

Novel Approaches to Efficient Flooding Search in Peer-to-Peer Networks

Shudong Jin and Hongbo Jiang *

Abstract

Efficient search algorithms are crucial to the success of unstructured and hybrid peer-to-peer networks. Performance requirements on search algorithms include low search traffic, low search latency, and determinism in returning the searched items. However, existing search algorithms fail to meet these goals. We propose, analyze, and evaluate two novel flooding search algorithms. The first algorithm conducts on-the-fly estimation of the popularity of the searched item, and uses such knowledge to guide a peer's search process. It requires the minimum search cost and very low latency, and albeit its non-determinism, often returns the desired number of results. The second algorithm, Hurricane flooding, exponentially expands the search horizon of the source of a search in a spiral pattern. Hurricane flooding is deterministic, requires search cost arbitrarily close to a lower bound, and returns the results in logarithmic time. We analyze and optimize our proposed algorithms, and evaluate them using various network models, including a real Gnutella network topology.

1 Introduction

Peer-to-peer systems are one of the most vital Internet applications and a major portion of Internet traffic is attributed to them [15]. Broadly, there are three different architectures for peer-to-peer networks: centralized, decentralized but structured, and decentralized and unstructured. Centralized architecture scales poorly and does not tolerate failures. Decentralized but structured networks such as Chord [17] need to maintain the overlay topology. In decentralized and unstructured networks, there is neither a central directory server nor any control over the network topology. Peers join the network by connecting to existing peers in a very loose and random fashion. Such an architecture overcomes the problems in other peer-to-peer systems. Gnutella and KaZaA are two successful and widely used

*The authors are with Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106. Corresponding author: Shudong Jin (E-mail: jins@case.edu; Telephone: 1-216-368-5877; Fax: 1-216-368-1534)

systems. However, these networks are infamous for their extremely inefficient searching mechanisms. Since there is neither a central directory server nor distributed indexing service, the best a peer can do is a blind search. The typical flooding search method generates high loads on the peers and are extremely not scalable. One approach to making unstructured peer-to-peer networks more scalable is via modifications to the architecture, or use of hybrid architecture [21, 12]. Both KaZaA and Gnutella currently have a more hierarchical architecture. A smaller number of upper level peers are responsible for propagating and answering queries initiated by more lower level peers. Another approach is via the design of more efficient search mechanisms. For example, [6] exploited the heterogeneity of peers and proposed to use a biased random walk based protocol to direct queries to high-capacity peers.

The focus of this work is to minimize the cost of search in peer-to-peer networks and to make them more scalable. The approach is via designing two novel flooding algorithms. Our first novel search algorithm [11] estimates the popularity of the searched item while searching for multiple results (or multiple copies of the searched item). This on-the-fly estimation of the popularity helps the source of the search to effectively locate the results and bound the search traffic. To that end we exploit Gnutella's dynamic querying like flooding algorithm. We propose an enhanced algorithm, which returns the desired number of results at minimum search traffic and low search latency. Our second novel algorithm, Hurricane flooding, is deterministic and requires provably minimum cost. With Hurricane flooding, the source of a search cautiously but exponentially expands its search horizon in a spiral pattern. We show that Hurricane flooding reduces the search cost to arbitrarily close to a lower bound for any search algorithms, and that it has a bounded latency which is a logarithmic function of the location of the target.

The remainder of this paper is organized as follows. We first describe the search problem and the performance requirements in Section 2, and then point out the weaknesses of previous search algorithms in Section 3. Section 4 describes our work on exploiting dynamic querying like flooding algorithms. Section 5 describes our Hurricane flooding algorithm. Section 6 presents an evaluation of the proposed algorithms and Section 7 concludes the paper.

2 Preliminaries

Consider the problem of searching for a copy or multiple copies of an item in a large, connected unstructured peer-to-peer network. The item is uniformly replicated in the network, i.e., each peer has the same probability of storing a copy of this item. Although this uniform replication assumption is not necessary for the correctness of any

algorithms considered in this paper, it allows us to focus on the more important aspects. Let us consider the search problem where the search *horizon* is controlled by the source. The horizon is defined as the scope of the search. Controlled search is desirable since otherwise every search involves all peers. It is often there are many more copies of an item in the network, but the search is only for one or a few of them.

The source of a search tries to find the *target*. For instance, the search can be for a single copy of an item. The target is the first peer discovered to have a copy. If the search is for multiple copies of an item, then the target is defined as the point when this desired number of copies are discovered. To locate the target, the source sends query packets to other peers. If this attempt is unsuccessful, the source will expand its search horizon. The search strategy determines how to conduct the search to locate the target. Different search strategies are classified into two broad classes: *deterministic* strategies and *non-deterministic* strategies with respect to the outcome of the search. With deterministic strategies, it is guaranteed that the source will discover a copy or multiple copies of the searched item as desired, provided that there are at least that many copies in the network. With non-deterministic strategies, it is possible that the source gives up after one or more unsuccessful attempts.

The *total search cost* of a search is defined as the total number of query packets sent or received by any peers. We ignore packets that are dropped in the network. This total cost includes both necessary cost and unnecessary cost. One unnecessary cost is due to the *repeated query packets* in a single search process. In many search strategies, when an earlier attempt to discover the target is unsuccessful, the source will try to send more query packets, to perhaps many peers who had received the query packets. These repeated query packets are distinguished from those duplicate query packets, which are multiple packets received by the same peer due to a single attempt of the source. Another possible cause of the unnecessary cost is the *overshooting* when the source is probing the search space. Let us define the *latency* of a search as the time needed to locate the target. While it is also a measure of the quality of the search strategy, it is not the primary one, since in unstructured peer-to-peer networks, the dominant factor limiting the scalability is the search cost.

2.1 A Lower Bound on Search Cost

Consider an arbitrarily large network where the replication ratio of an item is a constant p . A question is asked: on average at least how many peers will be contacted before a copy of the item is discovered? Consider a search process

with an oracle. The oracle tells which peers have been contacted already, such that the search process avoids sending more query packets to those peers. However, in a directory-less system, even the oracle cannot tell which peers have the copies of the item before the peers are contacted. On average this search process involves $1/p$ peers. Moreover, if the search is for k copies of the item, the expected number of queried peers is k/p . While this result hardly makes sense in practice, it represents an often unreachable lower bound on the cost of a search for any search strategies. This lower bound provides a fair means to understand if an algorithm is close to the optimal.

3 Related Work

3.1 The Expanding Ring Algorithm

One of the search strategies is TTL-based controlled flooding, often known as the expanding ring algorithm [13, 20]. The source of a search starts by choosing an initial time-to-live (TTL) value, a positive integer. The source then sends query packets towards its immediate neighbors with the TTL value. It decrements the TTL value by one and forwards the query packet to its other neighbors. This process continues until the item is found or the TTL field becomes 0. If no query packet is replied within a timeout interval, the source assumes this attempt has failed. It may start a new round of flooding with a larger TTL value. There could be multiple rounds of flooding.

Although the expanding ring algorithm is deterministic and has bounded latency, it is inefficient due to its high search cost. In addition to the overhead due to duplicate query packets within one round of flooding, there is also overhead due to repeated query packets caused by an inappropriate choice of TTL. To pick the right TTL value is not easy either, as the source does not have knowledge on the popularity of the searched item. Moreover, the expanding ring algorithm may also cause severe overshooting. In a high-degree network, increasing TTL by one greatly expands the search horizon and causes excessive traffic. Recent theoretical work [2, 4, 5] has provided results on the performance of the expanding ring algorithm. Both the worst case cost and average case cost were considered. Various flooding strategies (with different approaches to setting TTL) have been proposed, including those with a worst case cost ratio of 4 and with an average case cost ratio below 3. In the California Split algorithm, the source will double its search horizon on every unsuccessful attempt. This algorithm has a worst case competitive ratio of 4, which is the lowest achieved by any search strategy with fixed TTL sequence. Subsequent studies [4, 5] derived more sophisticated randomized algorithms to achieve worse case and average case cost ratios slightly below 3.

3.2 Random Walk Based Methods

The second class of search strategies utilizes the random walk techniques. Several studies [13, 7, 3, 16, 9, 10] have proposed algorithms to use random walks for efficient search, or have investigated their properties in unstructured networks. When a peer receives a query packet, it forwards the packet to a randomly chosen neighbor at each step until the item is found. The use of a single or few query packets reduces the search cost, at the cost of long user-perceived latency upon eventual success; the use of more query packets cuts the latency, but it increases the total search cost. While random walk based strategies have advantages in the networks with specific properties [9], it is not suitable for several reasons. First, these strategies are non-deterministic. A strategy can always reduce its search cost by not returning any results. Second, they present only a tradeoff between latency and cost. When deterministic is required, the tradeoff between latency and search cost does not look attractive. The lowest search cost (the lower bound) might be approached when there is a single query packet, but at very high latency. On the other hand, the use of multiple isolated query packets requires proper termination rules.

4 Dynamic Querying like Flooding

In peer-to-peer systems, often a search is for multiple results matching the keywords. The flooding search algorithm can determine how to expand its search horizon based on the on-the-fly estimation of the popularity of the searched item. In this section we exploit such techniques, termed as dynamic querying like flooding algorithms.

4.1 Dynamic Querying in Gnutella

In modern Gnutella peer-to-peer network, there are two types of peers: ultrapeers and leaf peers. The ultrapeers manage the search process for their leaves and forward query packets from other ultrapeers. Hereafter, a peer refers to an ultrapeer when the discussion is about Gnutella. When a peer receives a search request, it propagates the query to the network using the dynamic querying algorithm [8]. It works in a more conservative way than the expanding ring algorithm. It seeks to hit the minimum number of peers necessary to obtain the desired number of results.

The source peer first sends a probe query via a few neighbors with a small TTL. The purpose of this probe is to have an initial estimate of the popularity of the searched item. Assume the probe does not return the desired number of results. The source peer will begin an iterative process of dynamically calculating the TTL for the remaining

neighbors. In each iteration, the source peer has the current number of returned results. It estimates the number of peers theoretically queried so far, and the theoretical popularity of the searched item. Then the source peer can estimate how many more peers should be contacted in order to receive the desired number of results. This estimate is divided by the number of remaining neighbors. In this way, the source peer obtains the number of peers to query via the next neighbor, and estimates the minimum TTL to reach that number of peers. The source peer then sends the query via this neighbor using this TTL value, and waits for the results until a timeout. This iterative process continues until all neighbors are used, or the peer obtains the desired number of results. Intuitively, this technique dynamically adjusts the TTL value of outgoing query packets along each additional neighbor to hit only the necessary number of peers to obtain the desired number of results. The benefits in reducing the search cost are two-fold. First, this technique avoids sending query packets too far (towards too many peers). Second, it avoids sending query packets repeatedly to the same subset of peers.

4.2 Enhanced Dynamic Querying like Flooding

We find that the design of Gnutella's dynamic querying algorithm is flawed. The iterative process can introduce long delay. When there are many remaining neighbors, a query packet is propagated to just a small fraction of the required number of peers. Unlikely this iteration will return the desired number of results. This method is doomed to have a high latency. We propose the following enhanced algorithm.

The enhanced algorithm is similar to the original dynamic querying algorithm. The source peer begins the search (for N results) by first sending a query towards a few neighbors with a fixed TTL. In succession, the network replies $n \geq 0$ results that can be used to deduce the popularity of the item. After getting an estimate of its popularity, we can subsequently calculate the TTL for the next neighbor. A query packet with this TTL value is propagated via this next neighbor. The new number of results is obtained and iterated to calculate the TTL for another neighbor. This process continues until the desired number of results are obtained or all neighbors are used.

The main difference is the iterative process of the enhanced algorithm. This iterative process is (1) *greedy* — in each iteration, the source peer propagates a query packet to a new neighbor, hoping to find all the required number of results via this neighbor alone, and (2) *conservative* — at the same time, to avoid overshooting, the source peer uses a confidence interval method to provide a safety margin on the estimate of the popularity of the searched item.

Since we use a greedy iterative process to avoid excessive latency, more likely there will be big random overshooting of the search space. Therefore, we need a more conservative estimate of the popularity of the searched item using, a confidence interval method. This is the intuition behind our conservative approach. Next we present details on how to estimate the popularity of the searched item conservatively and how to calculate the TTL value.

Estimating Item Popularity We use the following confidence interval method to estimate the popularity of the item conservatively. First we assume uniform replication of the item. The search process is considered as a sequence of Bernoulli trials. Let H_0 denote the current search horizon (i.e., the sample size). The probability of obtaining i results is given by the binomial distribution

$$\text{Prob}_p(i|H_0) = \binom{i}{H_0} p^i (1-p)^{H_0-i},$$

where p is the replication ratio or the popularity of the item. When the sample size H_0 is large, this probability approaches the Poisson distribution

$$\lim_{H_0 \rightarrow \infty} \text{Prob}_p(i|H_0) = \frac{m^i}{i!} e^{-m},$$

where $m = pH_0$. This distribution has mean m , variance m , and standard deviation \sqrt{m} . Then we use Pearson's confidence interval on Poisson distribution as follows. Given the number of returned results n , we choose a conservative estimate m' of the true mean m by solving:

$$m' - \delta\sqrt{m'} = n,$$

where the constant $\delta > 0$. We obtain

$$m' = n + \frac{\delta^2}{2} + \delta\sqrt{n + \frac{\delta^2}{4}}, \quad (1)$$

which is the upper limit of Pearson's confidence interval. This result says, if there are $n \geq 0$ returned results, then the expected number of returned results is less than m' with a probability determined by the parameter δ .

There is a tradeoff between choosing a larger value for the parameter δ and choosing a smaller value. First choosing a larger value for δ results in more conservative estimate of the mean m . For example, when $\delta = 1$, the confidence level is about 68% and the upper limit probability is about 16%, but if $\delta = 3$, the confidence level is about 99.7%. Therefore, choosing a larger value for δ will less likely cause overshooting. On the other hand, being more

conservative means that it often takes longer time for the source peer to discover the required number of results. In the enhanced algorithm, we choose $\delta = 3$. The value provides a high confidence level. In our evaluation, we will consider other values too.

Calculating TTL value Since we have an estimate of the popularity m'/H_0 , we can compute the search horizon H for the next neighbor, which should be equal to $H_0(N - n)/m'$. Assume the next neighbor has degree d which can be known, and the average degree of the network is D which can be estimated. We should pick the TTL value such that H equals the horizon within TTL hops from this neighbor:

$$H = (d - 1) \sum_{i=0}^{TTL-1} (D - 1)^i \approx \frac{(d - 1)(D - 1)^{TTL}}{D - 2}.$$

This is equivalently to

$$TTL \approx \log_{(D-1)} \frac{H(D-2)}{d-1}. \quad (2)$$

One problem is, the obtained TTL value is often a floating number. We modify peers' forwarding algorithm as follows. When a peer receives a query packet with $TTL \geq 1$, it forwards it (with TTL decreased by 1) to all neighbors as usual. If the TTL value is a fraction, the peer forwards the packet (with TTL set to 0) to a subset of its neighbors. The size of this subset should be equal to $d^{TTL} - 1$, where d is this peer's degree. To understand this, just notice that when $TTL = 0$, the packet should not be forwarded to any neighbors; when $TTL = 1$, the packet is forwarded to all neighbors. Therefore, the peer needs to forward the packet to each of its $d - 1$ neighbor with a probability equal to $(d^{TTL} - 1)/(d - 1)$.

For extremely less popular items, the probe phase may not return any matched results. Consequently, the original Gnutella dynamic querying algorithm fails since it cannot estimate the search horizon and TTL value. On the other hand, our algorithm sets $m' = \delta^2/2$ as a conservative estimate and calculates the TTL accordingly. In our evaluation, we will show results for such unpopular items.

5 Hurricane Flooding

The previous section describes and enhances dynamic querying like flooding algorithms. They have limitations. First, they are useful only when the search is for many results. Second, they are not deterministic. This section describes our Hurricane flooding algorithm, and analyzes and optimizes its performance.

5.1 Design of Hurricane Flooding

Hurricane flooding exploits the ideas of both the expanding ring algorithm and the dynamic querying algorithm. Like the expanding ring algorithm, Hurricane flooding increases the scope of flooding after each round. Like the dynamic querying algorithm, the source floods queries via different neighbors in multiple rounds.

5.1.1 A Brief Description

The source peer divides its neighbors into r groups with approximately the same size. The special case $r = 1$ corresponds to the expanding ring algorithm. In general, the source sends query packets to its neighbors in the first group, starting the first *round* of flooding. These neighbors faithfully broadcast the query packets (but not back to the source). The source also sets a limit on the scope of these broadcasting query packets, e.g., by using a TTL value. This limit is updated by every peer before forwarding the query packet. The first round of flooding may have a very narrow scope. This round of flooding may not return the desired result. Then the source sends query packets to its neighbors in the second group, with a larger limit on the scope of the flooding. This new limit is $b > 1$ times the limit during the previous round. This process repeats until the source obtains the desired result. It is possible that after r rounds of flooding, the source has yet to locate the target. In such cases, the source starts the $(r + 1)$ -th round by sending query packets to its neighbors in the first group again. At this point of time, a new *cycle* of flooding begins. Finally, it is possible the source experiences multiple cycles of flooding before it terminates the process. Figure 1 shows an example of Hurricane flooding with $r = 3$.

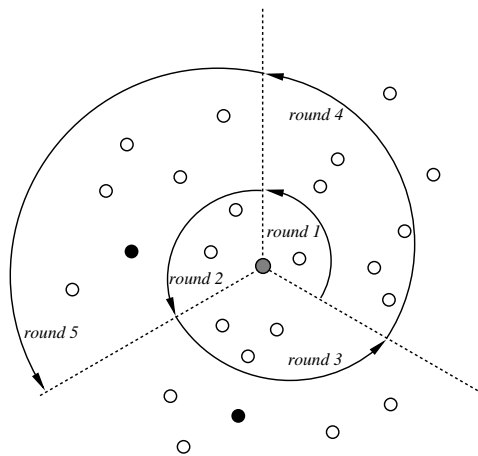


Figure 1: An example of Hurricane flooding, $r = 3$. The source (center) searches for copies in the solid nodes.

Theorem 1 *In any connected networks, Hurricane flooding will discover the target if it exists.*

This result is clear from its design. With Hurricane flooding, the TTL value will be increased by a constant after each unsuccessful round of flooding. Therefore, the target in a connected network will eventually be located.

5.1.2 Forming Rounds of Flooding

One issue is how to divided the neighbors into r groups. Consider two scenarios. First, the number of neighbors is much larger than r . A simple round-robin method can be used to group the neighbors. Further improvement is possible. For instance, each neighbor can have a weight equal to the number of its neighbors less one. Then a standard bin-packing algorithm (such as Best Fit-Decreasing) will group them. This can result in more balanced group sizes. In the second scenario, the number of neighbors is small. We can set r to be the degree of the source. In the extreme case, the source has only one neighbor and Hurricane flooding would degenerate to the expanding ring algorithm. Yet, in this extreme case, there is a solution. The source asks its neighbor to be its proxy to manage the search process; this proxy uses again Hurricane flooding.

5.1.3 Using Fine-grained Expansion

Another issue is how to decide and control the scope of the search in each round. The growth rate $b > 1$ is defined as the ratio between the sizes of the scopes of two consecutive rounds. We use the following fine-grained expansion. The source uses a floating number for the TTL value. After each unsuccessful round, the TTL value is increased by a fraction which is equal to $\log_{(d-1)} b$. When an intermediate peer i receives a query packet with $TTL \geq 1$, it forwards the packet (with TTL decreased by one) to all neighbors. If the TTL value is a fraction smaller than one, the peer forwards the packet (with TTL set to 0) to its neighbors probabilistically. As described in Section 4.2, a peer with degree d forwards the packet to each of its other $(d - 1)$ neighbors with a probability equal to $(d^{TTL} - 1)/(d - 1)$.

5.2 Analysis and Optimization of Hurricane Flooding

5.2.1 Analysis of Hurricane Flooding

Let us first consider a simple tree network model where the source is located at the root of the tree. We analyze three aspects of Hurricane flooding's performance. The first is repeated query packets. Notice that there are no duplicate

query packets in a tree network. A total count on the number of repeated query packets is not a fair measure. To reach a remote target often incurs more repeated query packets than to reach a nearby target. For this reason, we consider the ratio between the number of repeated query packets to the number of unique query packets. The second performance aspect is the overshooting of the search space. Overshooting is measured by the ratio between the number of unnecessary query packets to the number of unique query packets. Combining these two aspects, we obtain the total overhead ratio. The overhead ratio is equal to the search cost ratio less one. The third performance aspect is the latency perceived by the source. The latency is determined by the number of timeouts, i.e., the number of rounds of flooding required to locate the target, as well as the individual timeout intervals. The timeout intervals can be a function of the scope of the search, e.g., if in one round of flooding, the scope is larger, then we may set a larger timeout interval. For simplicity, we use the number of rounds of flooding.

Overhead due to Repeated Query Packets: Repeated query packets occur when a new cycle of flooding starts and the query packets are sent to the same neighbors. The growth rate from one cycle to the next is b^r . Therefore the ratio between the number of the repeated query packets in a round and the total number of the query packets in this round is $\frac{1}{b^r}$. Consider theoretically infinite number of cycles, the overhead ratio due to the repeated query packets is computed as $\sum_{i=1}^{\infty} \frac{1}{b^{i*r}} = \frac{1}{b^r-1}$. This result indicates that, when r is large enough, repeated query packets are rare; when b is not too small, even if r is relatively small, repeated query packets are still not common.

Overhead due to Overshooting Query Packets: Before computing the overhead due to overshooting, let us notice that there are worst case overshooting and average case overshooting. Let us mainly consider the worst case overshooting. It occurs when the current search horizon is just not large enough to reach the target, and hence an additional round of flooding is needed. We do not need an additional cycle of flooding since the target is an abstract term. In this scenario, the entire additional round of flooding is an overshooting. To compute this overshooting, let us first compute the current search horizon. The horizon is equal to the total number of peers reached in the last cycle. Any peers reached before the last cycle are also included in the last cycle. Therefore, the search horizon is $\sum_{i=0}^{r-1} b^i S_0 = \frac{b^r-1}{b-1} S_0$, where S_0 denotes the scope of the search in the first round of the last cycle. Since the search space size of the current overshooting round is $b^r S_0$, the overhead ratio due to overshooting is $\frac{b^r(b-1)}{b^r-1}$. When $b^r \gg 1$, overshooting is mainly determined by b . If we set b to a small number, overshooting is effectively bounded.

On the other hand, the average case overshooting depends on the probability distribution function of the target

location. For instance, if the target is uniformly distributed in the overshooting space, the average case overshooting is half of the worst case overshooting. Such a dependency implies, if the probability distribution function is not known, it is difficult (if not impossible) to obtain the average quantity. Fortunately, we will find that even the worst case cost can be arbitrarily close to the lower bound.

Total search cost ratio: The total search cost includes the necessary number of query packets to find the target location, the overhead due to repeated query packets, and the overhead due to overshooting. Based on the previous analysis, we can calculate the worst case search cost ratio $C_{\text{worst-case}}(b, r) = 1 + \frac{1}{b^r - 1} + \frac{b^r(b-1)}{b^r - 1} = \frac{b^{r+1}}{b^r - 1}$.

Search latency: Finally, to compute the latency perceived by the source, we first let H_0 denote the initial search horizon in the first round of flooding, and let $H_{\text{target}} \gg H_0$ denote the search horizon necessary to discover the target. The question is how many rounds (i.e., latency L) of flooding is required. An observation is the search horizon in the current cycle is always b^r times that in the previous cycle. Hence, $L = \log_b (H_{\text{target}}/H_0)$. Therefore,

Theorem 2 *Under the tree network model, in Hurricane flooding, the search latency is a logarithmic function of the target location.*

5.2.2 Minimum-cost Hurricane Flooding

Our main objective is to minimize the total search cost. This subsection focuses on optimizing Hurricane flooding such that it has the lowest search cost. We have the follows:

Theorem 3 *There is an optimal Hurricane flooding to minimize the search cost in tree networks.*

Just notice from the last subsection that we have the total search cost ratio $C_{\text{worst-case}}(b, r) = \frac{b^{r+1}}{b^r - 1}$. To minimize $C_{\text{worst-case}}(b, r)$, we let $\frac{dC}{db} = 0$ and obtain the optimal point $b^* = (r + 1)^{1/r}$.

Figure 2 shows the search cost ratio as a function of b when r is set to typical values. This figure shows several results. First, we observe that each curve has two distinguishable regions. When b is small, the cost is dominated by repeated query packets. Increasing b causes a sharp decrease in the search cost. When b is large, the cost is mainly due to overshooting. The optimal point is a balance in the middle. Second, when r increases, the search cost decreases. In particular, when r is small, increasing r results in a sharp reduction in the search cost. To display this effect, let us plot in Figure 3 the search cost ratio at the optimal point when r varies. This second figure shows that

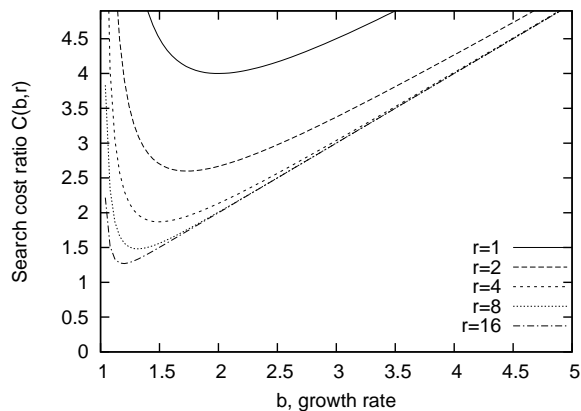


Figure 2: Overhead ratio as a function of the growth rate b when the number of rounds per cycle r is a typical value.

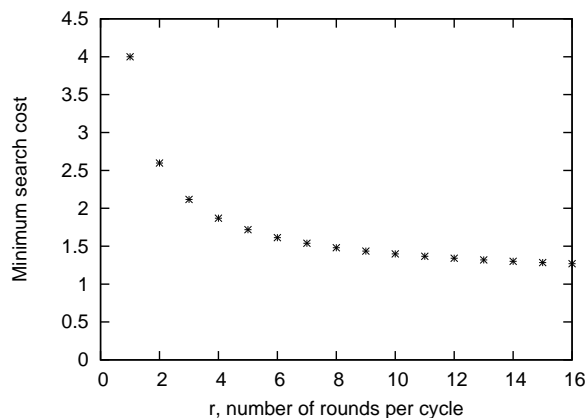


Figure 3: The minimum search cost ratio as a function of r , the number of rounds per cycle.

there is a sweet point approximately between $r = 5$ and $r = 8$. Beyond this point, there is a diminishing return from increasing r . This observation indicates that Hurricane flooding will be effective even in random networks with relatively low average degree.

The special case of $r = 1$ deserves more discussions. In this case Hurricane flooding degenerates to the expanding ring algorithm. The worst case search cost ratio is 4 when $b = 2$. This ratio bound was previously achieved by the California Split search strategy, the best deterministic algorithm known in the literature. With Hurricane flooding, the search cost ratio can be arbitrarily close to 1 if we choose a large number of rounds per cycle.

Finally, although we have minimized the worst case cost, often average case cost is a more meaningful quantity. We can expect more reduction in average case cost compared to Figure 2 and Figure 3. However, the exact expression for the average case cost depends on the probability distribution function of the target location. Therefore, it is

impossible to solve the general problem of minimizing the average case cost. On an encouraging note, Figure 2 shows that when b is small, for example just below 1.5, the worst case search cost is close to optimal for a wide range ($r > 4$). Thus in practice, we can simply set b to be such a small value.¹

5.2.3 Hurricane Flooding and Network Models

In the previous analysis we assumed a tree network model where the source reaches two non-overlapped subsets of peers via any two neighbors. However, real peer-to-peer networks are not trees. It is necessary for us to understand what is the degree of overlapping in more complex and more realistic networks.

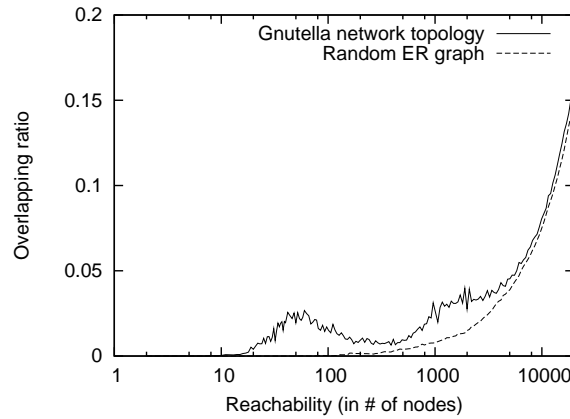


Figure 4: The average overlapping ratio of the search horizons of two neighbors of a source.

We conduct the following experiments. First we obtain two network topologies, one Erdős-Rényi (ER) random graph with about 160K peers and 400K edges, and one real Gnutella network topology with about 160K peers and 1.9M edges (described in Section 6). Then let us randomly pick one source in a network and two of its neighbors. Then starting from each of the two neighbors, a source reaches a number of peers (but not back to the source) using breadth-first search. Finally, let us compute what is the overlapping ratio between these two subsets (which have the same size). This ratio is defined as the number of peers reached via both neighbors divided by the size of each subset. Let us repeat this process for different random sources and compute the average overlapping ratio. Furthermore, let us compute the average ratios when the reachability (i.e., the size of the subsets) increases. The results are plotted

¹To reduce the average case cost, one may set b just slightly larger than the optimal point $(r + 1)^{1/r}$ for the worst case. Since between the two parts of the overhead cost, the overhead due to repeated query packets is fixed, and the overhead due to overshooting is lower than it is in the worse case. Thus the setting of b should be weighed more towards decreasing the first part, i.e. increasing b .

in Figure 4, for both the ER random graph and the Gnutella network topology. With the ER random graph, the overlapping ratio stays very low even when both neighbors reach thousands of peers. With the Gnutella network, the overlapping ratio is a little higher. However, the absolute ratio is not high. For example, when both neighbors reach 1000 peers, only about 30 peers are reached by both. Therefore, we expect Hurricane flooding to be effective the Gnutella network topology.

6 Performance Evaluation

6.1 Evaluation Methodology

6.1.1 Evaluated Algorithms

We evaluate the expanding ring algorithm, the two described dynamic querying like algorithms, and the Hurricane flooding algorithm with different growth rates $b = 1.5, 2.0,$ and $3.0,$ respectively. We do not consider random walk based algorithms since they take a different, non flooding based approach. We are also aware of variants of the expanding ring algorithm and do not attempt to evaluate them via simulation. Like the expanding ring algorithm, those algorithms often require search cost much higher than the lower bound. The dynamic querying like flooding algorithms are implemented with the following details. We have followed the protocol specifications and used the recommended parameter settings in Gnutella. Unless otherwise noted, a peer with degree at least 15 is picked to manage a search process. There is no restriction on the degree of peers who forward queries. They faithfully propagate the queries when TTL is not equal to zero. In the probe phase, the query is propagated down three neighbors with TTL=2. We use the approach described in the same document to estimate theoretical horizon and the average popularity of the searched item. The default maximum TTL value allowed for each neighbor is 4. The calculated TTL value is rounded up or down to an integer value. The timeout interval is set to TTL times 2.4 seconds.

6.1.2 Performance Metrics

We use three metrics to compare the performance of different flooding algorithms. (1) The first is the total number of query packets transmitted by all peers. Often this number is higher when the source peer finds more results. A related metric is the average number of query packets transmitted per result. This quantity is higher than the lower

bound $1/p$ since we use uniform replication with replication p . (2) The second is the total latency of the search process. (3) The third is the number of returned results when we evaluate dynamic querying like flooding. Note in our evaluation, Hurricane flooding is asked to locate only one copy of the searched item. Its relative performance (compared to that of the expanding ring algorithm) does not change when the search is for multiple copies.

6.1.3 Network Topologies

Table 1 summarizes the topologies, including the real Gnutella network topology mentioned in Section 5.2.3. The Gnutella network topology, dated on February 2, 2005, was obtained as a result of an ongoing research project at University of Oregon [18, 19]. This snapshot includes over 160K peers and its average number of neighbors per peer is close to 24. Our evaluation of dynamic querying like flooding is based on this topology.

Table 1: Network topologies used in performance evaluation.

Model	# of peers	Mean degree
Random d -regular	110000	3.0
Erdős-Rényi random graphs	110000	30.0
Power-law networks, $\alpha = 1.25$	109440	4.99
Gnutella network topology	161538	23.5

We also generated other network topologies for the evaluation of Hurricane flooding. The first is the random d -regular graphs. A random d -regular graph is very close to a k -ary tree when $k = d - 1$. The reachability, i.e., the number of peers within h hops, grows as $(d - 1)^h$ roughly when h is small. In particular, since we choose $d = 3$, the expanding ring algorithm mimics the California Split algorithm and is expected to have a low search cost. The second topology model is the Erdős-Rényi random graphs, and the average degree is 30.0. The third topology model is the power-law random graphs [1]. In this model, the probability that a peer has degree larger than d is proportional to $d^{-\alpha}$, where the constant α is the power-law exponent. Several studies [13, 14] discovered that the peer degrees of early Gnutella topology follow power-law distributions or two-segment power-law distributions. In this evaluation, we set the power-law exponent $\alpha = 1.25$. This relatively low exponent means a very high variability of peer degree. This allows us to show the effects of such variability. The results present in this paper are representative, and we can

obtain similar results while using larger values for α in the simulations.

6.2 Performance of Dynamic Querying Like Flooding

We first compare the expanding ring algorithm, the original dynamic querying, and our enhanced algorithm. The Gnutella network topology is used. We set the replication ratio to be 0.01. Since there are over 160K peers in the topology, we uniformly place just over 1600 copies of the searched item in the network. We have considered more skewed replication schemes, and found the relative performance of different algorithms does not change much. Each search is for 50 results. The probe phase returns about 10 results (with variations). We have considered a range of replication ratio. Although the replication ratio affects the absolute search cost and the absolute latency, it does not affect the relative performance of the algorithms. It is worth noting that when the replication ratio is p and the copies are uniformly placed, the minimum average search cost is $1/p$ packets per result (see the lower bound on search cost in Section 2.1). We find all algorithms have the per-result cost slightly higher than this minimum value. This observation is consistent when the replication ratio is within a wide range.

Figure 5–7 show the performance of the expanding ring algorithm, the original dynamic querying, and our enhanced algorithm, respectively. Each has 100 simulation runs. We can observe the follows. First, the expanding ring algorithm often has big overshooting and returns much more results than necessary. On the contrary, dynamic querying like algorithms often return just more than the desired number of results. Consequently, they reduce the search cost by several times. We observe that for both algorithms, the average number of packets required is around 6000 and the enhanced algorithm does slightly better. Second, the Gnutella dynamic querying algorithm obtains lower search cost at the price of high latency. With expanding ring, a source peer often floods the query packets three times (with TTL =1,2,3) and rarely floods the query packets with TTL=4. The average latency is below 20 seconds. With dynamic querying, the average latency is close to 70 seconds. It is often above 100 seconds, and occasionally more than 3 minutes. With our enhanced algorithm, the average latency is very close to that of the expanding ring algorithm.

When we set a smaller $\delta = 1$ for our algorithm, Figure 8 shows that both the number of returned results and the search cost become more variable. The latency becomes lower. These observations are consistent with our discussions in Section 4.2. When we set a lower replication ration $p = 0.002$, i.e., there are about 320 copies

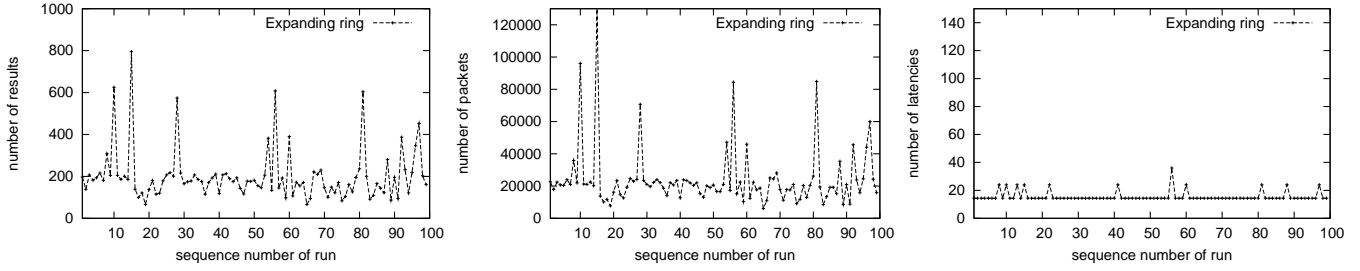


Figure 5: The performance of the expanding ring algorithm.

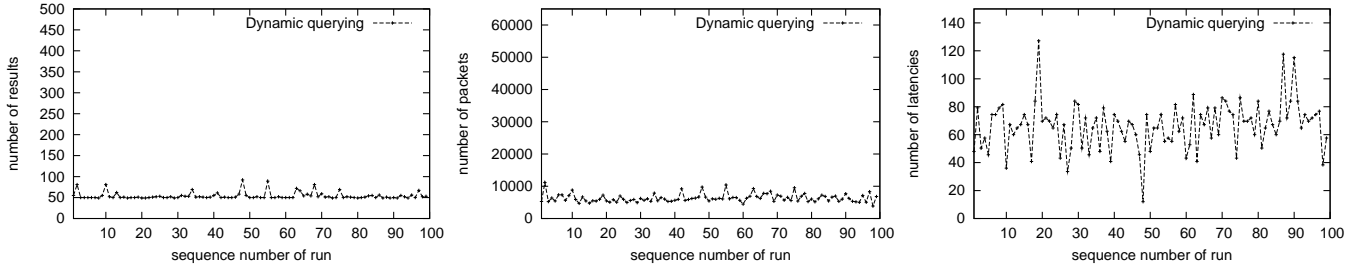


Figure 6: The performance of the dynamic querying algorithm.

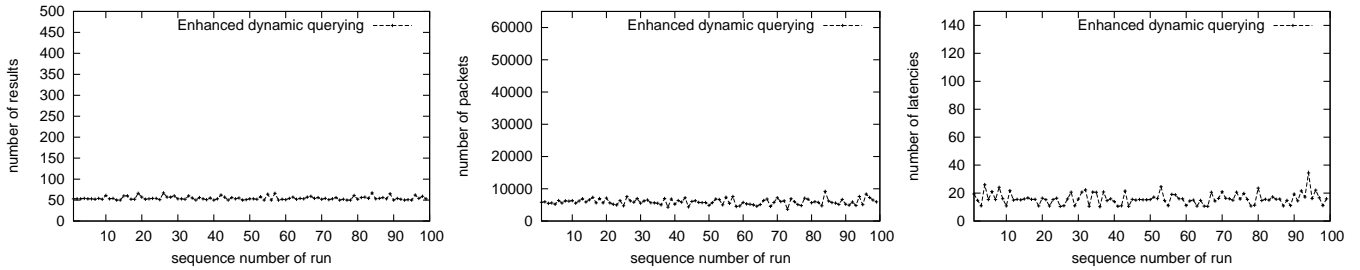


Figure 7: The performance of our enhanced algorithm.

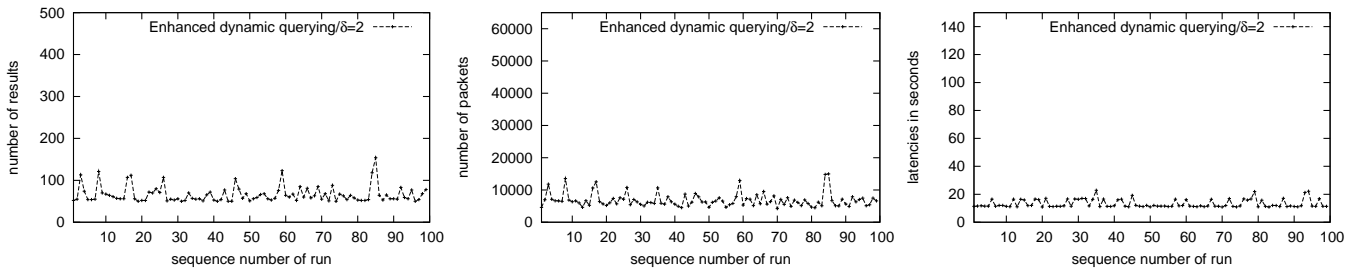


Figure 8: The performance of our enhanced algorithm when $\delta = 1$.

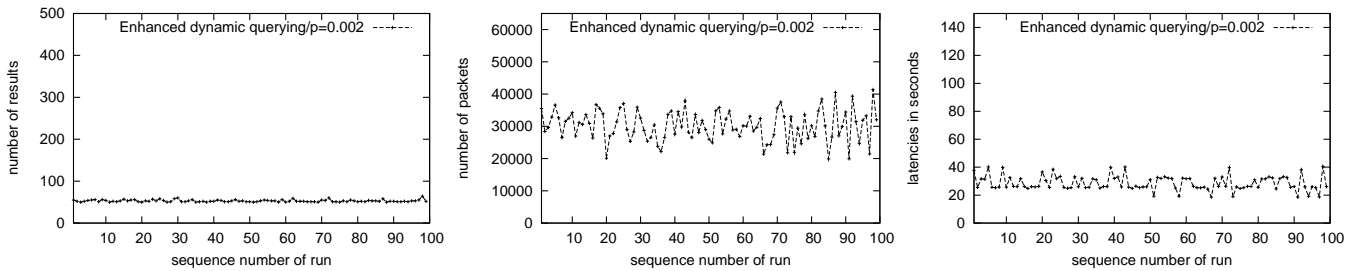


Figure 9: The performance of our enhanced algorithm when replication ratio $p = 0.002$.

in the entire network, Figure 9 shows that (1) the search cost is about five times that when $p = 0.01$, and (2) the search latency has noticeably increased. As the replication ratio decreases, the source needs to contact more peers to discover enough copies, at higher search latency. We have also measured the performance of the expanding ring algorithm and found the relative performance is consistent. The original dynamic querying algorithm fails as the probe phase often does not return any results.

To summarize, our proposed algorithm requires search cost close to the minimum and effectively bounds the search latency. We conclude that when search is for multiple results, estimating item popularity is effective. Dynamic querying like flooding algorithms can capitalize on it and can be useful for unstructured peer-to-peer networks.

6.3 Performance of Hurricane Flooding

In this subsection we evaluate Hurricane flooding to locate a single copy of the searched item. It is important to notice that dynamic querying like algorithms work only if the search is for multiple copies (in order to estimate item popularity). Therefore, in the evaluation in this subsection, we compare Hurricane flooding and the expanding ring algorithm only. The performance metrics are the search cost and the search latency. To be consistent with our analysis of Hurricane flooding, we use the search cost ratio (between the actual cost and the lower bound), and measure the latency of a search as the number of rounds of flooding required to discover the copies. We translate these two performance metrics so that we can compare Hurricane flooding and dynamic querying like flooding when evaluated with the Gnutella network topology.

6.3.1 Evaluation with Random d -regular Graphs

Figure 10 shows the search cost ratio (on left) and latency (on right) of the expanding ring algorithm and Hurricane flooding. All three instances of Hurricane flooding reduce the search cost by a large fraction, consistently over the range of replication ratio. Hurricane flooding with $b = 2.0$ appears to be the best and it reduces search cost to almost half that of the expanding ring algorithm. When b is larger ($b = 3.0$), as we analyzed, overshooting causes too many query packets. On the other hand, when b is small ($b = 1.5$), repeated query packets between two consecutive cycles become more important. Note here the number of rounds of flooding per cycle is $r = 3$. Thus the disadvantage (more repeated query packets) of using a small $b = 1.5$ may outweighs the benefit (reduced overshooting).

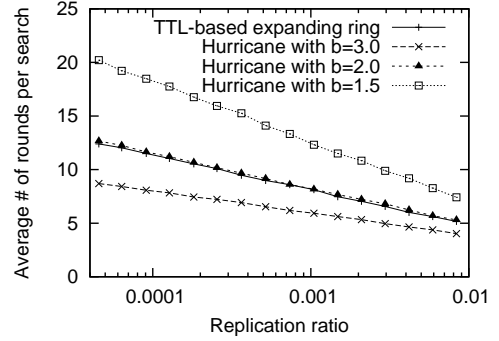
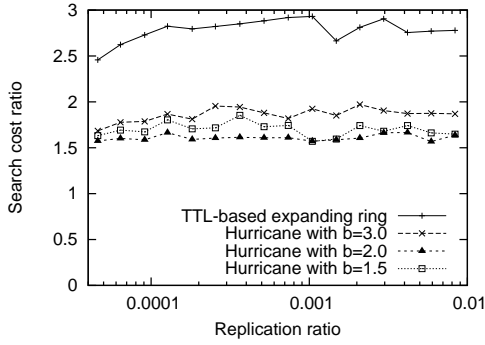


Figure 10: Performance of expanding ring flooding and Hurricane flooding in a random 3-regular graph.

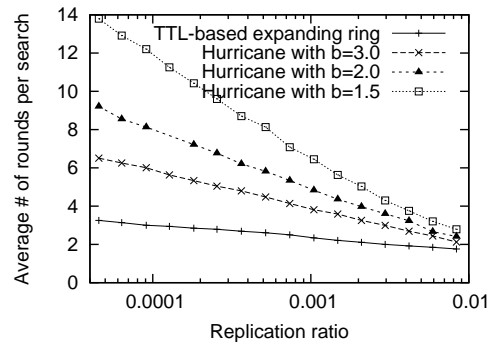
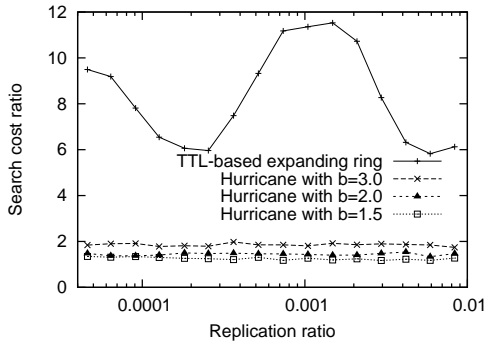


Figure 11: Performance of expanding ring flooding and Hurricane flooding in an ER graph (mean degree=30).

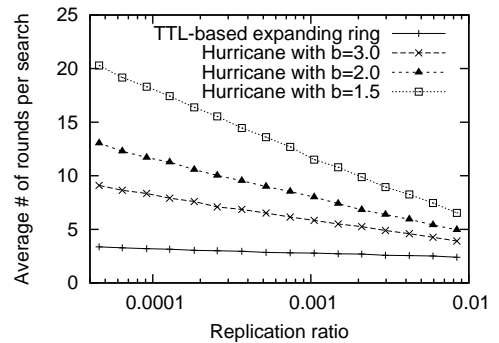
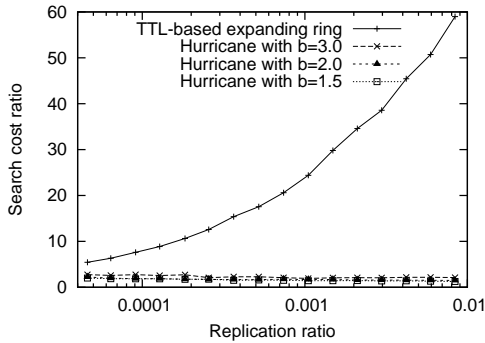


Figure 12: Performance of expanding ring flooding and Hurricane flooding in a power-law graph (mean degree=5).

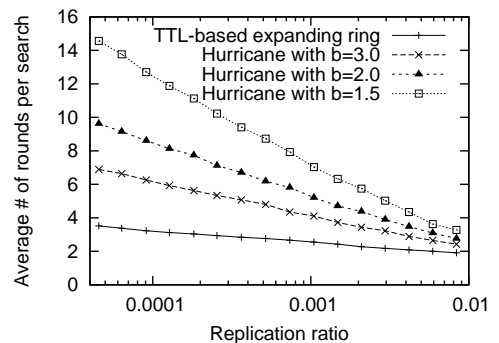
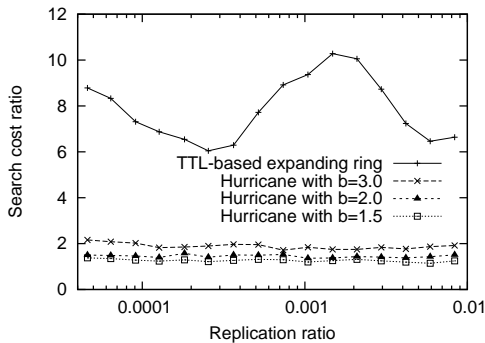


Figure 13: Performance expanding ring flooding and Hurricane flooding in a Gnutella network (mean degree=23.5).

Figure 10 shows a different scenario for the latency metric. With Hurricane flooding, the latency decreases when b is larger. The expanding ring algorithm has almost the same latency as Hurricane flooding with $b = 2.0$. This is expected as in both flooding strategies, the search horizon is doubled after each round of futile flooding. The latency of Hurricane flooding is close to the analytical results. The figure shows straight lines in the semi-logscale plot, indicating a logarithmic relationship between the latency and the target location.

All above results show that Hurricane flooding outperforms the expanding ring algorithm under the random 3-regular graph model. Since here the expanding ring algorithm is close to the California Split algorithm, it has a reasonable search cost and latency. The search cost ratio (an average quantity here) is below 3. Yet, even with such a perfect configuration for the expanding ring algorithm, Hurricane flooding still wins by a large margin.

6.3.2 Evaluation with Random ER Graphs

With the high-degree ER random graph, we can see more pronounced differences in Figure 11. The search cost of the expanding ring algorithm is almost one magnitude higher than that of Hurricane flooding. With an average degree of 30, the expanding ring algorithm expands the search horizon so drastically that overshooting becomes severe. Hurricane flooding allows fine-grained expansion without incurring many repeated query packets. Among three Hurricane flooding strategies, the one with $b = 1.5$ has the lowest search cost: it is about only 1.3 times the lower bound. Another strong impression from the figure is, the search cost line for the expanding ring algorithm has several curves. Notice each curve corresponds to suddenly the need of one more round of flooding in order to discover a copy of an item with that replication ratio. On the other hand, the expanding ring algorithm has a small latency in high-degree low-diameter peer-to-peer networks. Extremely high search cost makes it not attractive.

6.3.3 Evaluation with Random Power-law Graphs

Figure 12 shows a significant performance gain by using Hurricane flooding. Although visually not clear from the figure, it is found the search cost of three Hurricane flooding is always close to the lower bound. When $b = 1.5$, the ratio to the lower bound is often smaller than 1.5. On the contrary, the expanding ring algorithm is extremely inefficient. The difference is close to two magnitudes when the replication ratio is high. With the expanding ring algorithm, even when there are many copies of an item in the network, the search cost stays high. When no copy is

discovered after a round of flooding, merely increasing the TTL value by one for a new round of flooding will likely cause too many unnecessary query packets, especially if a query packet is forwarded by a high-degree peer.

6.3.4 Evaluation with Gnutella Network Topology

From Figure 13, the main observation is that the relative performance of different flooding strategies are close to that in Figure 11. Again, the average cost reduction ratio by using Hurricane flooding is close to one magnitude. It would be interesting to compare Hurricane flooding and dynamic querying like flooding in the Gnutella network topology. For that purpose, we translate the performance metrics. To translate the search cost, we note that to discover $k = 50$ copies of the searched item when the replication ratio $p = 0.01$, the lower bound on search cost (in packets) is 5000. Since two instances of Hurricane flooding have search cost ratio between 1.3 and 1.5 as shown in Figure 13, the total number of packets should be between 6500 and 7500, which is slightly higher than that of dynamic querying like flooding. In terms of search latency, Hurricane flooding with $b = 3$ requires latency about 1.8 times that of the expanding ring algorithm. This latency is much lower than that of the original dynamic querying algorithm and is slightly higher than that of the enhanced algorithm. The advantage of Hurricane flooding though is its determinism.

7 Conclusion

This paper has exploited novel approaches to designing efficient flooding search algorithms in peer-to-peer networks. We presented two flooding search algorithms. When the search is for multiple results, the first algorithm capitalizes on the on-the-fly estimation of the popularity of the searched item. Albeit its non-determinism, the proposed algorithm—a greedy and conservative algorithm often returns the desired number of results at minimum search cost and low search latency. The second algorithm, Hurricane flooding, is deterministic and can answer queries for any number of results. The paper has described the algorithm, and analyzed its performance in terms of search cost and search latency. In particular, we have shown that a minimum-cost Hurricane flooding requires search cost arbitrarily close to a lower bound and that it has a bounded latency that is a logarithmic function of the location of the target. To summarize, our work shows that there are great potentials to optimize flooding search algorithms for use in peer-to-peer networks. We have demonstrated significant performance gains by our algorithms. We expect our work can be extended for flooding search in other unstructured networks such as ad hoc networks and sensor networks.

Acknowledgment

The authors would like to thank Reza Rejaie and Daniel Stutzbach for making a snapshot of their Gnutella network topology available for this research.

References

- [1] W. Aiello, F. R. K. Chung, and L. Lu. A random graph model for massive graphs. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 171–180, 2000.
- [2] Y. Baryshnikov, E. Coffman, P. Jelenkovic, P. Momcilovic, and D. Rubenstein. Flood search under the california split rule. *Operations Research Letters*, 32(3), 2004.
- [3] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proceedings of IEEE ICDCS*, July 2002.
- [4] N. Chang and M. Liu. Revisiting the TTL-based controlled flooding search: Optimality and randomization. In *Proceedings of ACM MobiCom*, September 2004.
- [5] N. Chang and M. Liu. Controlled flooding search in a large network. In *Proceedings of Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, March 2005.
- [6] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM*, 2003.
- [7] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of ACM SIGCOMM*, pages 177–190, August 2002.
- [8] A. Fisk. Gnutella dynamic query protocol v0.1, May 2003. http://www9.limewire.com/developer/dynamic_query.html.
- [9] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, March 2004.
- [10] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, March 2005.

- [11] H. Jiang and S. Jin. Exploiting dynamic querying like flooding techniques in unstructured peer-to-peer networks. In *Proceedings of IEEE ICNP*, November 2005.
- [12] B. T. Loo, R. Huebsch, I. Stoica, and J. Hellerstein. The case for a hybrid P2P search infrastructure. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2004.
- [13] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of International Conference on Supercomputing*, November 2002.
- [14] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing (Special issue on peer-to-peer networking)*, 6(1), 2002.
- [15] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *ACM Multimedia Systems Journal*, 8(5), 2002.
- [16] S. Shakkottai. Asymptotics of query strategies over a sensor network. In *Proceedings of IEEE INFOCOM*, March 2004.
- [17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, August 2001.
- [18] D. Stutzbach and R. Rejaie. Characterizing the two-tier Gnutella topology. In *Proceedings of ACM SIGMETRICS (Poster)*, June 2005.
- [19] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In *Proceedings of ACM SIGCOMM Internet Measurement Conference (IMC)*, October 2005.
- [20] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer systems. In *Proceedings of IEEE ICDCS*, July 2002.
- [21] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of IEEE ICDE*, March 2003.