Absence of Evidence as Evidence of Absence: A Simple Mechanism for Scalable P2P Search

Stratis Ioannidis
University of Toronto
Toronto, ON, Canada
Email: stratis@cs.toronto.edu

Peter Marbach
University of Toronto
Toronto, ON, Canada
Email: marbach@cs.toronto.edu

***Abstract*—We propose a novel search mechanism for unstructured p2p networks, and show that it is both scalable, *i.e.*, it leads to a bounded query traffic load per peer as the peer population grows, and reliable, *i.e.*, it successfully locates all files that are (sufficiently often) brought into the system. To the best of our knowledge, this is the first time that a search mechanism for unstructured p2p networks has been shown to be both scalable and reliable. We provide both a formal analysis and a numerical case study to illustrate this result. Our analysis is based on a random graph model for the overlay graph topology and uses a mean-field approximation to characterize the evolution of how files are replicated in the network.**

I. Introduction

In this paper, we consider an unstructured peer-to-peer (p2p) network, whose purpose is to allow peers to share their files. A peer interested in a file propagates a query over the p2p network. If the query reaches a peer that has the file, this peer notifies the one that issued the query and lets it download a copy. Finally, a peer that downloads such a copy shares it for the remainder of the time it stays in the system.

Due to their inherent simplicity, unstructured peer-to-peer systems have many attractive features: they require very little maintenance, are very efficient under high churn and, by the arbitrary nature of their topology, are also highly resilient under peer failures.

However, searching (*i.e.*, propagating query messages) in unstructured systems can generate high traffic overheads, thereby rendering such systems unscalable. For example, it is well-known that the naïve flooding mechanism used in early Gnutella implementations does not scale [1]. To address this, an extensive body of work has focused on designing search mechanisms that reduce query traffic [2]. Random walk and expanding ring mechanisms have attracted the most interest because they can be easily implemented in a distributed way. Moreover, they indeed reduce the amount of traffic incurred by queries for files that are available in the network, *i.e.*, are stored by at least one peer [3]–[5].

But random walk (and expanding ring) search mechanisms still generate a high query traffic overhead when the requested file is not stored by any peer. We show this formally in Section VI, as well as through simulations in Section VII. Unfortunately, searches for unavailable files are very common in practice (see, *e.g.*, the measurement study in [6]). As a result, unstructured p2p networks using such search mechanisms do not scale, precisely because of the high traffic overhead generated by queries for files not in the system.

One approach used in practice to reduce the amount of query traffic generated by searches for unavailable files is to set the query time-to-live (TTL) to a small value (see also our discussion in Sections VI-A and VII-B). However, queries with a small TTL can only reach a small fraction of the peers in the system. Consequently, it has been observed that a search is unlikely to succeed, even if the requested file is actually available in the network [6], [7]. Indeed, query success rates in actual systems are typically around 10-20% [8].

The reason why a small TTL leads to such a low success rate is that it limits the search radius of both searches for files that are in the system and files that are not. This suggests that a more efficient approach would be to only limit queries for unavailable files without affecting queries for files in the system. This would make the system scalable (by eliminating long, unsuccessful searches) without affecting the success rate of queries for available files.

Therefore, a fundamental question in the design of search mechanisms for unstructured p2p networks is whether this can be done. *I.e.*, is it possible to limit searches for unavailable files without reducing the query success rate for files that are in the system? The aim of this paper is to study this issue. Surprisingly, we find is that this can indeed be done.

The following provides some intuition as to why this is true. If peers knew a priori whether a file is in the system or not, limiting searches for unavailable files would be trivial: peers simply need not search for such files, as they have no chance of finding them. Of course, it is not clear how to determine whether a file is available before searching for it. On the other hand, *after* searching for a file, a peer has more information about it than before: a failed search suggests that it is likely to not be in the system. In this sense, the outcome of a search can help in determining whether a file is available or not.

This observation leads us to consider the following mechanism. If a peer fails to locate a file, it treats the failed query as "evidence of absence" of the file. It then stores this information and shares it with other peers, for as long as it stays in the system. That is, whenever it receives a query for this same file, it stops the search and informs the peer that issued the query that the file is (likely to be) unavailable. The peer that receives this information treats it again as "evidence of absence" of the

file, and shares this information with other peers in the same manner. Basically, the system treats the *absence of evidence* (the failure to locate the file) as *evidence of absence* (evidence that the file is not available).

The main contribution of this paper is to show that the above mechanism indeed succeeds at stopping searches for files that are not in the system early on, without jeopardizing the query success rate. More precisely, we show that the proposed search mechanism is (a) *scalable*, *i.e.*, it keeps the query traffic load generated by searches for any file bounded at each peer, and (b) *reliable*, *i.e.*, it is able to locate all files that are brought into the system sufficiently often. We obtain this result both through a formal analysis, on a mathematical model of unstructured p2p networks that we introduce in Section IV, as well as by numerical results, presented in Section VII.

In order to keep the discussion as concrete as possible, we focus on the application of the above mechanism to unstructured p2p file sharing systems. We note however that our approach can be applied to searching over other unstructured systems as well, such as a decentralized BitTorrent tracker [9] or a universal swarm.

II. Related Work

There exists an extensive literature on search mechanisms for unstructured p2p systems (see, *e.g.*, the survey by Risson and Moors [2]). Roughly, the proposed search mechanisms can be classified into two categories: (a) search mechanisms based on passive replication, whereby a file is replicated only at peers that request and download it, and (b) search mechanisms based on proactive replication, whereby peers proactively replicate their files at peers that have not requested them.

a) Search Mechanisms using Passive Replication: Most search mechanisms with passive replication are based on a random walk or an expanding ring search mechanism. The random walk search mechanism was first proposed and analyzed by Lv *et al.* [3]. In their analysis, Lv *et al.* [3] studied the performance of random walk searches for files that are available in the system using a model based on uniform sampling. This model significantly simplifies the analysis of a random walk, as it allows to ignore the impact of the network topology on the amount of query traffic generated. Gkantsidis *et al.* relate the cover time of a random walk to the network's relaxation time [10] and, as such, take the network topology into account. Our analysis follows a similar approach, by using well-known results on the relaxation time appearing in the monograph by Aldous and Fill [11].

Though the random walk performs very well in terms of incurred query traffic, it can lead to poor performance in terms of query response times (delay). Several variants of the random walk have been proposed to address this [3], [12], [13].

The expanding ring search mechanism can also significantly improve the delay performance, without increasing the query traffic considerably [3]. Our results can be extended to other search mechanisms, that fare better in terms of delay, such as the expanding ring; however, the analysis becomes more involved and, as such, is beyond the scope of this paper.

b) Search Mechanisms using Proactive Replication: One approach to reduce query traffic in unstructured p2p systems is to proactively replicate files in the network [13]–[17]. This increases the availability of a file, as peers that neither requested it or brought it into the system may have a copy. Increasing the number of replicas of a file decreases the traffic generated by queries for the file, so there is a fundamental trade-off between query traffic and replication traffic [13], [16]. Proactive replication under constraints on the peers' storage capacity has been studied by Cohen and Shenker [14] and Tewari and Kleinrock [5].

As proactive replication increases the availability of a file, search mechanisms with proactive replication generate less query traffic compared with mechanisms that use passive replication. However, as such mechanisms can only replicate files that are in the system, the traffic generated by queries for unavailable files is still high.

c) Dealing with Absent Files: The search mechanisms based on passive and proactive replication have been shown to work well for files available in the system, but they can lead to a large amount of query traffic for unavailable files. As a result, reducing the query traffic generated by unavailable files remains a fundamental open question.

One proposed approach to reduce the query traffic generated by unavailable files, and to the best of our knowledge, the only approach that has been presented in the literature, is to use a hybrid architecture [6], [7], [18]–[20], consisting both of an unstructured and a structured p2p network. Queries for files that are very likely to be in the system are served by the unstructured p2p network, whereas queries for rare files are handled by the structured p2p network.

A challenge posed by the above approach is how to obtain the information on file availability that is needed to decide whether a query should be served by the unstructured or the structured network. In [7], [18], Loo *et al.* propose that each peer estimates file availability by monitoring its local query traffic for popular terms and inferring from them which files are widely available in the system. This approach has a potential pitfall, as files which are often requested may not necessarily be widely replicated (see, *e.g.*, our discussion in Section III). Zaharia *et al.* propose collecting global statistics on the availability of files through a gossiping mechanism [20]. Note that obtaining statistics through gossiping introduces an additional traffic overhead.

Our approach, namely, using "absence of evidence" as "evidence of absense", is simpler than the above as (a) it does not require an additional infrastructure for serving queries for rare files and (b) it does not require to proactively collect and maintain information about file availability.

III. Properties of Unstructured P2P Networks

In this section, we provide a summary of some key properties of unstructured p2p networks and discuss how they are captured in our model. A complete description of our model can be found in the next section (Section IV).

A. System Size

An important characteristic of unstructured p2p networks is that they can grow significantly over time [21], [22], reaching population sizes in the order of millions [21]. This growth occurs over long-term periods of time, such as months or years [21], [22]. Over shorter periods of time, such a days [23] or weeks [24], [25], the population size is relatively stable, tending to oscillate around a fixed operating point.

In our analysis, we consider p2p networks for which the population size is constant and equal to n. More precisely, we assume that, whenever a peer departs, it is immediately replaced by a new peer, keeping thus the system size fixed. By fixing n, we characterize system behavior over short-term periods of time. To capture long-term growth, we consider a sequence of unstructured p2p networks of increasing size n.

Note that assuming that the number of peers in the system is constant and equal to n (over a short-term period) is a simplification. In our numerical study, appearing in Section VII, we relax this assumption and let the system size oscillate around an operating point n.

B. File Popularity and Availability

Two other important aspects of unstructured p2p networks are file "popularity", *i.e.*, the fraction of new peers that request the file when they join the system, and file "availability", *i.e.* the fraction of peers in the network that have a copy of the file. The following two observations were made on actual unstructured p2p systems:

1) The popularity and availability of a file varies very little within short time periods such as a day or a week [26], [27], though both can change significantly over longer periods of time [26].
2) There is almost no correlation between the popularity and the availability of a file: it is quite common that a file is very popular (*i.e.*, a large fraction of new peers request this file) but hardly available (*i.e.*, only a small fraction of peers has a copy of the file), and vice versa [27].

The difference between file popularity and availability can be modeled through a discrepancy between the number of peers that "publish" a file, *i.e.*, bring it in the system with the purpose of sharing it, and the number of peers that request it. One reason why a very popular file may be published by only a small fraction of peers is "free-riding": peers may not share content they downloaded in the past. Note that, for a file to be available, it has to be brought into the system (published) by at least one peer. If only a small fraction of peers bring a given file into the system when they join, then queries for this file are likely to fail, and only a small fraction of peers will be able to locate and download the file. As a result, the availability of this file will be small, even if it is very popular and a large fraction of peers request it.

To capture the above aspect of unstructured p2p systems in our analysis, we associate two quantities with each file: the *publishing probability* q and the *request probability* p. The request probability models the file popularity, *i.e.*, the fraction of new peers that request the file. The publishing probability captures the fraction of peers that bring the file into the system when they arrive. We treat all arriving users as new users; in reality, users that publish the file can in fact be returning users, that downloaded the file at some time in the past. In this sense, q can be interpreted as the likelihood that a peer downloaded the file in the past *and* is willing to share it.

We allow probabilities p and q to be functions of the system size n, *i.e.*, we allow p and q to change as the system grows (over a longer time period). We denote by $p(n)$ and $q(n)$ the request and publishing probability of a file for the system of size n. Note that, in a system of size n, the expected number of peers that requested the file upon their arrival is equal to $np(n)$, and the expected number of peers that brought the file into the system is $nq(n)$. If, *e.g.*, $q(n) = 0.01$, then 1% of all peers publish the file, in expectation. The expected number of peers that publish file is thus $0.01n$ and increases linearly as the system size grows. If $q(n) = c/n$, where c a constant, only c peers publish the file, in expectation. In the case where $q(n)$ decays faster than $1/n$, *e.g.*, $q(n) = 1/n^2$, the expected number of peers that publish the file *decreases* as the system size grows, *i.e.*, fewer and fewer peers publish it. Similar observations can be made, w.r.t. $p(n)$, about the expected number of peers that request the file.

C. Overlay Network

When new peers join an unstructured p2p network, they typically connect to a random subset of existing peers [21]. As a result, the topology of the overlay graph, *i.e.*, the network formed by the peers in the system, is highly dynamic.

Another characteristic of overlay graphs in unstructured p2p networks is that the link degrees of peers tend to be concentrated around a single value [21]. For example, it has been observed that link degrees in unstructured p2p networks based on the Gnutella protocol tend to be concentrated around 32 connections, with no more than 6% of peers exceeding this number [21]. This is because (a) implementations of unstructured p2p networks typically set a limit to the number of connections each peer maintains (to limit the overhead) and (b) peers actively try to maintain a number of connections as close to this limit as possible, by establishing new connections when old ones are dropped.

For our analysis, we assume that each peer has the same degree d and model the overlay graph as a connected d-regular graph [28], where d is even. Furthermore, we assume that the overlay graph belongs to a particular class of d-regular graphs, namely the class of $\mathcal{H}_{n,d}$ graphs (see for example [28], [29]). There are two reasons for making this assumption. First, practical protocols for peers joining an unstructured p2p network [10], [30] lead to a $\mathcal{H}_{n,d}$ graph (see our discussion in Section VII). Second, with high probability, a random d-regular graph of size n is a $\mathcal{H}_{n,d}$ graph [28]. In this sense, the above model describes the behavior under general d-regular graphs (with high probability).

The class $\mathcal{H}_{n,d}$ is as follows. Each graph in the class consists of $d/2$ Hamilton cycles, *i.e.*, cycles of length n that contain all

peers [28]. These cycles are superimposed, and the result is the d-regular graph. The class $\mathcal{H}_{n,d}$ contains all graphs that can be constructed this way (*i.e.*, all graphs that can be decomposed into $d/2$ Hamilton cycles).

IV. Model

We will use the following model for our analysis.

A. Network Model

To study the scalability of the proposed search mechanism, we consider a sequence of unstructured peer-to-peer networks indexed by n, $n \geq 1$, where n is the number of peers connected to the network. More precisely, we assume that peers join and leave the system as described below.

Initially (at time 0), the network that consists of n peers. Each peer stays in the system for an exponentially distributed time (lifetime) with mean $1/\mu$, independent of the lifetimes of all other peers in the system. When a peer departs, it is immediately replaced by a new peer that again has an exponentially distributed lifetime with mean $1/\mu$, independent of the lifetimes of all other peers that are, or have been, in the system. Note that under this process the total number of peers in the system is at all times equal to n.

The overlay graph is dynamic: we assume that incoming and departing peers use a protocol (as, *e.g.*, in [30]) to join and leave the network so that, at any point in time, the network is an $\mathcal{H}_{n,d}$ graph.

B. Request and Publishing Probabilities

We assume that peers may issue queries for $M(n)$ distinct files, that may or may not be present in the system. A file j, $j = 1, \ldots, M(n)$ is requested by an incoming peer with a probability $p_j(n)$, and brought into system by a new peer with a probability $q_j(n)$. We call p_j the *request probability* of file j. The expected number of queries an incoming peer issues and the expected number of files it brings is given by $\sum_{i=1}^{M(n)} p_i(n)$ and $\sum_{i=1}^{M(n)} q_i(n)$, respectively.

Without loss of generality, we focus in our analysis on a single file. In particular, we characterize the query traffic load generated by queries for a single file j. The total traffic load generated by all $M(n)$ types of queries can be obtained by summing the individual loads generated per file, as discussed in Section VIII. We therefore omit the index in our analysis and denote the request and the publishing probabilities by $p(n)$ and $q(n)$, respectively.

For simplicity, we assume that all files are requested by a peer immediately when it enters the system. However, our analysis and our results can be extended to the case where peers issue a query for a file at an epoch uniformly distributed over a peer's lifetime (see [31]).

C. Search Mechanism

To define the precise operation of our search mechanism, we use the following notation. For a given file, we distinguish among three different types of peers: *positive* peers, which are peers that have a copy of the file, *negative* peers, which are peers that believe that the file is not in the system, and *null* peers, which are peers that neither have the file nor believe it is not in the system. Peers that publish the file, *i.e.*, that already have a copy of the file when they enter the network, are always positive. Peers that request the file can become either positive or negative, depending on the outcome of the search. Finally, peers indifferent to the file, that neither request it or bring it in the system, are null.

The search mechanism then works as follows. In order to locate a file, peers propagate a query over the overlay network using a random walk mechanism with TTL. That is, a peer issuing a query (the source peer) chooses one of its d neighbors in the overlay network at random and forwards a query packet to it. Each query contains an expiration time field that is initialized to a predefined value $\text{TTL}(n) = \Theta(n)$ by the source peer (*i.e.*, the TTL is proportional to the size of the p2p network). When a query reaches a positive peer, *i.e.*, a peer that has a copy of the file, the positive peer responds to the source peer by allowing it to download a copy of the file, thus converting it to a positive peer. When a query reaches a negative peer, the negative peer responds by informing the source peer that the file is (likely) not in the system, thus converting it to a negative peer. The query propagation is thus terminated when the query reaches either a positive or a negative peer. When a query reaches a null peer, the null peer first reduces the expiration time by one; if the resulting expiration time is larger than 0, then the null peer continues to forward the query by choosing one of its d neighbors at random, and sending the query to it. On the other hand, if the expiration time is equal to 0, then the null peer terminates the query and informs the source peer. Such a query is considered failed, and the source peer becomes negative.

Note that the additional storage overhead introduced by this mechanism is minimal. In particular, a peer can become negative *only if it first requests the file*. In other words, the only peers required by the above mechanism to store the "evidence" of a file's absence are the ones that were willing to store and share the entire file in the first place.

We require that the TTL value grows proportionally to the system size n, *i.e.*, $\text{TTL}(n) = \Theta(n)$, but not that it is exactly equal to n. The peer issuing the query thus merely needs to have an estimate of the system size. There are known distributed algorithms for obtaining such an estimate at a small overhead (see, *e.g.*, [32]). Moreover, such algorithms can be executed infrequently (*e.g.*, once a day), as the system size grows slowly.

For our analysis, we assume that (a) each transmission of a query packet is exponentially distributed with mean δ time units, (b) the network topology does not change during a query propagation, and (c) at the start of each query propagation the network topology is independent from the network topology at previous searches, and chosen uniformly at random from the set of all $\mathcal{H}_{n,d}$ graphs [29].

The assumption that the network topology does not change during the query propagation simplifies the analysis; we relax this assumption in our numerical study in Section VII,

validating our results even when the topology changes while queries are propagated. Similarly, the assumption that the overlay graph is independent between consecutive searches is unrealistic. It introduces more "randomness" into the overlay graph topology than there exists in a real system: as arrivals and departures affect the network only "locally", overlay topologies between searches might overlap. Again, we relax this assumption in our numerical study, validating our results even when the topology changes only locally.

D. Performance Metrics

In our analysis, we characterize the performance of the proposed search mechanism as a function of the network size n. In particular, for a given a file with request probability $p(n)$ and publishing probability $q(n)$, we characterize the performance in terms of the *average load per peer* $\rho(n)$, which is defined as the expected number of query packets that a peer has to forward per unit time, and the *query success rate* $\gamma(n)$, which is the steady state probability that a query for this file is successful (and the source peer becomes positive).

We say that the search mechanism is *scalable* if the average load per peer $\rho(n)$ stays bounded as the system size n increases, *i.e.*, we have

$$\rho(n) = O(1). \tag{1}$$

for all possible combinations of $p(n)$ and $q(n)$.

In addition, a search mechanism should be considered reliable if queries for files that are available in the system are successful. In the following, we will use a weaker notion of reliability: we only require that queries for files which are brought into the system sufficiently often are successful. More precisely, we will say that a search mechanism is *reliable* if

$$\text{if } q(n) = \omega\left(\frac{1}{n}\right) \text{ then } \lim_{n\to\infty} \gamma(n) = 1, \tag{2}$$

for all possible values of $p(n)$.

Keeping in mind that $nq(n)$ is the expected number of peers in the system that publish the file (*i.e.*, already had a copy of the file when they entered the system), $q(n) = \omega\left(\frac{1}{n}\right)$ means the expected number of peers in the system that publish the file increases with n; however, this increase can be arbitrarily slow. For example, the expected number of peers in the system that publish the file can grow as slowly as $\log\log(n)$, or even slower. The above definition is therefore just slightly weaker than requiring that queries are successful if there exists at least one peer in the system that publishes the file.

V. Main Result

In our main result, we show that the search mechanism that we proposed is both scalable and reliable. It is interesting to contrast this result with the performance of the traditional random walk search mechanism that uses a TTL value, but does not share information about failed queries. In Section VI-A, we show that this mechanism cannot be both scalable and reliable, no matter what value is used for the TTL. More precisely, in order to obtain a bounded average load per peer for the traditional random walk search mechanism, the TTL value has to be of the order $O(1)$, *i.e.*, it has to stay bounded as the system size grows. However, queries under such a TTL value will reach only a small fraction of the network; they may succeed only if the publishing probability q is constant, *i.e.*, the number of peers in the network that publishes the file grows linearly in n. In light of the above, the fact that the proposed search mechanism with "evidence of absence" leads to a bounded average load per peer for all possible functions $p(n)$ and $q(n)$, while almost all queries succeed as long as $q(n) = \omega(1/n)$, is quite remarkable.

We discuss the above results in more detail in Sections V-A and V-B. Due to space constraints, we formally state our main results in Section VI without proofs; the full analysis can be found in our technical report [33]. The following discussion provides some insight why these results hold.

A. Scalability

We can get some intuition into why the system is scalable by approximating a random walk search with uniform sampling: whenever we send a query to a new peer we pick this peer uniformly at random from all peers in the network.

Recall that a query stops either when it reaches a positive or negative peer, or when its expiration time becomes zero. Assume for simplicity that $\text{TTL}(n) = n$. Consider a file with request and publishing probabilities $p(n)$ and $q(n)$ such that

$$p(n) + q(n) \geq \frac{1}{n}.$$

Then, at each step of the query propagation under uniform sampling, the probability that we hit a positive or negative peer is equal to $p(n)+q(n)$, and the expected time to hit a positive or negative peer is equal to

$$\frac{1}{p(n) + q(n)} \leq n = \text{TTL}.$$

Therefore, each query for this file will generate a total query traffic of (roughly) $1/(p(n) + q(n))$ packets. This traffic is generated whenever a new peer that enters the system requests the file. Hence, the total query traffic per unit time that is generated by searches for this file is equal to

$$\frac{n\mu p(n)}{p(n) + q(n)} \leq n\mu,$$

where $n\mu$ is the rate at which new peers arrive in our model.

Note that this is the total traffic generated per unit time in the network, and thus the average query traffic load per peer $\rho(n)$ is no more than μ. As μ is also the lifetime of a peer, each peer sees no more than 1 query packet for this file while it is in the system. This is independent of $p(n)$ and $q(n)$, as long as we have that $p(n)+q(n) \geq 1/n$. This may be quite surprising at first, but the above derivations provide some intuition why it is true.

Similarly, for a file such that

$$p(n) + q(n) < \frac{1}{n},$$

the expected time to hit a positive or negative peer is equal to

$$\frac{1}{p(n)+q(n)} > \text{TTL}(n).$$

and the TTL will expire before a positive or negative peer is found. As a result, the total query traffic per unit time that is generated by searches for this file is equal to

$$n\mu p(n)\text{TTL}(n) = n^2\mu p(n),$$

and the query traffic per unit time at a peer $\rho(n)$ is equal to

$$\rho(n) = \mu p(n)n < \mu,$$

as we assumed that the request and publishing probabilities are such that $p(n)+q(n) < \frac{1}{n}$. Again, the query traffic load will be bounded and a peer sees no more than one query packet for this file while it is in the system.

The above argument is a simplification, as a random walk can go from a given peer only to one of its neighbors. To address this, our proof of Theorem 3 takes also into account the topology of the overlay graph. In particular, using bounds from Aldous and Fill [11], we relate the relaxation time of a graph in $\mathcal{H}_{n,d}$ to the number of query messages the random walk generates. We then bound the relaxation time of a graph uniformly sampled from $\mathcal{H}_{n,d}$ using the analysis of Friedman [29]. This allows us to fully describe the message cost per query and determine the average traffic load per peer.

B. Reliability

The main challenge in proving that the proposed mechanism is reliable is to show that one can indeed treat the "absence of evidence" as "evidence of absence". Or in other words, one has to show that, if $q(n) = \omega(1/n)$ and the file is published sufficiently often, then false negatives (*i.e.*, queries that failed even though the file is actually in the system) are very rare, and searches for such a file succeed with high probability. The following analogy provides some insight into why this so.

Consider a bin consisting of n balls that are either blue, red, or white. At times $t = 1, 2, 3...$, one ball is removed from the bin by either Alice, Bob, or Charlie, and is immediately replaced as follows. Let $p, q \in (0, 1]$ be such that $p + q \leq 1$. Then, with probability q, Alice replaces the removed ball with a blue ball, and with probability $1-p-q$ Charlie replaces it with a white ball. Otherwise, Bob replaces the removed ball with either a blue or red ball. The probability that Bob puts in a red ball is equal to the ratio of the number of red balls to the total number of red and blue balls in the bin.

Let's now just focus on the ratio of blue to red balls in the bin. Clearly, Charlie does not affect this ratio as she only puts in white balls. Furthermore, Bob also does (on average) not change the ratio of red to blue balls that are currently in the bin. However, Alice always puts in a blue ball, and hence changes the ratio between red and blue balls always in the favor of blue balls. Because of this, if the process goes on forever there will eventually only be white and blue balls in the bin, but no red balls.

We can map the situation to our search mechanism by letting positive peers be blue balls, negative peers be red balls, and null peers be white balls; and let $q(n)$ and $p(n)$ be the publishing and request probabilities of a given file. With probability $q(n)$, a new peer that enters the system will publish the file and become a positive peer. Hence, such an event corresponds to the action of Alice in the above analogy, and puts a blue ball into the bin.

With probability $p(n)$, a new peer will request the file and become a positive or negative depending the outcome of the search. In our technical report [33], we show that if $q(n) = \omega(1/n)$ then searches terminate by either hitting a positive or negative peer, with high probability. Hence, such an event corresponds to the action of Bob, and puts a blue or red ball in the bin. Moreover, a ball is chosen according to the current ratio of positive and negative peers (red and blue balls) in the system. Finally, with probability $1–p(n)–q(n)$ a new peer will not request the file and become a null peer, which corresponds to the action of Charlie.

Using the above analogy, if a sufficiently large number of peers bring the file into the system (*i.e.* if $q(n) = \omega(1/n)$) then, eventually, there will only be positive (blue balls) and null (white balls) peers in the system, and all negative peers (red balls) will die out. Furthermore, as for $q(n) = \omega(1/n)$ almost all searches terminate by either hitting a positive or a negative peer, searches must terminate by hitting a positive peer (blue ball) and hence be successful.

Formalizing the above argument requires additional effort to address the fact that most searches find a non-null peer asymptotically (as n tends to infinity). To do so, in our proof of Theorem 4, we use a mean field analysis, based on the work of Benaïm and Le Boudec [34]. In particular, we show that the stochastic process describing the number of positive peers in the system, over the total number of non-null peers, can be approximated by a deterministic process as n tends to infinity, provided that $q(n) = \omega(1/n)$.

VI. Analysis

In this section, we formally state our main results. We first present results obtained for the traditional random walk search mechanism with a TTL [3]. We then present the analysis of the random walk mechanism that uses evidence of absence, which was proposed in Section IV-C.

Due to space constraints, we state our theorems without proofs; all proofs can be found in our technical report [33].

A. Random Walk with TTL.

Consider the traditional random walk search mechanism that uses a TTL(n) value that depends on the system size n. Peers do not share information about failed queries —a peer that does not locate a copy simply becomes null. We then have the following result.

Theorem 1. *A random walk search with TTL has bounded average load per peer $\rho(n) = O(1)$ for all possible functions $p(n)$ and $q(n)$ only if*

$$\textit{TTL}(n) = O(1).$$

This result states that in order to bound the average load per peer for the traditional random walk search mechanism, the TTL value has to be of the order $O(1)$. The intuition behind this is that one needs $\text{TTL}(n) = O(1)$ in order to ensure that queries for unavailable files only generate a bounded query traffic load at each peer (as the system grows).

However, queries under such a TTL value will not reach all peers in the network. As a result, such queries cannot succeed unless a large number of peers publish the file.

Theorem 2. *For a random walk search with TTL$(n) = O(1)$, queries are guaranteed to be successful,* i.e.*, we have*

$$\lim_{n\to\infty} \gamma(n) = 1$$

for all possible values of $p(n)$, only if

$$q(n) = \Omega(1).$$

The restriction that $q(n) = \Omega(1)$ implies that the expected number of peers that publish the file has to grow linearly with the system size n, in order to guarantee that queries succeed. This is a strong assumption. For example, suppose that $q(n) = 1/n^{0.1}$. Then, a large number of peers bring the file into the system: the expected number of peers that publish the file *grows as* $n^{0.9}$, almost linearly in n. However, even though so many peers have the file, the traditional random walk search with constant TTL cannot locate it reliably.

The above two theorems state that the traditional random walk search mechanism cannot be both scalable and reliable, under the definitions we gave in Section IV-D. To achieve a bounded query traffic load per peer, the TTL value has to be so low that most queries fail, even for files that are widely available (see also our discussion in Section I). We illustrate this result in Section VII using a numerical case study.

B. Random Walk with Evidence of Absence.

In this section, we present our main result, which states that our proposed mechanism, defined in Section IV-C, is both scalable and reliable.

Recall that we assume that the overlay graph of the p2p network is a d-regular graph. The next theorem states that, if $d \geq 12$, the proposed search mechanism is scalable.

Theorem 3. *If the overlay graph has a degree $d \geq 12$, then the random walk search mechanism with evidence of absence is scalable.* I.e.*, $\rho(n) = O(1)$, for all functions $p(n)$ and $q(n)$.*

In other words, the average load per peer is bounded, irrespectively of how often the file is requested or published. In addition, the search mechanism is reliable:

Theorem 4. *If the overlay graph has a degree $d \geq 12$, then the random walk search mechanism with evidence of absence is reliable.* I.e.*, if the file is brought into the system sufficiently often, such that*

$$q(n) = \omega\left(\frac{1}{n}\right)$$

then the file will be located reliably, i.e.*,*

$$\lim_{n\to\infty} \gamma(n) = 1$$

for all $p(n)$.

Note that the restriction on $q(n)$ is much weaker than the restriction on $q(n)$ needed in Theorem 2 for the traditional random walk with a TTL to be reliable.

VII. Numerical Study

In this section, we present a numerical study to illustrate the theorems of the previous section. To do that, we relax the modelling assumptions we made in Sections III and IV as follows: (a) we no longer assume that the system size is fixed, (b) we no longer assume that the overlay topologies are independent between searches, and (c) we allow the overlay network to change even during the propagation of queries.

A. Simulation Setup

In our simulations, new peers arrive according to a Poisson process of rate λ. The lifetime of each peer is exponentially distributed with mean $1/\mu = 20$min while the time to transmit a query is exponentially distributed with mean $\delta = 20$msec. We repeated our simulations for different arrival rates λ, between 10,000μ and 500,000μ. As a result, the average system size $n = \lambda/\mu$ is scaled in each experiment from ten thousand to half a million peers.

To connect peers during arrivals and departures, we implemented the connection protocol defined by Law and Siu [30]. The graphs constructed by the Law and Siu connection protocol are precisely the $\mathcal{H}_{n,d}$ graphs; we use $d = 16$ in our simulations. In particular, at any point in time, the overlay network consists of $d/2$ cycles. Every peer in the graph has degree d: for each cycle k, $k = 1, \ldots, d$, peer i is connected to two other peers, its predecessor $pred_k(i)$ and its successor $succ_k(i)$. When a new peer i' enters the system, it joins each of the $d/2$ cycles at a random position as follows: for each cycle k, $k = 1, \ldots, d/2$ peer i' picks a node j at random and becomes its successor on the cycle while also becoming $succ_k(j)$'s predecessor. Departures are handled similarly: when a node i leaves, its $d/2$ predecessors reconnect to the respective $d/2$ successors of i, maintaining thus both a constant degree and the d cycles property in the graph. We note that the protocol is fully distributed, and that peers need only keep track of their immediate neighbors.

We present results for two different systems: a system in which a traditional random walk with TTL is used, and one in which the walk uses evidence of absence.

B. Random Walk with TTL

Scalability: When a traditional random walk is used, queries for unavailable files can generate an unbounded traffic load. Fig. 1(a) shows the traffic load with and without evidence of absence for $p(n) = 0.3$, $q(n) = e^{-n}$ and $\text{TTL}(n) = 0.01n$. In both simulations, all queries failed. In the simple random walk the traffic load grows linearly, reaching

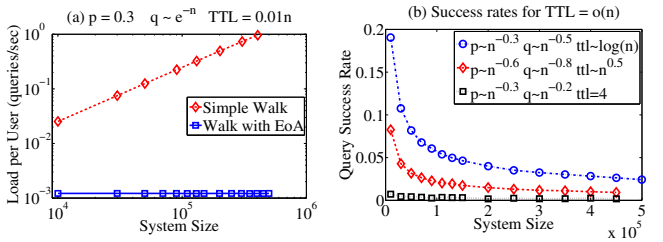

Fig. 1. Traffic load and success rate of the random walk with TTL. In (a), the traffic load generated for a file that is brought rarely in the system grows linearly in n, while using evidence of absence reduces it to a constant. In (b), the TTL(n) grows slower than linear; as a result, even when the file is brought reliably in the system, most queries for it fail.

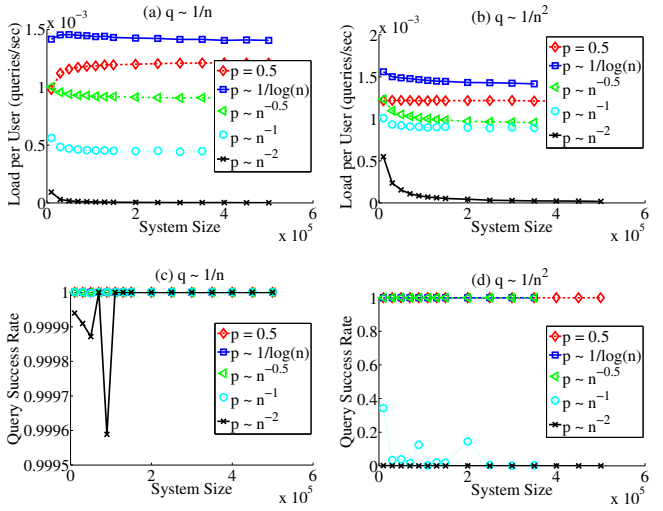

Fig. 2. Traffic loads and success rates for files brought in the system with publishing probabilities $q(n) = 1000/n$ and $q(n) = 1000^2/n^2$. Queries for the files brought in the system with probability $q(n) = 1000/n$ almost always succeed, which is not true for $q(n) = 1000^2/n^2$. In all cases however, the traffic load per peer remains bounded as the system size grows.

0.9609sec^{-1} (57.6 queries per minute) when the system size is 400,000 peers. In contrast, the traffic load when evidence of absence used is bounded: for all sizes, it varies between $1.217\text{E}{-}3\text{sec}^{-1}$ and $1.218\text{E}{-}3\text{sec}^{-1}$ (about 4.38 queries per hour). This happens precisely because negative peers stop unsuccessful queries quickly.

Reliability: Decreasing the TTL (*e.g.*, to $\sqrt{n}$) improves the traffic load incurred by the simple random walk, but does not make it constant in n —such simulations can be found in our technical report [33]. Most importantly, choosing a smaller TTL makes the system unreliable. We illustrate this in Fig. 1(b), where query success rates for the simple random walk with TTL$(n) = 0.1\sqrt{n}$, TTL$(n) = \log(n)$ and TTL$(n) = 4$ are plotted. In all three experiments, $q(n) = \omega\left(\frac{1}{n}\right)$. We see that the query success rate goes to zero even if an increasing number of peers publishes the file.

C. Random Walk with Evidence of Absence

Scalability: In Fig. 2(a) and 2(b), we plot the average traffic load per peer for two different publishing probabilities, $q(n) = 1000/n$ and $q(n) = 1000^2/n^2$, respectively. In both cases, we set the time-to-live to TTL$(n) = 0.01n$, *i.e.*, linear in n. For $q(n) = 1000/n$, the expected number of peers bringing the file in the system are 1000. This number does not grow with n and, therefore, the experiments in Fig. 2(a) are "border line" in terms availability of the file. Note that a larger $q(n)$ (such as, *e.g.*, $q(n) = 1/\sqrt{n}$), can only improve the system's performance, by reducing the traffic load and increasing the query success rate. In the case where $q(n) = 1000^2/n^2$, the file is not brought reliably in the system, as the expected number of peers that publish it decreases with n.

In each graph, we plot the traffic loads with the following request probabilities: $p(n) = 0.3$, $p(n) = \log(1000)/\log(n)$, $p(n) = \sqrt{1000}/\sqrt{n}$, $p(n) = 1000/(n)$ and $p(n) = 1000^2/n^2$. In Fig. 2(a), in all cases except the last, the average traffic load per peer is bounded; for $p(n) = 1000^2/n^2$, we observe a decreasing traffic load. We observe an almost identical behavior in Fig. 2(b): the only difference is that, for $p(n) = \Theta(1/n)$, the average traffic load per peer is 1 query every 1000 seconds —as opposed to 1 query every 2000 seconds, in Fig. 2(a).

Over all, the traffic load generated in all experiments does not grow with the system size, both when the file is reliably brought into the system and when it is not.

Reliability: In Fig. 2(c) and (d) we show the query success rate of the same experiments. When the file is brought reliably in the system (*i.e.*, for $q(n) = 1000/n$), the query success rate was virtually 1 for all $p(n)$ and for all n: as seen in Fig. 2(c), almost all queries succeeded (*i.e.*, located positive peers). In Fig 2(c), as $q(n) = o(1/n)$, we are not guaranteed to find the file by Theorem 4. However, we observe that for high request probabilities the file can still be located reliably; this is not true for $p(n) = 1000/n$ and $p(n) = 1000^2/n^2$, as almost all queries fail in this case.

System Dynamics: Because our focus is on the metrics $\rho(n)$ and $\gamma(n)$, both our theoretical and our numerical analysis describe the steady state behavior of the system. Understanding the system's transient behavior (*i.e.*, its behavior until it reaches the steady state) is not within the scope of this work. Nonetheless, the transient behavior can be important as, *e.g.*, it can help determine the system's response to abrupt phenomena, like "flash crowds". For this reason, we briefly describe the evolution of a transient system below.

Figures 3(a) and 3(b) show how the fractions of positive and negative peers evolve during a simulation. In both cases, $p = 0.8$, $q = 0.1$, TTL $= 100$ and $n = 10,000$. In Fig. 3(a), the simulation starts with all peers being null. Initially, negative peers grow faster than positive peers (as most queries fail); however, within 20min the positive peers have surpassed the negative peers, which peak at 33min and then start to decay. In Fig. 3(b), the same experiment is repeated starting from a system where all peers are negative. Again, the effect of the initial state eventually dies out: although negative peers initially outnumber the positive peers, within 2 hours the positive peers prevail. In both simulations, the positive peer population converges to 90%, *i.e.*, to $p + q$, indicating that, eventually, most queries succeed and every peer that requests the file becomes a positive peer.

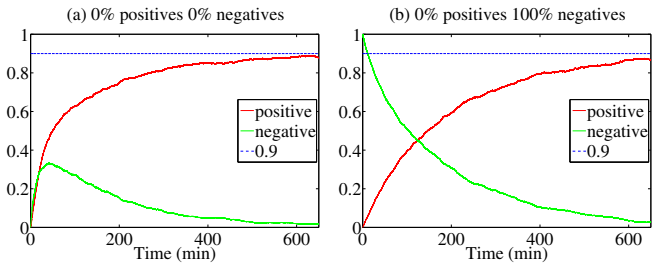

Fig. 3. Simulation traces of the fractions of positive and negative peers. In (a), all peers are initially null, while in (b) all peers are initially negative. In both cases, positive peers eventually prevail and negative peers vanish.

VIII. File Distribution

Our results describe the query traffic load per peer generated by queries issued for a single file. The aggregate traffic load (over all files) incurred at a peer will depend on number of files $M(n)$ that can be requested, as well as on the distribution of the request and publishing probabilities $p_j(n)$ and $q_j(n)$ over different files j. An immediate implication of Theorem 3 is that the aggregate traffic load is bounded if the number of files served by the system is no more than a constant, *i.e.*, if $M(n) = O(1)$. We note however that this condition is sufficient but not necessary: the following lemma implies that the traffic load can be bounded even if the number of files is not.

Lemma 1. *The aggregate traffic load at a peer is bounded if there exists a constant B such that* $\lim_{n\to\infty} \sum_{i=1}^{M(n)} \frac{p_i(n)}{p_i(n)+q_i(n)} \leq B$.

IX. Conclusions and Future Work

Our results show that using the "absence of evidence" as "evidence of absence" can make an unstructured p2p system scalable, by limiting the traffic load generated by queries for files that are not in the system. Most importantly, the above mechanism is also reliable, in the sense that queries for files that are brought into the system sufficiently often are guaranteed to succeed.

There are several interesting extensions of the above work. One is investigating our scheme under different query propagation mechanisms, as well as under proactive replication. The latter approach includes, *e.g.*, proactively changing negative peers to positive, thus increasing the availability of a file, at an additional traffic cost. Understanding the transient behavior of such a mechanism would also be interesting. This is because, *e.g.*, such proactive replication could expedite the convergence of the system to the steady state in the face of "flash crowds".

References

[1] J. Ritter, "Why gnutella can't scale. no, really." 2001, http://www.darkridge.com/ jpr5/doc/gnutella.html.

[2] J. Risson and T. Moors, "Survey of research towards robust peer-to-peer networks: Search methods," *Computer Networks*, vol. 50, no. 17, pp. 3485–521, Dec. 2006.

[3] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS*, 2002.

[4] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *ICDCS*, 2002.

[5] S. Tewari and L. Kleinrock, "Proportional replication in peer-to-peer networks," in *INFOCOM*, 2006.

[6] W. Acosta and S. Chandra, "Understanding the practical limits of the gnutella p2p system: An analysis of query terms and object name distributions," in *MMCN*, 2008.

[7] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein, "The case for a hybrid p2p search infrastructure," in *IPTPS*, 2004.

[8] M. A. Zaharia, A. Chandel, S. Saroiu, and S. Keshav, "Finding content in file-sharing networks when you can't even spell," in *IPTPS*, 2007.

[9] C. P. Fry and M. K. Reiter, "Really truly trackerless bittorrent," Carnegie Mellon University, Tech. Rep. CMU-CS-06-148, 2008.

[10] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *INFOCOM*, 2004.

[11] D. Aldous and J. Fill, "Reversible Markov Chains and Random Walks on Graphs," monograph in preparation.

[12] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM*, 2003.

[13] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchman, "BubbleStorm: Resilient, probabilistic and exhaustive peer-to-peer search," in *SIGCOMM*, 2007.

[14] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 177–190, 2002.

[15] R. A. Ferreira, M. K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan, "Search with probabilistic guarantees in unstructured peer-to-peer networks," in *P2P*, 2005.

[16] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. A. Marsh, "Efficient lookup on unstructured topologies," in *PODC*, 2005.

[17] K. Puttaswamy, A. Sala, and B. Zhao, "Searching for rare objects using index replication," in *INFOCOM*, 2008.

[18] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica, "Enhancing p2p file-sharing with an internet-scale query processor," in *VLDB*, 2004.

[19] M. Castro, M. Costa, and A. Rowstron, "Peer-to-peer overlays: structured, unstructured or both?" Microsoft Research, Tech. Rep. MSR-TR-2004-73, 2004.

[20] M. Zaharia and S. Keshav, "Gossip-based search selection in hybrid peer-to-peer networks," *J. Concurrency and Computation: Practice and Experience*, vol. 20, no. 2, Feb. 2008.

[21] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern p2p file-sharing systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, 2008.

[22] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing*, vol. 6, no. 1, 2002.

[23] D. G. Deschenes, S. D. Weber, and B. D. Davison, "Crawling gnutella: Lessons learned," Lehigh University, Tech. Rep. LU-CSE-04-005, 2004.

[24] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of napster and gnutella hosts," *Multimedia Systems Journal*, vol. 8, no. 5, 2002.

[25] D. Stutzbach and R. Rejaie, "Towards a better understanding of churn in peer-to-peer networks," Univ. of Oregon, Tech. Rep., 2004.

[26] D. Stutzbach, S. Zhao, and R. Rejaie, "Characterizing files in the modern gnutella network," *Multimedia Systems*, vol. 13, no. 1, 2007.

[27] W. Acosta and S. Chandra, "On the need for query-centric unstructured peer-to-peer overlays," in *HotP2P*, 2008.

[28] N. Wormald, "Models of random regular graphs," in *Surveys in Combinatorics*, 1999.

[29] J. Friedman, "A proof of Alon's second eigenvalue conjecture," in *STOC '03*. New York, NY, USA: ACM Press, 2003, pp. 720–724.

[30] C. Law and K.-Y. Siu, "Distributed construction of random expander networks," in *INFOCOM*, 2003.

[31] S. Ioannidis and P. Marbach, "On the design of hybrid peer-to-peer systems," in *SIGMETRICS*, 2008.

[32] A. Ganesh, A.-M. Kermarrec, E. L. Merrer, and L. Massoulié, "Peer counting and sampling in overlay networks based on random walks," *Journal of Distributed Computing*, vol. 20, no. 4, 2007.

[33] S. Ioannidis and P. Marbach, "Absence of evidence as evidence of absence: A simple mechanism for p2p search," Univ. of Toronto, Tech. Rep. CNRL-08-001, 2008.

[34] M. Benaïm and J.-Y. Le Boudec, "A class of mean field interaction models for computer and communication systems," EPFL, Tech. Rep. LCA-REPORT-2008-010, 2008.