Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks

Hongbo Jiang Shudong Jin
Division of Computer Science, EECS
Case Western Reserve University, Cleveland, OH 44106
{hongbo.jiang,shudong.jin}@case.edu

Abstract

In unstructured peer-to-peer networks, controlled flooding aims at locating an item at the minimum message cost. Dynamic querying is a new controlled flooding technique. While it is implemented in some peer-to-peer networks, little is known about its undesirable behavior and little is known about its general usefulness in unstructured peer-to-peer networks. This paper describes the first evaluation and analysis of such techniques, and proposes novel techniques to improve them. We make three contributions. First, we find the current dynamic querying design is flawed. Although it is advantageous over the expanding ring algorithm in terms of search cost, it is much less attractive in terms of peer perceived latency, and its strict constraints on network connectivity prevent it from being widely adopted. Second, we propose an enhanced flooding technique which requires the search cost close to the minimum, reduces the search latency by more than four times, and loosens the constraints on the network connectivity. Thus, we make such techniques useful for the general unstructured peer-to-peer networks. Third, we show that our proposal requires only minor modifications to the existing search mechanisms and can be incrementally deployed in peer-to-peer networks.

1. Introduction

Peer-to-peer networks such as Gnutella, KaZaA, and BitTorrent have emerged as a new Internet computing paradigm over the past few years. A significant portion of Internet traffic is from peer-to-peer applications [12]. In unstructured peer-to-peer networks, searching for an item incurs high query traffic because the networks do not offer any directory service and they are often constructed in a random fashion. One feasible solution is flooding based search. To that end, two such techniques have been adopted.

One of them is the expanding ring algorithm [11, 5, 2, 3] adopted by many unstructured networks. It is also called TTL based controlled flooding. Briefly, this algorithm works as follows. The source of the search first chooses a small initial TTL value. It sends query packets towards all its immediate neighbors with the TTL value as a field in the packets. If a copy of the searched item is hit in a neighbor, this neighbor replies the source. The neighbor then decrements the TTL value by one and forwards the query packet to its other neighbors. This forwarding continues until the TTL field has become zero. If after one round of flooding, the source has not received the desired number of results, it may start with a new round of flooding with a larger TTL value. There could be multiple rounds of flooding before enough results are found or the source gives up. The other one is dynamic querying like techniques such as the one [7] adopted by the Gnutella network. Briefly, it works as follows. The source peer first sends query packets towards a few neighbors with a small TTL value. The purpose of this probe phase is to have an initial estimate of the popularity of the searched item. Then an iterative process takes place. In each iteration, (1) the source peer estimates the number of peers to be contacted in order to obtain the desired number of results; (2) then it calculates the TTL of the query packet to be sent to the next neighbor; (3) finally it propagates the query packet towards this neighbor. This iterative process stops when the desired number of results are returned, or all neighbors have been visited. Intuitively, this flooding algorithm is *dynamic* in that the source estimates the item popularity and adjusts the TTL value accordingly.

While the expanding ring algorithm is well studied, little is known about the dynamic flooding techniques. An evaluation and analysis of such techniques is necessary for us to understand both its desirable and undesirable behaviors, and is necessary before we can adopt such techniques in the general unstructured peer-to-peer networks. In this paper, we look into the details of such techniques and evaluate them in the representative Gnutella peer-to-peer network. We make three contributions: (1) We find the design of the existing dynamic querying technique is flawed. Although it is better in minimizing search cost, it drastically increases the latency perceived by the peers. Moreover, this technique makes a strong assumption on the network topology. This prevents the technique from being widely adopted. (2) We propose and evaluate a new technique to reduce the high latency. It requires search cost close to the minimum, eliminates the excessive latency, and also loosens the constraints on the network connectivity. Thus, dynamic querying like flooding techniques can be used by the general unstructured peer-to-peer networks. (3) We discuss the implementation and deployment issues related to our proposal. We show our enhancement requires minor or no modifications to the existing peer software in peer-to-peer networks, and can be deployed incrementally.

The rest of the paper is organized as follows. Section 2 briefly describes related work on search algorithms in unstructured peer-to-peer networks. Section 3 enumerates a set of performance requirements on search algorithms. Section 4 describes the existing dynamic querying technique and presents a performance comparison to the expanding ring algorithm. Section 5 presents our enhancing technique and demonstrates its effectiveness in eliminating the high latency and loosening the constraints on network connectivity. We then discuss the issues related to its implementation and deployment. Section 7 concludes the paper.

2. Related Work

The algorithms considered in this paper fall into the category of flooding based search algorithms. More specifically, they are all controlled flooding algorithms. Within the context of peer-to-peer networks, several studies [11, 16] have compared the performance of the expanding ring algorithm (called iterative deepening in [16]) and other controlled flooding algorithms. Within the context of wireless ad hoc networks, several studies [5, 2, 3] have empirically or analytically investigated the performance of the expanding ring algorithm, and in particular [2, 3] have proposed randomized algorithms to reduce search cost without considering the search latency.

Another category of search algorithms is random walk based techniques. They have been studied in [11, 6, 1, 13, 8, 9], in the general context of unstructured networks (including peer-to-peer networks, sensor networks, and wireless ad hoc networks). For example, [4] proposed to use a biased random walk based technique to direct queries to high-capacity nodes in the Gnutella network, and hence, to increase the chance of finding the wanted item. There is usually a tradeoff between flooding based and random walk based techniques. Random walk based techniques result in much higher latency than flooding based techniques. For energy-constrained networks such as sensor networks, random walks are considered good choices. However, in peer-to-peer networks, a long search latency results in bad user experiences. Although multiple random walkers can be used as proposed by [11], they achieve short latency at the price of increased search cost.

Another approach to making unstructured peer-to-peer networks more efficient and scalable is to modify the architecture [17, 10]. For example, both KaZaA and Gnutella currently have hierarchical architectures. A smaller number of upper level peers (supernodes in KaZaA and ultrapeers in Gnutella) are responsible for propagating and answering queries for more lower level peers. Nevertheless, efficient search algorithms for the upper level peers are still key to such hierarchical peer-to-peer networks.

3. Problem Statement and Goals

In peer-to-peer networks, search for an item works as follows. A peer accepts queries from the user. Each query includes one or more keywords and it specifies the desired number of results. Then the peer initiates a search process to discover the desired number of results that match the keywords. To do this, the peer propagates query packets (which include the keywords) towards other peers in the networks. Any result matching the keywords causes a reply message from the corresponding peer to the initiating peer. Finally the initiating peer returns the obtained results to the user.

In unstructured peer-to-peer networks, search is blind and expensive, since there is no central servers to maintain a directory, and the network is constructed in an often random fashion that it does not provide any clues to facilitate search. Usually, the only feasible solution is flooding based search. Significant network traffic may be generated due to search. Therefore, good search algorithms are key to the success of such networks.

To evaluate search algorithms, it is necessary to first specify a set of requirements (goals). To illustrate these goals, let us consider the search process of discovering the desired number of matching results. Figure 1 shows the progress of two search processes when time elapses. First, a search algorithm should be deterministic in that, it will return the desired number of results when there are abundant copies of the searched item in the network. In the figure we show that the two search processes rise above the target line. However, bad search algorithms may stop the search processes prematurely and fail to satisfy the queries. Second, a good search algorithm should be able to minimize the search cost, whereas the search cost is often defined as the total volume of query traffic. To achieve this goal, the algorithm should have two properties: (1) causing small or no overshooting of the number of returned results, and (2) having a low per-result cost. Bad search algorithms may cause heavy query traffic (either because they obtain too many results or because their per-result cost is too high). Third, a good search algorithm should return the results in a timely manner, *i.e.* the rising time of the search process should be short. Search latency is important for user satisfaction. To summarize, an ideal search algorithm should return the desired number of results at the lowest search cost and lowest latency. Figure 1 shows the first search process achieves these goals better than the second process does, as it has smaller overshooting and lower latency, although both reach the target line.

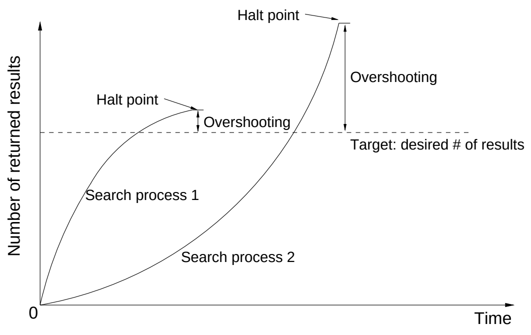


Figure 1. An intuitive example showing the requirements on search algorithms in unstructured peer-to-peer networks.

In general these three goals often conflict with each other. In order to guarantee the number of results and to minimize the search latency, unrestricted flooding (to all peers in the entire network) is the best choice. However, its search cost (query traffic) is extremely high. On the other hand, in order to minimize the search cost, an algorithm should avoid overshooting and be more conservative. Thus it may cause excessive search latency, and even not be able to return the desired number of results before stopping.

4. Evaluating Dynamic Querying like Flooding

This section describes the Gnutella dynamic querying technique as an example, and presents a performance comparison to the TTL based expanding ring algorithm. This helps us understand both the good and the bad aspects of dynamic querying like flooding techniques.

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 will propagate the query to the network using the dynamic querying protocol [7]. This protocol works in a more conservative way than the previous expanding ring algorithm. It seeks to hit the minimum number of peers necessary to obtain the desired number of results for a given search. The general process is described as follows.

The source peer starts by sending a probe query via a few neighbors with a small TTL. The purpose of this probe phase is to have an initial estimate of the popularity of the searched item. Once the probe is sent, the standard algorithm takes effect. Assume the probe does not achieve 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, called theoretical horizon, 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. Since the degree of the next neighbor can be known, the source peer can estimate the minimum TTL to reach the desired number of peers. The source peer then sends the query down 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 offers a way to dynamically adjust 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. However, the existing design of this technique is proposed in a very ad hoc fashion and has not been validated. An evaluation and analysis is necessary for us to understand both its desirable and undesirable behaviors, and it is necessary before we adopt dynamic querying like techniques in the general peer-to-peer networks.

4.2. Evaluation Methodology

We use three metrics to compare the performance of different flooding algorithms. These metrics are consistent with the requirements in Section 3. (1) The first is the number of returned results. The source peer is asked to discover a certain number of results. A good flooding algorithm should return at least close to that number of results. (2) The second metric 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. (3) The third metric is the total latency of the search process. Excessive latency is not acceptable to the end users.

We use a snapshot of the Gnutella network topology dated on February 2, 2005. This snapshot was obtained as a result of an ongoing research project at University of Oregon. The authors of [15] have described the implementation of their measurement work and the characteristics of modern Gnutella network topology. More details are provided in a technical report [14]. This snapshot includes over 160K peers and its average number of neighbors per peer is close to 24. For completeness, we show its degree distribution in Figure 2. The degree distribution in this snapshot is close to the observations from [15]. First, many peers have degree close to 32, indicating that dynamic querying's recommended number of neighbors has been well accepted by the peers. We have also noticed about 15% of the peers have degree less than 15 (notice the logarithmic scale of the plot). As recommended in [7], dynamic querying will not be used by such peers if not specially noted. Second, the degree distribution does not follow a power law. Only a few peers have extremely large numbers of peers. In our evaluation, we discard such peers for two reasons: (1) the study in [15] revealed that these peers might be used for purposes other than supporting flooding queries, and (2) such peers may not be able to faithfully propagate the query packets as they will easily be overloaded. Such peers are only over 0.01% of all peers in the snapshot.

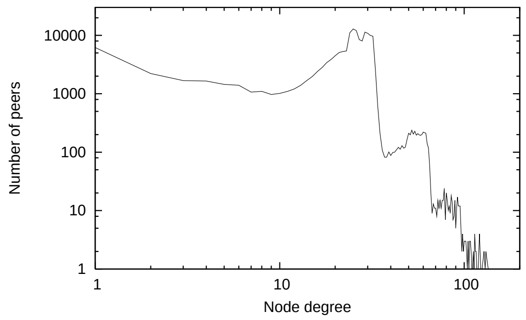


Figure 2. The degree distribution of a snapshot of the Gnutella network used in our evaluation.

We have implemented the dynamic querying algorithm in a simulator. We have followed the protocol specifications and used the recommended parameter settings in [7]. 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, just like in the expanding ring algorithm. 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 as recommended.

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. This is the maximum allowed number of results requested by a user. The probe phase returns about 10 results (with a large variation). 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. Therefore, similar conclusions have been reached. 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. That is, on average at least $1/p$ peers must be contacted to return a result. This is a property of Bernoulli trials. 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.

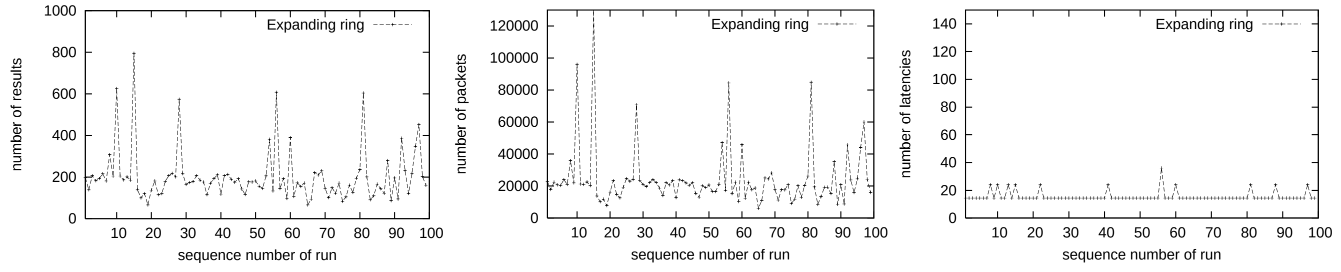
4.3. Performance Comparison to Expanding Ring

Figure 3 shows the performance of the expanding ring algorithm in terms of (a) the number of returned results, (b) the number of query packets transmitted, and (c) the peer perceived latency. Figure 4 shows the performance of dynamic querying. In each plot, the x-axis is the sequence number of our simulation runs. Thus each point in the figures represents the result from one run. Although we have run the simulator for longer periods of time, for clarity, this figure shows only the first 100 runs of each flooding algorithm. For the expanding ring algorithm, the TTL value is incremented by one after each unsuccessful round of flooding. For dynamic querying, we show the results when only high degree (at least 15) peers are enabled to use dynamic querying; for comparison (and to better understand dynamic querying), we also show the results when low degree (between 7 and 9) peers manage the search process for its leaves in Figure 5.

4.3.1 The "Good" of Dynamic Querying like Flooding

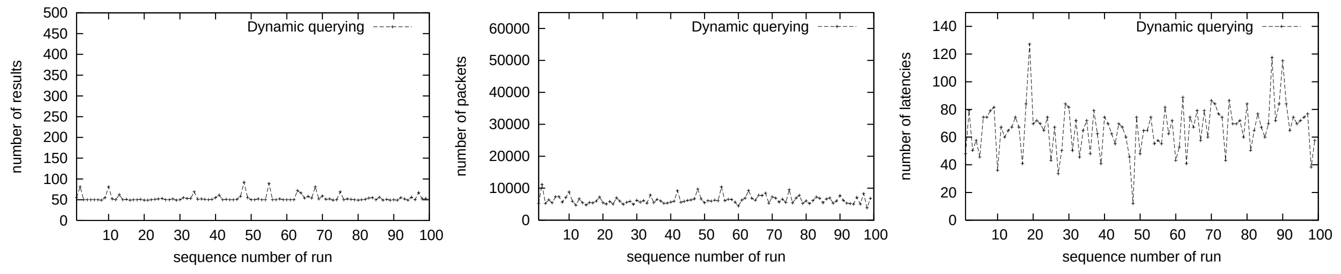
Let us first look at the number of returned results by the expanding ring algorithm and dynamic querying, see Figures 3(a) and 4(a). The expanding ring algorithm often has big overshooting and return much more results than necessary. This is because, the network topology has high average degree. The gap between two consecutive TTL values is so huge that it is almost impossible for the expanding ring algorithm to find just enough results. That means, the coarse-grain control does not work well in high degree networks. On the contrary, dynamic querying often returns just more than the desired number of results. The cautious way of flooding queries pays off.

Consequently, Figure 4(b) shows that dynamic querying reduces the search cost by several times. We have obtained more statistics from 1000 simulation runs with different



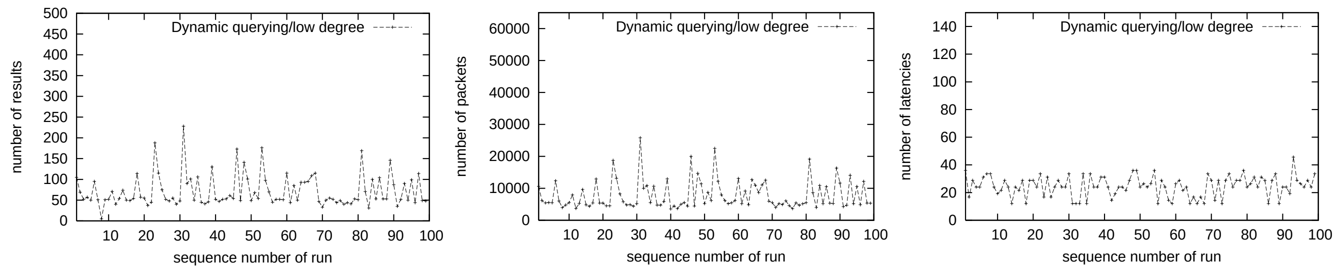
(a) Number of returned results (b) Number of transmitted packets (c) Latency

Figure 3. The performance of the expanding ring algorithm.



(a) Number of returned results (b) Number of transmitted packets (c) Latency

Figure 4. The performance of the dynamic querying algorithm.



(a) Number of returned results (b) Number of transmitted packets (c) Latency

Figure 5. The performance of dynamic querying with low degree (between 7 and 9) peers.

random seeds. On average, dynamic querying returns close to 55 results and the number of transmitted packets is close to 6400. We have also found that the search cost is usually proportional to the number of returned results. For expanding ring, the number of query packets per result is about 120. For dynamic querying, this quantity is about 117. In our configuration, we use uniform replication with replication ratio p=0.01. On average the least possible number of packets per result is $1/p$=100. Therefore, dynamic querying does not incur much unnecessary overhead (17% in our simulation).

4.3.2 The "Bad" of Dynamic Querying like Flooding

Unfortunately, dynamic querying obtains lower search cost at the price of a much undesirable property. Figures 3(c) and 4(c) show the latencies perceived by the users. 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. Given that the timeout interval is set to TTL times 2.4 seconds, 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 it is above 3 minutes. It is also much more variable.

This undesirable behavior is due to the flawed design of this algorithm. Dynamic querying takes a very conservative approach to avoid sending query packets to too many peers. In each iteration, it estimates the number of peers to be contacted in order to find the desired number of results. The source peer has many, say 30, remaining neighbors. It first sends a query packet via one of these neighbors. The TTL is set such that in this iteration the source peer will contact about 1/30 times the total number of peers to be contacted. Thus an unlucky source peer may find it must send query packets via all neighbors, one after one. Therefore, the latency can be extremely high. In other words, more likely the dynamic querying technique will lead to a search process as the second one in Figure 1 than the first one.

4.3.3 The "Ugly" of Dynamic Querying like Flooding

We say that this flooding algorithm is ugly because its performance highly depends on the connectivity of the source peer. The algorithm has a strict constraint on the network connectivity: it requires that a source peer should maintain at least 15 neighbors; otherwise it would be inefficient and the search process would become less stable. To show this problem and understand why dynamic querying needs such a strict constraint, we simulate it when the source peers have lower degrees (their degrees are between 7 and 9). Figure 5 shows the results.

The figure shows that the search cost (as well as the number of results) becomes higher, more variable and less predictable than that with high degree peers as the source peers. Furthermore, when we simulate dynamic querying from peers with even lower degrees, the cost increases further and becomes even more variable. Although the latency is also reduced, the number of returned results is often much smaller than the desired number and sometimes is much larger that the desired number. To interpret these differences, just notice that now the source peer has fewer neighbors. There are two consequences. First, since there are not many neighbors after the probe phase, the iterative process will often prematurely halt. Second, the source peer becomes more aggressive in propagating a query packet towards each neighbor. It is more likely to cause big random overshooting.

For this reason, the existing Gnutella implementation recommends each peer should have degree up to 32. This design choice has greatly transformed the underlying network topology. Currently the network has a very fat topology (with diameter just over 4 hops). However, there are at least two reasons against this design choice. First, the technique has limited its usefulness to the current Gnutella network only, not the general unstructured networks. Second, this fat topology may affect other than the flooding algorithm. For example, maintaining a larger number of neighbors incurs higher communication overhead, especially if the peer relationships are more dynamic due to frequent peer joining/leaving.

5. Exploiting Dynamic Querying like Flooding

The original dynamic querying algorithm is very conservative in propagating query packets to the network. It causes excessive search latency and can be used by high degree peers only. In this section we modify its iterative process to improve its performance and to extend its application to more general unstructured peer-to-peer networks.

5.1. Algorithm Design

The enhanced algorithm contains all aspects of a single dynamic querying, which mainly includes probe queries and TTL calculation. The source peer begins the search (for N results) by first sending a query towards a few neighbors with a fixed TTL, This is just like in the original dynamic querying. 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.

Recall that in the original dynamic querying, when there are many remaining neighbors, a query packet is propagated to just a small fraction of the required number of peers. This method is doomed to have a high latency. On the other hand, if in each iteration the TTL is set larger and the query packet is propagated to the right number of peers, then likely in just one or few iterations, there will be enough returned results. This is the intuition behind our greedy approach.

Since we use a greedy iterative process (which is more aggressive than that in the original dynamic querying), 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, for example, a confidence interval method. This is the intuition behind our conservative approach.

To complete the description of our proposal, next we present details on how to estimate the popularity of the searched item conservatively and how to calculate the TTL value for a new neighbor.

5.1.1 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

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

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

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

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

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

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 (and in our evaluation later), we choose $\delta = 3$. The value provides a high confidence level. We do not think an even larger value is necessary. Our main reason is, since the average search cost is representative of the query load in the network, an occasional (very rare indeed) overshooting is not overwhelmingly concerned.

Notably, this above approach handles well the special case of no returned result ($n = 0$). Our enhanced algorithm has a conservative estimate of $m' = \delta > 0$ returned results. It implies that even if the searched item has a much variable replication ratio, our enhanced algorithm still works. On the contrary, the original dynamic querying algorithm needs to handle this special case separately (the source peer usually increases the TTL value for the next neighbor).

5.1.2 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:

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

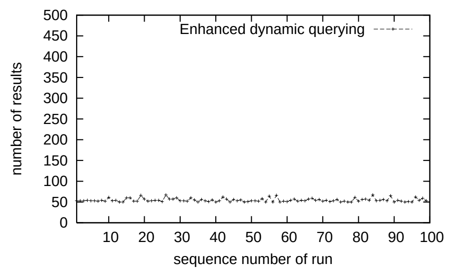
This is equivalently to

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

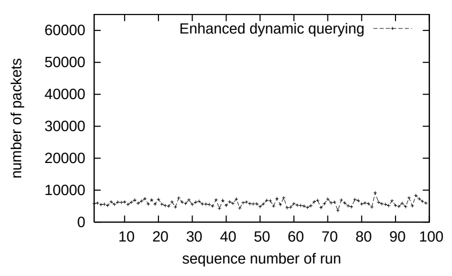
One problem is, the obtained TTL value is often a floating number. There are two approaches to handling it. The first and the simpler one is, we can round this floating number into an integer (We will evaluate this approach later). To achieve fine-grain control, we can take the second approach and modify peers' forwarding algorithm slightly. When a peer receives a query packet with TTL ≥ 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 $(d^{TTL} - 1)/(d-1)$.

5.2. Performance Evaluation

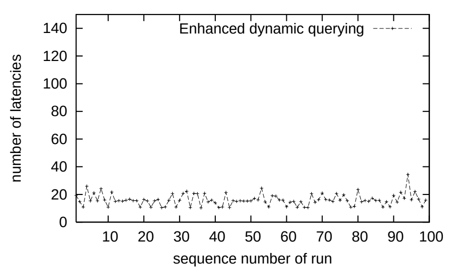
We have implemented the enhanced algorithm in our simulator. It uses the same parameter settings of dynamic querying. In addition, we set $\delta = 3$ in estimating item popularity and set the TTL value to be the calculated floating number. The average degree $D \approx 24$ is calculated from the topology. We have also found the algorithm is not sensitive to this value. Hence, in real implementation the algorithm can use an approximate value of D. Figure 6 shows the performance of the enhanced algorithm when the source peers have degree no less than 15, and Figure 7 shows its performance when the source peers have degree between 7 and 9. Thus, we should compare these two figures to Figures 4 and 5, respectively. In both figures we show the results from the first 100 simulation runs. In addition, we have obtained statistics from more simulation runs of various algorithms, and plotted the results in Figure 8. In this last figure, we show both the average quantities (of result count, packet count, and latency) and the 90-percentiles from 1000 simulation runs of each algorithm. In the figure, "TTL" represents the expanding ring algorithm; "DQ" and "DQ/low degree" represent dynamic querying with high degree (at least 15) peers and with low degree (between 7 and 9) peers, respectively; "DQ+" and "DQ+/low degree" represent our enhanced algorithm with high degree (at least 15)



(a) Number of returned results

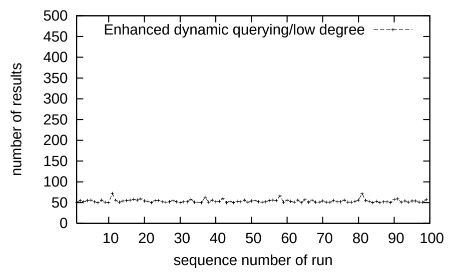


(b) Number of transmitted packets

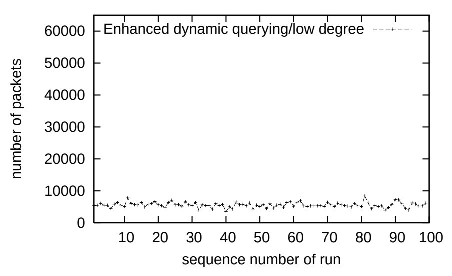


(c) Latency

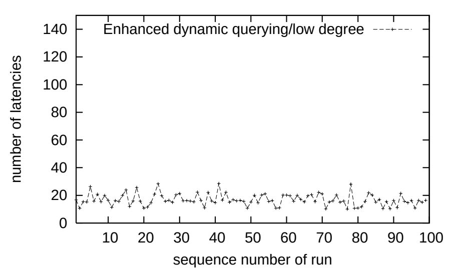
Figure 6. The performance of our enhanced algorithm.



(a) Number of returned results

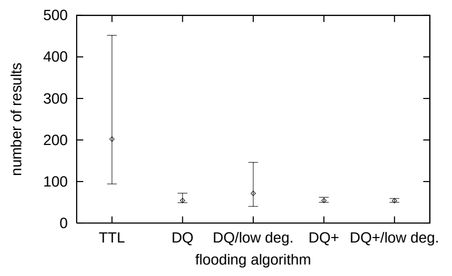


(b) Number of transmitted packets

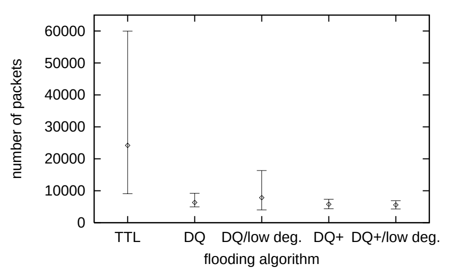


(c) Latency

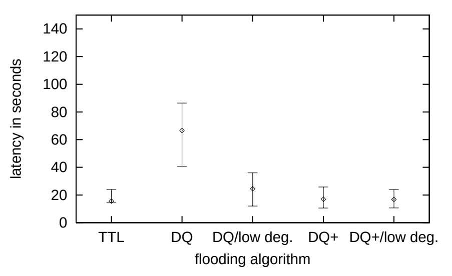
Figure 7. The performance of our enhanced algorithm with low degree (between 7 and 9) peers.



(a) Number of returned results



(b) Number of transmitted packets



(c) Latency

Figure 8. A performance comparison of expanding ring (TTL), dynamic querying (DQ), and our enhanced algorithm (DQ+).

peers and with low degree (between 7 and 9) peers, respectively. Thus, the five mean values (and confidence intervals) correspond to the results in Figures 3-7.

The figures show that the enhanced algorithm obtains almost exactly the desired number of results, see Figures 6(a) and 8(a). We also observe that it has slightly lower search cost than the original dynamic querying. The total number of packets transmitted per run is only 5900, compared to approximately 6400 packets for the original dynamic querying. The number of transmitted packets per results is about 109. This is very close to the least possible number of 100 with our configuration.

Figure 7 shows the results from the simulations of the enhanced algorithm when the source peer has a lower degree (between 7 and 9). Recall that the original algorithm cannot be used by these low degree peers to perform flooding efficiently (see the results in Figure 5 and the discussions in Section 4). However, with our enhanced algorithm these low degree peers often find the right number of results at low search cost. The performance (number of returned results and number of transmitted packets) is much more stable and predictable. Therefore, our enhanced algorithm loosens the strict constraints on the network connectivity. Note, given that the probe phase uses three neighbors, the iterative process has the other 4 to 6 neighbors available. Our enhanced algorithm often does not need that many iterations to reach the target number of results. We have also simulated the enhanced algorithm when the source peer has an even lower degree (between 4 and 6). Two of these 4-6 neighbors are used in the probe phase. With this configuration, the enhanced algorithm does reasonably well (the average number of results is about 55 and the average number of packets per result is about 106).

Finally Figures 6(c), 7(c), and 8(c) show that the enhanced algorithm effectively bounds the search latency. Its average latency of about 16 seconds is very close to that of the expanding ring algorithm, and is much smaller than that of the original dynamic querying. Moreover, the latency is less variable.

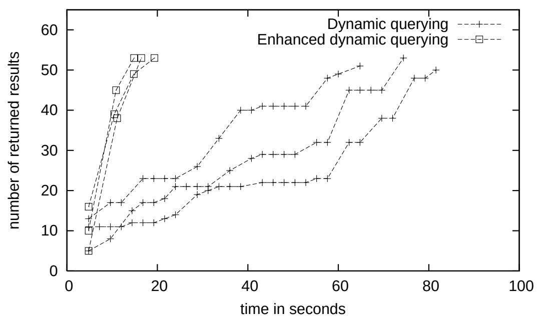


Figure 9. Traces from six typical simulation runs of the original dynamic querying algorithm and the enhanced algorithm.

To further understand the differences between the two algorithms, Figure 9 plots six typical traces of simulation runs, three of them using the original dynamic querying and the other three using the enhanced algorithm. Each curve shows the number of returned results when time has elapsed since the start of the query process. The first point of each curve represents the end of the probe phase and each subsequent point represents the end of each iteration. The purpose of this figure is just to contrast the two algorithms. With the original dynamic querying, the number of returned results increases slowly, reflecting the very conservative nature of the iterative process. On the contrary, the enhanced algorithm is initially more aggressive, then levels off when the number of returned results approaches the target. As a result, the enhanced algorithm needs fewer iterations and lower search latency.

6. Implementation and Deployment Issues

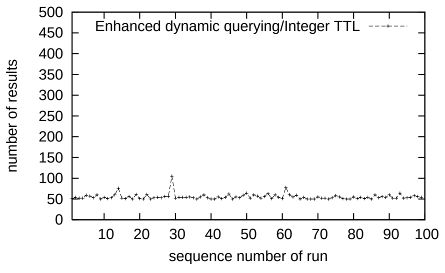
In this section we discuss issues related to the implementation and deployment of dynamic querying like flooding techniques. We use the Gnutella network as an example, but the conclusions also apply to the general unstructured peer-to-peer networks which implement flooding search.

In dynamic querying like flooding techniques (including the original algorithm in Gnutella and our enhanced algorithm), the source peers manage the search process and other peers in the network faithfully forward the query packets. In the original algorithm, there is no need to upgrade the software in the other peers. As we discussed in Section 5.1, there is one exception in our enhanced algorithm: the obtained TTL value is often a floating number and currently this may not recognized by the peers in the network. We proposed two approaches to handling this.

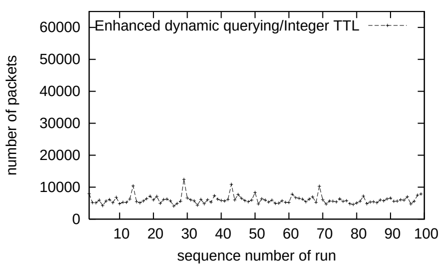
The first approach is, we round the floating TTL value into an integer. Thus, the algorithm does not require any upgrade in the other peers and it can be readily deployed too. One issue is how to round this number. We should weigh it towards the floor of the floating number since picking the ceiling would more likely cause overshooting. To evaluate this approach, Figure 10 shows the results of 100 simulation runs when the TTL value is rounded into an integer such that the ratio between picking the ceiling and picking the floor is 0.3:0.7. Compared to the original dynamic querying (Figure 4), this figure shows that this simplified version of our enhanced algorithm has approximately the same performance in the number of returned results and the number of transmitted packets, but has much lower latency (which averages just over 28 seconds). Therefore, even though using a coarse-grain TTL setting, the enhanced algorithm is much better than the original dynamic querying.

Our second approach is to modify peers' forwarding algorithm to support floating TTL values, as described in Section 5.1. For example, in the Gnutella network, we can slightly modify the query message header. The message header includes a message identifier (16 bytes), payload type (1 byte), TTL (1 byte), hops (1 byte), and payload length (4 bytes). The existing peers interpret the TTL value as an integer, so we use this field to represent the integral part of the floating TTL value. To represent the remaining fraction, we have two choices. The first choice is, we use one of those reserved bytes in in the header. For example, the last byte of the message identifier is currently set to all 0's; the last two bytes of the payload length are always 0's as the length is limited (less than 4KB). The second choice is, we append one byte to the entire query message, and accordingly increases the payload length by one. This just works well since the payload length field is only for peers to find the beginning of the next message in the input stream.

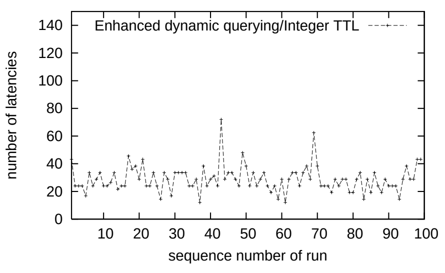
Finally, an important issue related to deployment is the incentives for users to adopt our enhanced protocol. We argue that the original dynamic querying algorithm is designed to be *altruistic*. That is, the peers using the protocol restrain themselves and avoid flooding query packets to a large population in the network. However, an individual user behind such peers does not benefit from the adoption of this algorithm (indeed, they may suffer from the high search latency). Therefore, *selfish* users in the current network may still choose peer software that implements more aggressive flooding search. On the contrary, although our enhanced algorithm is designed to have altruism too, the users do not suffer from a high latency. Therefore, we expect the users will be more willing to adopt our proposal.



(a) Number of returned results



(b) Number of transmitted packets



(c) Latency

Figure 10. The performance of the enhanced algorithm when the TTL value is rounded into an integer.

7. Conclusion

This paper exploits new flooding techniques other than the expanding ring algorithm in unstructured peer-to-peer networks. Dynamic querying like techniques are different in that they adjust the scope of search based on the estimated popularity of the searched item. We first use Gnutella's dynamic querying algorithm as an example and present an evaluation. We find it causes excessive search latency and adds strict constraints on the network connectivity. We then propose an enhanced algorithm—a greedy and conservative flooding algorithm which requires search cost close to the minimum, reduces the latency by more than four times, and loosens the constraints on the network connectivity. Our proposed algorithm requires only minor or no modifications to the existing peer software. To summarize, our work reveals that there are potentials to make dynamic querying like flooding technique useful for the general unstructured peer-to-peer networks.

We are currently working in two directions. First we will analyze various search algorithms for unstructured peer-to-peer networks to gain deeper understanding of them. We need consider both flooding based and random walk based algorithms. Second we will explore flooding based algorithms for a wider range of applications, in particular other unstructured networks such as wireless ad hoc and mesh network, and sensor networks.

Acknowledgments The authors are grateful to Reza Rejaie and Daniel Stutzbach for making a snapshot of Gnutella topology available for this work. The authors would like to thank Limin Wang for discussions on peer-to-peer search, and the anonymous reviewers for their valuable comments.

References

[1] D. Braginsky and D.Estrin. Rumor routing algorithm for sensor networks. In *Proceedings of IEEE ICDCS*, July 2002.

[2] N. Chang and M. Liu. Revisiting the TTL-based controlled flooding search: Optimality and randomization. In *Proceedings of ACM MobiCom*, September 2004.

[3] 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.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM*, 2003.

[5] Z. Cheng and W. B. Heinzelman. Flooding strategy for target discovery in wireless networks. In *Proceedings of ACM International Workshop on Modeling, analysis, and simulation of wireless and mobile systems (MSWiM)*, September 2003.

[6] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of ACM SIGCOMM*, pages 177–190, August 2002.

[7] A. Fisk. Gnutella dynamic query protocol v0.1, May 2003. http://www9.limewire.com/developer/dynamic_query.html.

[8] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, March 2004.

[9] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, March 2005.

[10] 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.

[11] 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.

[12] 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.

[13] S. Shakkottai. Asymptotics of query strategies over a sensor network. In *Proceedings of IEEE INFOCOM*, March 2004.

[14] D. Stutzbach and R. Rejaie. Characterizing today's Gnutella topology. Technical Report CIS-TR-04-02, CIS, University of Oregon, November 2004.

[15] D. Stutzbach and R. Rejaie. Characterizing the two-tier Gnutella topology. In *Proceedings of ACM SIGMETRICS (Poster)*, June 2005.

[16] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer systems. In *Proceedings of IEEE ICDCS*, July 2002.

[17] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of IEEE ICDE*, March 2003.