Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 355 (2006) 48–64

Theoretical Computer Science

www.elsevier.com/locate/tcs

Scalable percolation search on complex networks

Nima Sarshar[a,*], Oscar Boykin[b], Vwani Roychowdhury[a]

[a]Complex Networks Group, Department of Electrical Engineering, University of California, Los Angeles, USA
[b]Department of Electrical and Computer Engineering, University of Florida, Gainesville

Abstract

We introduce a scalable searching protocol for locating and retrieving content in random networks with heavy-tailed and in particular power-law (PL) 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 approximate scale-free network structures), scalable. For example, simulations of the protocol on the limewire crawl number 5 network [Ripeanu et al., Mapping the Gnutella network: properties of large-scale peer-to-peer systems and implications for system design, IEEE Internet Comput. J. 6 (1) (2002)], 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%.

Keywords: Peer-to-peer networks; Unstructured complex networks; Scalable search; Percolation search algorithm

1. Introduction and motivation

The literature on scale-free or power-law (PL) structures in complex networks can be divided into three main classes:

Group (i): The class of empirically discovered PL relations, mostly in the degree distribution of these networks. The work of this group was mostly populated with the discovery of such PL relations for Internet in 1999 [12], quickly followed by similar discoveries for many other complex networks from neural networks to peer-to-peer (P2P) networks.

Group (ii): The class of work that aims at finding dynamical models that can produce networks with PL degree distributions. These models mostly constitute of gradual addition of nodes and some variations of preferential attachment of the new links to the existing nodes. By tuning the parameters of these models, a continuum of PL distributions could emerge (see [7] for a survey of these models). These dynamics were mainly aimed at modelling the emergence of empirically discovered PL relations.

Other authors have examined the emergence of PL structures in networks as the outcome of ongoing performance optimization in evolving networks (e.g., for optimized tolerance in [9]). For example, in the case of communication

* Corresponding author.
E-mail addresses: nima@ee.ucla.edu (N. Sarshar), boykin@ee.ucla.edu (O. Boykin), vwani@ee.ucla.edu (V. Roychowdhury).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.12.014

networks, optimal utilization of the heterogeneity of the resources available to the members of the network, calls for nonuniform connectivity distributions. To optimally utilize the resources of the network, nodes with more resources and capabilities should assume more *central* roles, often by acquiring more connections.

Group (iii): A fairly recent class of work based on the idea that network formation protocols can be *designed* to ensure the existence of PL connectivity structure in the emergent complex networks. This group [21,23], argues that random PL (or other heavy-tailed) networks are in fact desirable topologies for communication networks of various kinds, and such topologies can be actively exploited to provide scalable global services in highly dynamic and ad hoc environments.

The works of Group (iii) was motivated, among other things, by the controversy on the accuracy of dynamical models in accounting for characteristics observed in real networks, or even the sheer existence of some of the those characteristics. For instance, a couple of years after the discovery of PL relations in Internet [12], the authors in [10] reexamined the data on which the results in [12] were based and found that these data can only provide an incomplete view of the Internet topology. They showed that a more complete data set reveals that while the distribution of the connectivity in the Internet is still heavy-tailed, it deviates significantly from a strict PL. These findings thus challenge the extent of the validity of many PL relations found in the works of Group (i), as well as, the accuracy with which dynamical systems proposed in the works of Group (ii) (which predict the emergence of strict PL distributions) can model existing systems. Group (iii) avoids this debate and addresses the issue of Designer Complex Networks, i.e., vari- ants of the dynamical rules proposed in the work of Group (ii) can be used as the basic protocols for large-scale networks, resulting in the emergence of desired complex networks (e.g., networks with tunable PL exponents and multiple scale-free distributions), which can then be harnessed to provide global services in a robust fashion.

Whether or not PL distributions exist in existing complex networks, one thing is certain; these PL distributions will emerge if someone actually builds a network from scratch with the dynamical rules proposed in the works of Group (ii). Thus is PL (or other heavy-tailed distributions) are in fact desirable for a communication network, autonomous dynamical protocols can be designed to result in such connectivity structures in the emerging network. The works of *Group*(iii), is thus based on this idea; network formation protocols can be *designed* to ensure the existence of PL connectivity structure in the emergent complex networks.

The work presented in this paper, follows this philosophy. We show that an unstructured P2P communication system can exploit heavy-tailed connectivity structure of the network for scalable search. In separate works [21], we devise protocols that ensure the emergence of scale-free connectivity structures even in ad hoc and unreliable dynamical environments. Furthermore, we show how these protocols can take into account the heterogenous distribution of the resources between the nodes of the networks. For instance, the high-connectivity nodes (the hubs) in the emergent network can be guaranteed to be chosen from nodes with higher resources. Therefore, throughout this paper, we assume the network topology is a random PL network and we are only concerned about the performance of the proposed search algorithm on such networks.

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 [13], Limewire [19], 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 [25], Chord [24], PASTRY [20,11], and Viceroy [15], 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 Distributed Hash Table (DHT) 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 [22,18] have shown that the *structure of the existing networks has complex network characteristics, including approximate PL 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. (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. [18] 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 networks with heavy-tailed degree distributions, defined as those networks for which the *variance of the degree distribution diverges as* $\Omega(N^{\alpha})$, and in particular PL networks with exponents between 2 and 3, *when each node starts with a unique content, and queries are made randomly for any of these contents from any of the nodes* (Section 3). 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 involved 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 [23,18], are characterized by random PL and heavy-tailed degree distributions*. The scaling properties of our scheme for general heavy-tailed random networks are provided in Appendix B. While the analytical results are derived for random graphs on a given degree distribution, the conclusions remain valid for other classes of dynamically generated random graphs. This is verified (through simulations) for a particularly interesting class of such dynamical networks, introduced in [23], which guarantee the emergence of stable PL structures in unreliable environments. Finally, we provide both simulation and analytical studies of the improvements to be accrued from the percolation search algorithms when implemented on existing Gnutella crawl networks (Section 4).

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 [13] 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

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

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.

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

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 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. [14] 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 [14], 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, [14] 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 PL 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 PL 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, with some additional constraints* (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

[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 2000.

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 degree distribution of the network (e.g., τ can represent the PL exponent in a PL networks), and as will be proved shortly, is in general a sub-linear function of N for many heavy-tailed networks and in particular PL random graphs. 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*.

(ii) *Query implantation*: To start a query, a query request is *implanted* through a random walk again 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 of the network (see Appendix A for an introduction to the bond-percolation problem).

An instance of the percolation search algorithm is illustrated in Fig. 1.

We next derive scaling and performance measures of the above algorithm. We state detailed results for a PL random graph with an exponent $2 \leqslant \tau < 3$ for which the scalings can be explicitly found in terms of the network size and the maximum degree of the nodes. For more general heavy-tailed random graphs, we will quote the results in terms of the various moments of the degree distribution in Appendix B.

Our derivations will follow the following steps:

- Firstly, we define *high-degree* nodes and compute the number of high-degree nodes in a given network.
- Secondly, 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 a 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.
- Thirdly, 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, for PL random networks, we can examine 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 $2 \leqslant \tau < 3$, and indeed can be made to scale as $O(\log^2 N)$ with the proper choice of τ and $k_{\max}$, the maximum degree of the nodes in the network.

3.1. High-degree nodes

In this section, we define the notion of a high-degree node. Throughput, we assume that we deal with random PL graphs which have a degree distribution:

$$p_k = Ak^{-\tau},$$

where

$$A^{-1} = \sum_{k=2}^{k_{\max}} k^{-\tau} \approx \zeta(\tau) - 1,$$

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. For any node with degree k, we say it is a high-degree node if $k \geqslant k_{\max}/2$. Thus the number of

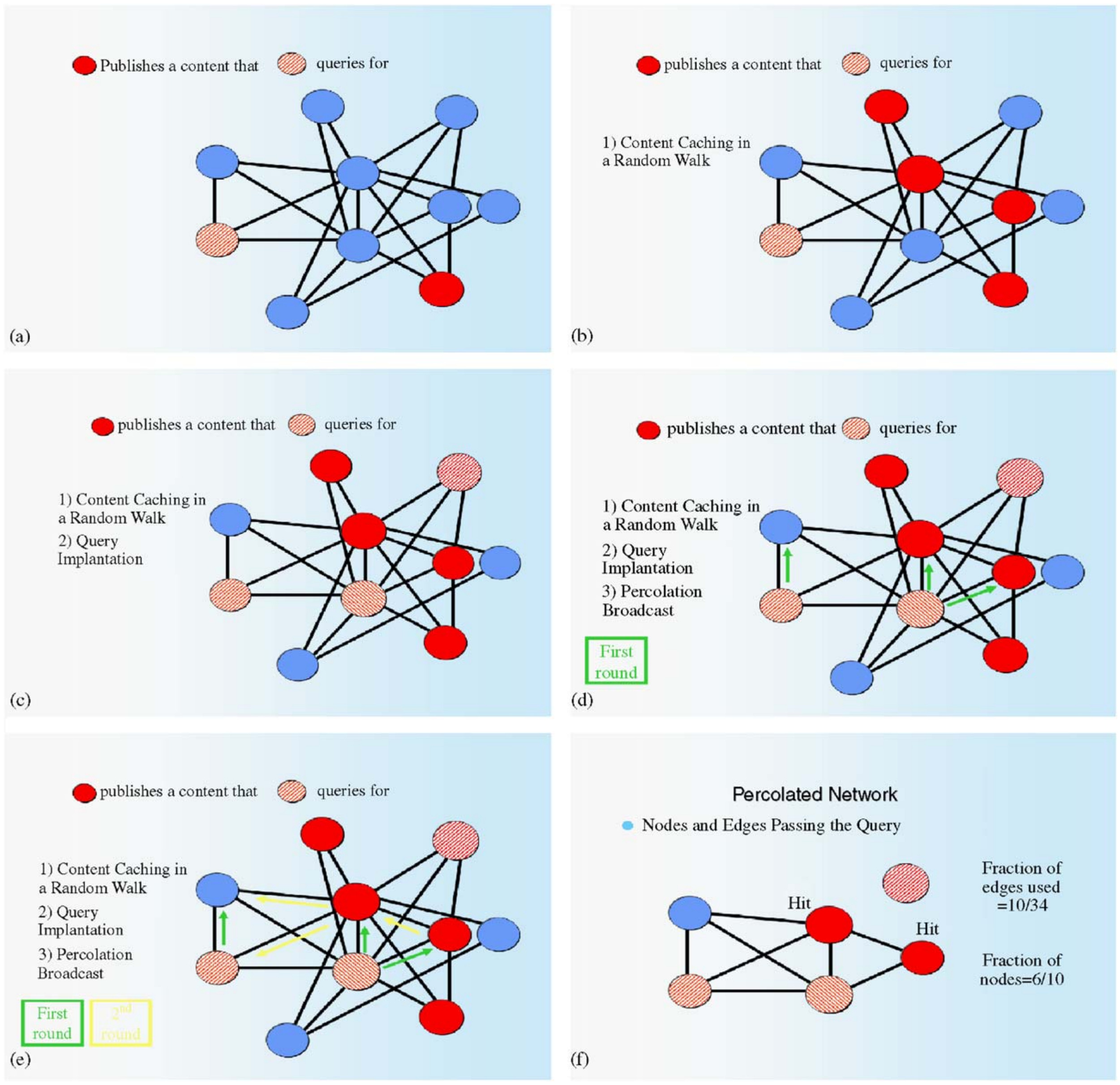


Fig. 1. Various phases of the percolation search algorithm. (a) The solid red node publishes a content which the dashed node will be looking for. (b) The initial caching phases, the red nodes is replicated in a random walk of length 4. (c) The start of the query, the dashed node trying to find the red node, will implant a query “seed” through a random walk. (d) Percolation search starts, any node with a query seed sends a query to a neighbors with probability $q = 1/3$. (e) This continues recursively, here there is only two recursions. (f) The set of all nodes who have received the search query. Two of these nodes also contain a replica of the red node, which means that there are two hits.

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 $\sum_{k=a}^{b} f(k) > \int_a^{b+1} f(k)\,\mathrm{d}k > \int_a^b f(k)\,\mathrm{d}k$ and $\sum_{k=a}^{b} f(k) < \int_{a-1}^{b} f(k)\,\mathrm{d}k$, we can bound H from above and below

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

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

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

We have shown that $H = \mathrm{O}(\frac{N}{k_{\max}^{\tau-1}})$. 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 $\mathrm{O}(N^{1/(\tau-1)})$. As we will discuss, we may also consider $k_{\max} = \mathrm{O}(N^{1/\tau})$. We should note that the first scaling law gives $H = \mathrm{O}(1)$, or a constant number of high-degree nodes as the system scales. The second gives $H = \mathrm{O}(N^{1/\tau})$. For all $\tau \geqslant 2$, 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 $q - q_c = \varepsilon > 0$, 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, where we deal with heavy-tailed networks (e.g., PL networks with $2 \leqslant \tau \leqslant 3$), $\lim_{N\to\infty} q_c = 0$ (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 $\mathrm{O}(\langle k\rangle Nq)$, we cannot afford to have a constant $\varepsilon > 0$ such that $q = \varepsilon + q_c$: the traffic will then scale linearly. The conventional analysis can be extended to show that even if $q = q_c/\gamma$ for a constant γ (thus, $\lim_{N\to\infty} q - q_c = 0$) one is still guaranteed to have a giant connected component in the percolated graph (see Appendix A).

We want to make our communications cost increase as slowly as possible. 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, including the scaling behavior of the size of the giant connected component(s) at a multiple of the percolation threshold. As for this paper, it suffices to prove the existence of a constant γ_τ independent of N for which a finite fraction of the high-degree nodes are within the same connected component after the percolation at q_c/γ for any $\gamma > \gamma_\tau$. Thus, any randomly picked high-degree node will be part of a connected component with a constant probability. This problem is briefly considered in the Appendix A which includes an introduction to the generating functions formalism for dealing with random graphs.

It remains to be shown that the diameter of the connected component is on the order of $\mathrm{O}(\log N)$. To see this, we use the approximate formula $l \approx \frac{\log M}{\log d}$ [17] of the diameter of a random graph with size M and average degree d. We know that the size of the percolated graph is $\frac{Nz}{k_{\max}}\langle k\rangle$ for some constant z, and the average degree is at least 2 (see Appendix A). Thus, the diameter of the giant component is

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

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 with a constant probability. Repeating the process by starting from randomly chosen high-degree nodes, the probability of the success can be made arbitrarily close to one with only a constant number of attempts. 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 $G_1(x)$ (see Appendix A) of the degree of the nodes arrived at by following a random link:

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

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

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

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

$$1 - (1 - P_\tau)^{\alpha/P_\tau} \geqslant 1 - \mathrm{e}^{-\alpha}.$$

Therefore, after $\mathrm{O}(1/P_\tau)$ 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 $\sum_{k=a}^{b} f(k) > \int_a^{b+1} f(k)\,\mathrm{d}k > \int_a^b f(k)\,\mathrm{d}k$ and $\sum_{k=a}^{b} f(k) < \int_{a-1}^{b} f(k)\,\mathrm{d}k$, 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

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

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

$$P_2 > \frac{\log 2}{\log(k_{\max})}. \tag{3}$$

For $\tau = 2$, then in $\mathrm{O}(1/P_2) = \mathrm{O}(\log k_{\max})$ 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

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

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

$$P_\tau > \frac{2^{\tau-2} - 1}{k_{\max}^{\tau-2} - 1}. \tag{4}$$

For $2 < \tau < 3$, then in $\mathrm{O}(1/P_\tau) = \mathrm{O}(k_{\max}^{\tau-2})$ 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 $\mathrm{O}(k_{\max}^{\tau-2})$ time steps to traverse $\mathrm{O}(k_{\max}^{\tau-2})$ edges, and hence, the query implantation time will dominate the search time, making the whole search time scale faster than $\mathrm{O}(\log N)$. Recall that the percolation search step will only require $\mathrm{O}(\log N)$ 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 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

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

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 $L(N, \tau) = \mathrm{O}(1/P_\tau)$ edges. When we make a query, if we reach the giant component, each edge passes it with probability q (if we do not reach a giant component only a constant number of edges pass the query). Thus, the total communications traffic scales at most as $qE = q_c\langle k\rangle N/\gamma$. Since $q_c = \langle k\rangle/\langle k^2\rangle$ we have $\mathcal{C}_\tau = \mathrm{O}(\frac{\langle k\rangle^2 N}{\langle k^2\rangle})$. For all $2 \leqslant \tau < 3$, $\langle k^2\rangle = \mathrm{O}(k_{\max}^{3-\tau})$. For $\tau = 2$, $\langle k\rangle = \log k_{\max}$ which gives

$$\mathcal{C}_2 = \mathrm{O}\left(\frac{\log^2 k_{\max} N}{k_{\max}}\right). \tag{5}$$

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

$$\mathcal{C}_\tau = \mathrm{O}(k_{\max}^{\tau-3} N). \tag{6}$$

In Section 3.1, we showed that the number of high-degree nodes $H = \mathrm{O}(N/k_{\max}^{\tau-1})$. We also know that $L(N, \tau) = \alpha/P_\tau$ and $P_2 = \mathrm{O}(1/\log k_{\max})$ and $P_\tau = \mathrm{O}(1/k_{\max}^{\tau-2})$. Thus we can rewrite the communication scaling in terms of the high-degree nodes, $\mathcal{C}_\tau = \mathrm{O}(L(N, \tau)^2 H)$. 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 $k_{\max}$ of a PL random graph

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

$$p_k = Ak^{-\tau}, 0 < k \leqslant k_{\max}, \tag{7}$$

where

$$A^{-1} = \sum_{k=1}^{k_{\max}} k^{-\tau}. \tag{8}$$

To construct the random PL graphs, N samples are then drawn from this distribution. For several reasons, the choice $k_{\max} = \mathrm{O}(N^{1/\tau})$ 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, $k_{\max} = \mathrm{O}(N^{\frac{1}{\tau-1}})$. 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

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)

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

maximum degree of a PL 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.

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.

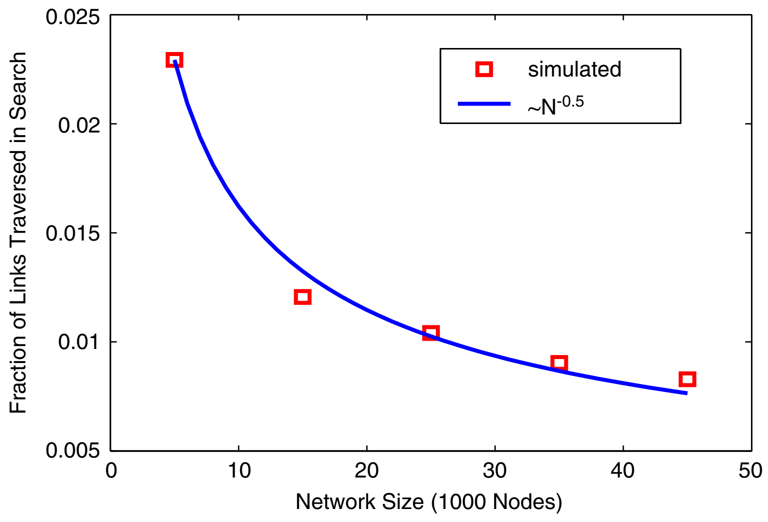
5. Simulations for dynamic PL networks

Our interest in PL random graphs is partly due to their ubiquitous existence in almost all examples of self-organized complex network systems, ranging from neural networks to the Internet. There is perhaps a more important side to the PL networks. Simple, local dynamical models can result in PL connectivity with tunable parameters, even in ad hoc and unreliable environments, as suggested by the authors in [23]. An scalable search strategy on PL networks, in combination with local protocols to emergence and maintenance of PL structures, can together provide an end-to-end solution to high-performance unstructured P2P networking.

Theoretical considerations in this paper, were based on a random network model on a given degree distribution. For networks generated with the dynamics in [23], we have verified the superior scaling properties of the percolation search algorithm through simulations. One such result is reported in Fig. 5. The network examined in Fig. 5 is created with the following dynamical model: starting from a small random core, (i) a new node is added at each time step. The new node will make 2 preferentially targeted links to the nodes already in the network. The probability that a node with degree k receives a connection is proportional to k. (ii) For any new node added, with probability $c = 0.6$, a randomly selected node and all its edges are deleted. (iii) If a node loses a link, it will initiate $n = 1$ preferentially targeted link compensation. The results of [23] suggest that the network grown with this dynamics will emerge into a PL graph with exponent $\tau = 1 + 2/(1 - c + 2nc) = 2.25$. Fig. 5 shows the traffic required to find 85% of all the contents *that exist* in the network at a given time.

6. Performance on heterogeneous random graphs

So far, we have assumed a uni-modal heavy-tailed distribution for the networks on which percolation search is to be performed. In reality, however, P2P networks are heterogeneous, consisting of categories of nodes with similar



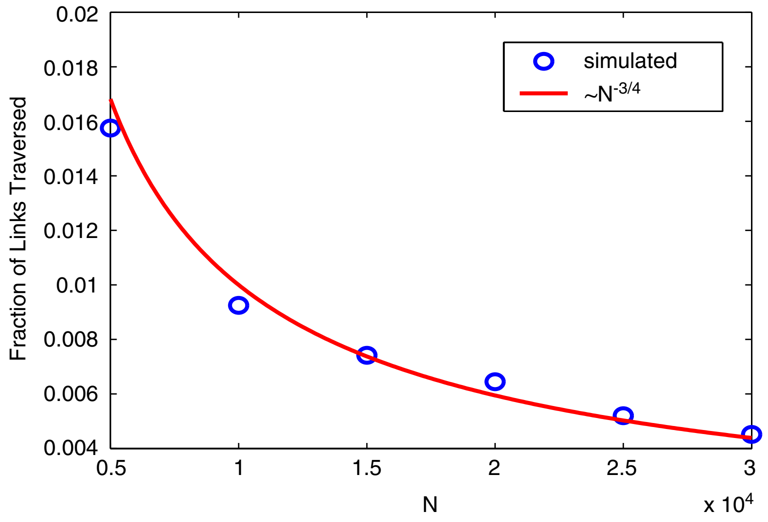


Fig. 2. Scaling behavior of the percolation probability required for a fixed hit-rate of 95% as a function of the network size for a network with $\tau = 2$. For the top figure, the maximum degree is scaled as $k_{\max} = 4N^{1/2}$ while it is scaled as $k_{\max} = N^{3/4}$ in the bottom figure. The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

capabilities or willingness to participate in the search process; e.g., *the dominant categories in existing P2P networks are, modems, DSL subscribers, and those connected via high-speed T-1 connections*. Thus, the degree distribution in a real network is expected to be a mixture of heavy-tailed (for nodes with high-capacity) and light-tailed (for nodes with lower capacity) distributions. We now show that the performance of the percolation search algorithm is not limited to the case of a uni-modal PL random graph. In fact, *the percolation search performs well as long as the variance of the degree distribution is much larger than its mean.*

Consider as an example the case of a bi-modal network, where a fraction x of the nodes have degree distribution P_k with a heavy tail, while the rest have a light-tailed degree distribution Q_k. Assume that the average degree of the two categories of nodes are the same for the sake of simplicity. The percolation threshold q_c^{bi} of this graph is then related to q_c the percolation threshold of a graph with the same degree distribution as of P_k as: $q_c^{\mathrm{bi}} = q_c/x$. Therefore, as long as a good fraction of all the nodes have a heavy tail, all observations of this paper still hold for a heterogeneous network. As far as the overall traffic is concerned, the total number of links traversed is at most $(xN)p_c^{\mathrm{bi}} = Np_c$ or the same as the case where all nodes had a heavy-tailed distribution P_k. The query and content implantation times are however a bit longer in this case.

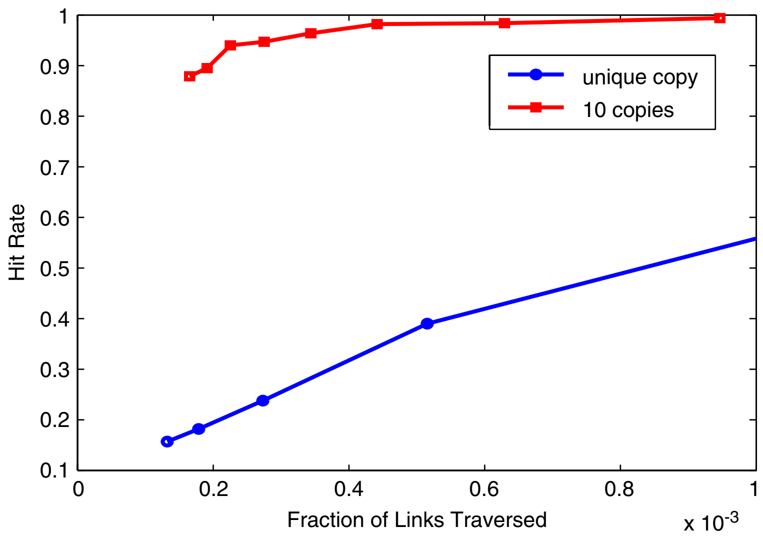


Fig. 3. Throughout the paper, we considered a unique replica of any content to exist in the network. The performance of the search algorithm greatly improves in the more realistic case where more than one copy of contents exist in the network. As an example, the case where only 10 replicas of any content is randomly spread in the network is considered in the above figure for a PL network with $\tau = 2$, $N = 30\,\text{K}$ and the average degree 6. Around 90% of all contents are found with only 0.02% of the original traffic.

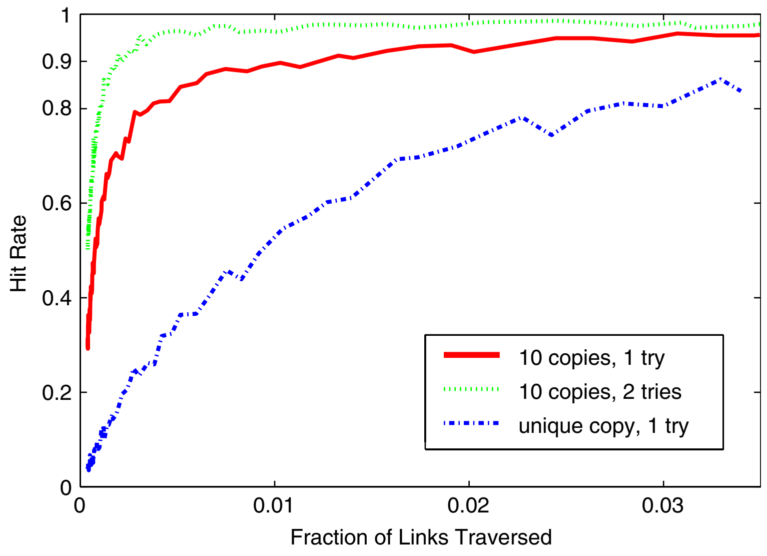


Fig. 4. Performance of the percolation search algorithm on a Gnutella crawl graph: the hit-rate as a function of the fraction of links used in search, for limewire crawl number 5. The crawl covers around 65,000 links and 10,000 nodes. The hit-rate for different number of trials are depicted separately. 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.

Heterogeneous networks, on the other hand, can naturally provide traffic shielding to low capabilities nodes as follows. Consider a network with say two categories of nodes. One with heavy-tailed degree distribution and the other being light tailed. The percolation search works by cutting out many links of the network, and *therefore almost all nodes participating in the search process are the ones that are highly connected, which are mostly part of the heavy-tailed group*. For instance, if the light-tailed group has exponential degree distribution, then the probability of any of node of the light-tailed category participating in the search process is exponentially small. Naturally then, *the nodes of the*

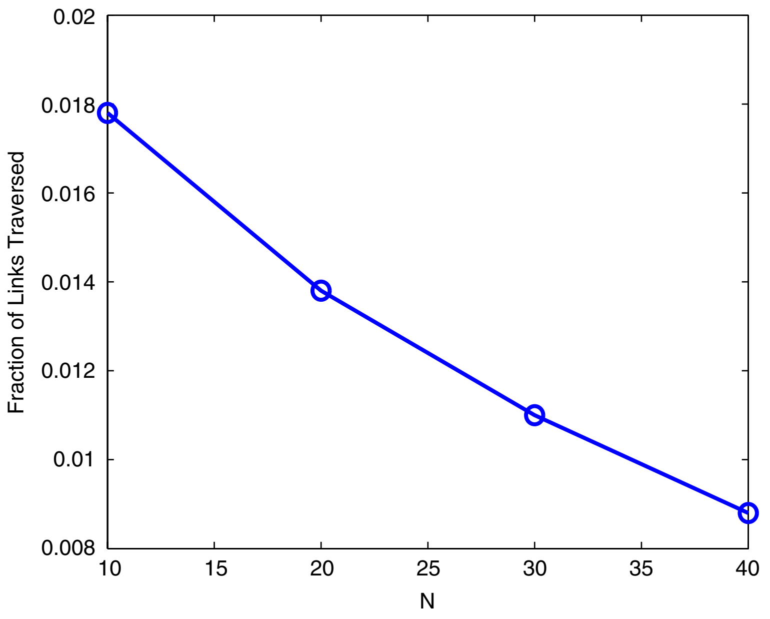


Fig. 5. The scaling of the fraction of links traversed to obtain 85% hit-rate as a function of the network size, for a PL dynamic network with $\gamma = 2.25$, constructed with deletion-compensation dynamics (see text for details). Caching is done with a constant TTL of 20.

Table 2
The fraction of *nodes* that participated in a search for a hit rate of 98%, in a network consisting of two power-law modes: 4000 nodes (called the heavy-tailed mode) have a power-law exponent $\tau = 2$ while 20,000 others (called the light-tailed mode) have an exponent $\tau = 4$

Heavy tailed	Light tailed	Overall
3.50e−2	2.22e−5	6.12e−3

TTL of 20 was used for both query and content implants.

light-tailed category are exempted from participation in the search process. Now if it is somehow ensured that only high-capability nodes are members of the heavy-tailed group, then the percolation search process naturally shields the low-degree nodes from the search traffic; see the following table for a typical simulation result (Table 2).

7. 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- 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.

Appendix A. The fraction of connected high-degree nodes in the percolated network

The percolation problem [16] on a graph can be described as follows: for any edge of the graph with probability q keep the edge and with probability $1 - q$ delete it. This is called bond percolation. Instead, if any node is deleted with probability $1 - q$ and is kept with probability q the problem is called the site percolation problem. A lesson from percolation theory is that usually for infinitely large graphs, there is a critical probability q_c such that for all $q > q_c$ the resulting network almost surely will not have any giant (infinite) connected component. Moreover, below that threshold the graph will almost surely have an infinitely large connected component.

A.1. Generating functions formalism

Consider a random graph with a specific degree distribution P_k, that is, the probability that a randomly chosen node has degree k is P_k. The generating function for x the degree of a randomly chosen node is

$$G_0(x) = \sum_{k=1}^{\infty} P_k x^k. \tag{A.1}$$

The generating function for the degree of a node found by following a random *link* would be:

$$G_1(x) = \frac{G_0'(x)}{G_0'(1)}. \tag{A.2}$$

Consider $H_0(x)$, the generating function for the size of the connected component that a randomly chosen node belongs to as well as $H_1(x)$ the generating function for the size of the connected component that a randomly selected link would belong to. If any edge is present with probability q (the site percolation) it can be shown that H_1, H_0 should satisfy the following set of equations:

$$H_1(x) = 1 - q + xG_1(H_1(x)), \tag{A.3}$$

$$H_0(x) = 1 - q + qxG_0(H_1(x)). \tag{A.4}$$

Consider now the average size of the connected components:

$$\begin{aligned}
\langle s_b \rangle &= H_0'(1) = qG_0(H_1(1)) + qH_1'(1)G_0'(H_1(1)),\\
H_1'(1) &= \frac{qG_1(1)}{1 - qG_1'(1)},\\
H_1(1) &= 1 \Rightarrow\\
\langle s_b \rangle &= q\left(1 + \frac{qG_1(1)G_0(1)}{1 - qG_1'(1)}\right),
\end{aligned}$$

where $\langle s_b \rangle$ is the average size of connected components when there is no giant connected component for bond percolation. This average diverges when $q_c = \frac{1}{G_1'(1)}$ corresponding to the phase transition and appearance (or disappearance) of a giant connected component. This is called the percolation probability

$$q_c = \frac{1}{G_1'(1)}. \tag{A.5}$$

The size of the giant connected component when present is found to be

$$S = 1 - H_0(1) = q\{1 - G_0(u)\}, \tag{A.6}$$

where u is the smallest positive solution to

$$u = 1 - q + qG_1(u). \tag{A.7}$$

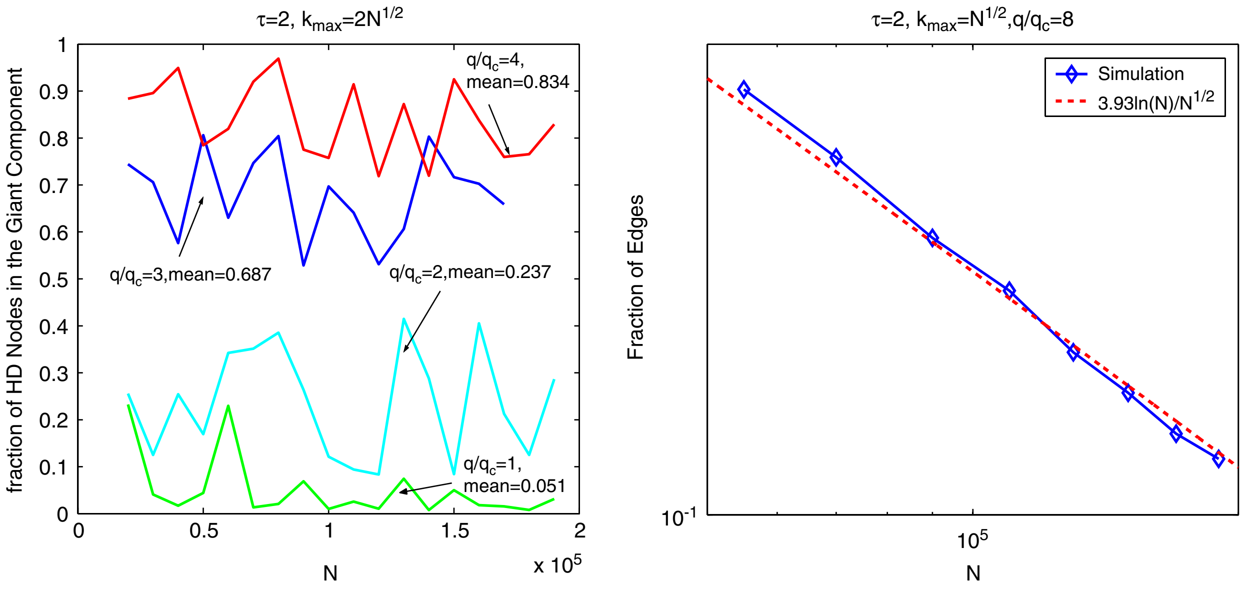


Fig. A.1. (Left) The fraction of the high-degree nodes in the giant connected component as a function of the network size for various values of q/q_c. For $q/q_c > 1$, this value is a constant almost independent of N. (Right) The scaling of the size of the largest connected component after percolation at a multiple of threshold $q/q_c = 5$.

A.2. Graphs with vanishing percolation threshold

We are interested in situations with q close to q_c to have as small messaging as possible while having a giant connected component. For finite network size N, the percolation threshold $q_c(N)$ might depend on N. In conventional percolation theory, one usually picks a *fixed* percolation probability q. If

$$q - \lim_{N\to\infty} q_c(N) = \varepsilon > 0, \tag{A.8}$$

then it can be shown that there exists a constant $0 < \lambda \leqslant 1$ such that the size of the connected component is at least λN as $N \to \infty$.

In contrast, we will derive the size of the connected component when the percolation probability $q(N)$ also depends on N and furthermore: $\lim_{N\to\infty} q(N) = \lim_{N\to\infty} q_c(N) = 0$. But: $\lim_{N\to\infty} \dfrac{q(N)}{q_c(N)} = k > 1$. Thus, we derive the size of the connected component when $q_c < q \ll 1$ by assuming u in (Eq. (A.7)) to be close to 1 and perform a Taylor series expansion of (Eq. (A.7)) around 1:

$$\begin{aligned} u &= 1 - \delta = 1 - q + q\{G_1(1-\delta)\} \\ &= 1 - q + q\{G_1(1) - \delta G_1'(1) + \delta^2 G_1''(1)\}, \end{aligned}$$

$$\delta = -q\delta G_1'(1) + q\frac{\delta^2 G_1''(1)}{2},$$

$$\delta = \frac{2(\frac{q}{q_c} - 1)}{qG_1''(1)}.$$

Inserting in (Eq. (A.6)) to find S when expanded around 1, one gets

$$S = \frac{2G_0'(1)(\frac{q}{q_c} - 1)}{qG_1''(1)} \simeq \frac{2G_0'(1)}{q_c G_1''(1)}. \tag{A.9}$$

The first equality in (Eq. (A.9)) is the characteristics of any infinite-dimensional percolation problem [8], that is, the size of the giant connected component linearly depends on $(q - q_c)$ for $q \to q_c^+$. In particular, for $q = 2q_c$, one gets $S \simeq \frac{G_1'(1)G_0'(1)}{G_1''(1)}$. For a PL exponent $\tau = 2$ and maximum degree $k_{\max} \gg 1$, and assuming $q_c/q = \gamma$, the size of the connected component would be $S \simeq 2(1-\gamma)\gamma^{-1}\frac{\log(k_{\max})}{k_{\max}}$, while for $2 < \tau < 3$, $S \propto \frac{1}{k_{\max}}$.

Further analysis (dropped from this appendix for brevity) is required to show that for any $2 \leqslant \tau < 3$, there exists a constant γ_τ such that for any $\gamma > \gamma_\tau$ a constant fraction of all the high-degree nodes (the nodes with degree greater than $k_{\max}/2$ belong to the same component). These are also verified through simulation in Fig. A.1.

Appendix B. Scaling properties for general heavy-tailed random graphs

The scaling properties of the percolation search algorithm for heavy-tailed PL random networks where derived in details in the text. In this Appendix, we drive the scaling properties for more general heavy-tailed random graphs. This will prove useful in characterizing, in broader terms, other properties of random graphs (apart from being heavy tailed) necessary for the success of the percolation search.

Consider a general random graph for which the generating function of the degree distribution and that of the degree of the nodes arrived at by following random links are $G_0(x) = \sum_{k=1}^{\infty} p_k x^k$, $G_1(x) = \frac{G_0'(x)}{G_0'(1)}$, respectively.

Let us assume that the graph has a finite mean (independent of N) and a variance that scales as $\langle k^2 \rangle = \Theta(G_1'(1)) = \Theta(N^\alpha)$ for some constant $0 < \alpha < 1$.

The percolation threshold is $q_c^{-1} = G_1'(1) = \Theta(\langle k^2 \rangle)$. Thus, the total number of links after the percolation is at most: $Nq_c = \mathrm{O}(N/G_1'(1))$. From (A.9) the total number of links in the giant component after the percolation is $\Theta\left(N\frac{G_1'(1)}{G_1''(1)}\right)$. Thus the probability of any remaining link after percolation to belong to the giant component is $\Omega\left(\frac{G_1''(1)}{(G_1'(1))^2}\right)$. Therefore, sampling at most $M = \mathrm{O}\left(\frac{(G_1'(1))^2}{G_1''(1)}\right)$ nodes that have any edges after the percolation, one can find a node who is part of the giant connected component.

Next, to find the size of the content and query implantation phases, we need to find *R*, the probability of a randomly followed link to arrive at a node who will have a link left after the percolation. Since the percolation is at a multiple of threshold $q_c = G_1'(1)^{-1}$, we would need to find the probability that a randomly followed link would arrive at a node with degree $\Theta(G_1'(1)) = \Theta(\langle k^2 \rangle)$. Consider the probability distribution of the degree of the nodes arrived at by following a random link, the generating function of which is $G_1'(x)$. The probability *R* is the probability that sampling this distribution will yield a value greater than the mean of this distribution.

Putting it all together, the scaling of the traffic would be as

$$T = \mathrm{O}\left(N\frac{\langle k \rangle}{\langle k^2 \rangle}\right)$$

while the scaling of the number of query and content implants are

$$C = \mathrm{O}\left(\frac{G_1''(1)}{RG_1'(1)^2}\right) = \mathrm{O}\left(\frac{\sum_{k=1}^{\infty} k^3 p_k}{\left(\sum_{k=1}^{\infty} k^2 p_k\right)^2 \sum_{k=\alpha\langle k^2 \rangle}^{\infty} k^2 p_k}\right)$$

for an arbitrary positive constant $\alpha < 1$. PL random graphs have the useful property that the probability *R* is not too small. In fact when $\tau = 2$, this probability would be in the order of one, and as such, it would only take a few steps for an implantation procedure to find a high-degree node.

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, B. Huberman, Search in power-law networks, Phys. Rev. E 64 (2001) 046135.
[5] W. Aiello, F. Chung, L. Lu, A random graph model for massive graphs, Proc. 32nd Annu. ACM Symp. on Theory of Computing, ACM Press, New York, 2000, pp. 171–180.
[6] W. Aiello, F. Chung, L. Lu, Random Evolution in Massive Graphs, IEEE Symp. on Foundations of Computer Science, 2001.
[7] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Rev. Modern Phys. 74 (2002) 47–97.
[8] A.-L. Barabási, R. Albert, H. Jeong, Mean-field theory for scale-free random networks, Physica A 272 (2000) 173–187.
[9] J.M. Carlson, J. Doyle, Hot: robustness and design in complex systems, Phys. Rev. Lett. 84 (2000) 2529–2532.
[10] Q. Chen, H. Chang, R. Govindan, S. Jamin, S.J. Shenker, W. Willinger, The origin of power laws in internet topologies revisited, Proc. of IEEE Infocom 2002, 2002.
[11] P. Druschel, A. Rowstron, PAST: a large-scale, persistent peer-to-peer storage utility, Proc. Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS), Schoss Elmau, Germany, May 2001.
[12] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the internet topology, SIGCOMM, 1999, pp. 251–262.
[13] T. Klingberg, R. Manfredi, Rfc-Gnutella, http://rfc-gnutella.sourceforge.net
[14] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, Proc. 16th Internat. Conf. on Supercomputing, ACM Press, New York, 2002, pp. 84–95.
[15] D. Malkhi, M. Naor, D. Ratajczak, Viceroy: a scalable and dynamic emulation of the butterfly, Proc. 21st Annu. Symp. on Principles of Distributed Computing, 2002, pp. 183–192.
[16] M. Molloy, B. Reed, A critical point for random graphs with a given degree sequence, Random Struct. Algorithms 6 (1995) 161–179.
[17] M.E.J. Newman, S.H. Strogatz, D.J. Watts, Random graphs with arbitrary degree distributions and their applications, Phys. Rev. E 64 (2001) 026118.
[18] M. Ripeanu, I. Foster, A. Iamnitchi, Mapping the Gnutella network: properties of large-scale peer-to-peer systems and implications for system design, IEEE Internet Comput. J. 6 (1) (2002).
[19] C. Rohrs, Limewire design, lime wire documents: http://www.limewire.org/project/www/design.html, 2001.
[20] A. Rowstron, P. Druschel, Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility, Proc. 18th ACM Symp. on Operating Systems Principles (SOSP'01), Chateau Lake Louise, Banff, Canada, October 2001.
[21] N. Sarshar, V.P. Roychowdhury, Multiple power-law structures in heterogeneous complex networks, Phys. Rev. E 72 (020101) (2005).
[22] S. Saroiu, S.D. Gribble, P. Krishna Gummadi, A measurement study of peer-to-peer file sharing systems, Proc. Multimedia Computing and Networking (MMCN), January 2002.
[23] N. Sarshar, V. Roychowdhury, Scale-free and stable structures in complex ad hoc networks, Phys. Rev. E 69 (026101) (2004).
[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, Proc. ACM SIGCOMM 2001 Technical Conf., San Diego, USA, August 2001.
[25] B.Y. Zhao, J.D. Kubiatowicz, A.D. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, Technical Report CSD-01-1141, U. C. Berkeley, April 2001.