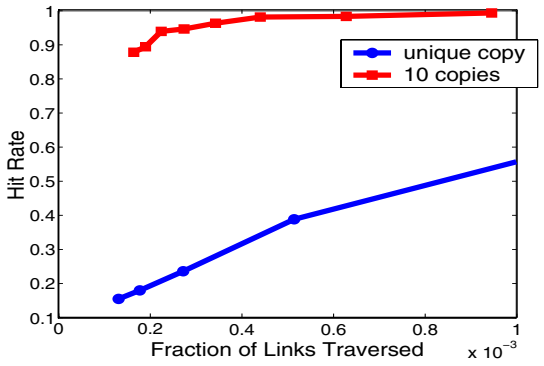

Figure 2. Throughout the paper, we considered a unique replica of any content to exist in the network. The performance of the search algorithm greatly improves in the more realistic case where more than one copy of contents exist in the network. As an example, the case where only 10 replicas of any content is randomly spread in the network is considered in the above figure for a PL network with $\tau = 2$, $N = 30K$ and the average degree 6. Around 90% of all contents are found with only 0.02% of the original traffic.

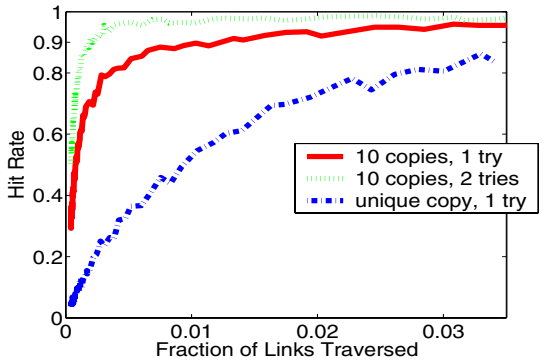

Figure 3. Performance of the Percolation Search Algorithm on A Gnutella Crawl Graph: The hit-rate as a function of the fraction of links used in search, for limewire crawl number 5. The crawl covers around around 65,000 links and 10,000 nodes. The hit-rate for different number of trials are depicted separately. The TTL used for both query and content implant has length 30. It shows that *even for this snapshot network, the traffic is reduced by a factor of at least 100* for a hit-rate greater than 90% in four attempts.

4 Simulations on Random PL Networks and Gnutella Crawl Networks

Figs. 1–5 provide simulation results verifying the performance and scaling predictions made by the analysis presented in the previous section. Note that in the simulations, TTL refers to the length of the random walks performed for content-list replication and query implantation.

5 Concluding Remarks

We have presented a novel scalable search algorithm that uses random-walks and bond percolation on random graphs with heavy-tailed degree distributions to provide access to any content on any node with probability one. ***While the concepts involved in the design of our search algorithm***

have deep theoretical underpinnings, any implementation of it is very straightforward, and can be easily incorporated into any software system and protocol. Our extensive simulation results using both random PL networks and Gnutella crawl networks show that unstructured P2P networks can indeed be made scalable. Moreover, we show that even in networks with different categories of nodes (i.e., graphs where the degree distribution is a mixture of heavy-tailed and light-tailed distributions) the search algorithm exhibits the favorable scaling features, while shielding the nodes with light-tailed degree distribution from the query-generated traffic.

Our ongoing and future work involves the design of systematic protocols that will guarantee the emergence of scale-free network topologies, even when the participating nodes have different bandwidth capacities. The percolation search algorithm, combined with such networking protocols, will then have the potential to lead to the systematic and truly decentralized design of scalable and robust unstructured P2P networks.

References

[1] *How peer-to-peer (p2p) and kazaa media desktop work*, Kazaa Website:http://www.kazaa.com/us/help/guide-aboutp2p.htm, 2003.

[2] *homepage*, http://www.morpheus.com/index.html, 2004.

[3] *homepage*, http://www.imesh.com/, 2004.

[4] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman, *Search in power-law networks*, Phys. Rev. E **64** (2001), 046135.

[5] William Aiello, Fan Chung, and Linyuan Lu, *A random graph model for massive graphs*, Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM Press, 2000, pp. 171–180.

[6] ———, *Random evolution in massive graphs*, IEEE Symposium on Foundations of Computer Science, 2001.

[7] P. Druschel and A. Rowstron, *PAST: A large-scale, persistent peer-to-peer storage utility*, Proceedings of the Eigth IEEE Workshop on Hot Topics in Operating Systems (HotOS) (Schoss Elmau, germany), May 2001.

[8] T. Klingberg and R. Manfredi, *Rfc-gnutella*, http://rfc-gnutella.sourceforge.net.

[9] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker, *Search and replication in unstructured peer-to-peer networks*, Proceedings of the 16th international conference on Supercomputing, ACM Press, 2002, pp. 84–95.

[10] Dahlia Malkhi, Moni Naor, and David Ratajczak, *Viceroy: a scalable and dynamic emulation of the butterfly*, Proceedings of the twenty-first annual symposium on Principles of distributed computing, 2002, pp. 183–192.

[11] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, *Random graphs with arbitrary degree distributions and their applications*, Phys. Rev. E **64** (2001), 026118.

[12] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi, *Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design*, IEEE Internet Computing Journal **6** (2002), no. 1.

[13] Christopher Rohrs, *Limewire design*, lime wire documents:http://www.limewire.org/project/www/design.html, 2001.

[14] A. Rowstron and P. Druschel, *Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility*, Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01) (Chateau Lake Louise, Banff, Canada), October 2001.

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

[16] Nima Sarshar, P. Oscar Boykin, and Vwani P. Roychowdhury, *Percolation at a multiple of the threshold*, 2004.

[17] Nima Sarshar and Vwani Roychowdhury, *Scale-free and stable structures in complex ad hoc networks*, Phys. Rev. E **69** (2004), no. 026101.

[18] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, Proceedings of the ACM SIGCOMM 2001 Technical Conference (San Diego, USA), August 2001.

[19] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*, Tech. Report CSD-01-1141, U. C. Berkeley, Apr 2001.