# Search and Replication in Unstructured Peer-to-Peer Networks

Qin Lv
Dept. of Computer Science
Princeton University
*qlv@CS.Princeton.EDU*

Pei Cao
Cisco Systems, Inc.
*cao@cisco.com*

Edith Cohen
AT&T Labs-Research
*edith@research.att.com*

Kai Li
Dept. of Computer Science
Princeton University
*li@CS.Princeton.EDU*

Scott Shenker
ICSI
*shenker@icsi.berkeley.edu*

## ABSTRACT

Decentralized and unstructured peer-to-peer networks such as Gnutella are attractive for certain applications because they require no centralized directories and no precise control over network topology or data placement. However, the flooding-based query algorithm used in Gnutella does not scale; each individual query generates a large amount of traffic and large systems quickly become overwhelmed by the query-induced load. This paper explores various alternatives to Gnutella's query algorithm and data replication strategy. We propose a query algorithm based on multiple random walks that resolves queries almost as quickly as Gnutella's flooding method while reducing the network traffic by two orders of magnitude in many cases. We also present a distributed replication strategy that yields close-to-optimal performance.

## 1. INTRODUCTION

Since Napster's introduction in 1999, there has been an explosive increase in Peer-to-Peer (P2P) network usage. Recent measurement data suggests that P2P applications are having a very significant and rapidly growing impact on Internet traffic. Currently, there are several different architectures for P2P networks:

**Centralized:** Systems like Napster maintain a constantly updated directory at central locations. Nodes issue queries to the central directory server to find which nodes hold the desired files. Such centralized approaches scale poorly and have single point of failure.

**Decentralized but Structured:** These systems have no central directory server, but have a significant amount of structure: the P2P network topology is tightly controlled and files are placed at specified locations that makes subsequent queries easier to satisfy. In *loosely structured* systems like Freenet [2], file placement is based on hints; In *highly structured* systems, both the structure of the P2P network and the placement of files are precisely determined. There is a growing literature on highly structured P2P systems which support a hash-table-like interface; see [3, 5, 4, 6].

**Decentralized and Unstructured:** These systems have neither a centralized directory nor any precise control over the network topology or file placement. The network is formed by nodes joining the network following some loose rules. The most typical query method is flooding, where the query is propagated to all neighbors within a certain radius [1]. These unstructured designs are extremely resilient to nodes entering and leaving the system. However, the current search mechanisms are extremely unscalable, generating large loads on the network participants.

In this paper, we study more-scalable alternatives to existing Gnutella algorithms, focusing on the search and replication aspects. We propose a k-walker random walk algorithm that greatly reduces the load generated by each query. We also show that active replication (where files can be stored at arbitrary nodes) produces lower overall query load than non-active node-based replication (*i.e.*, a file is only replicated at the requester).

## 2. FINDING BETTER SEARCH METHODS

Gnutella uses TTL-based flooding to search for objects. This scheme introduces many duplicate messages, particularly in high connectivity graphs. Also, it's not easy to choose the appropriate TTL, which is affected by the overlay topology as well as the replication ratio. To address the TTL selection problem, one can use successive floods with increasing TTLs. A node starts a flood with small TTL. It increases the TTL and starts another flood if the current search is not successful. This process repeats until the object is found. This method is called *Expanding Ring*. This method performs well when hot objects are replicated more widely than cold objects.

| query/replic | metrics | flood | ring | check | state |
|---|---|---|---|---|---|
| Uniform / Uniform | #hops | 3.40 | 5.77 | 10.30 | 7.00 |
| | #msgs/node | 2.51 | 0.06 | 0.03 | 0.02 |
| | #nodes probed | 9220 | 536 | 149 | 163 |
| | peak msgs | 6.37 | 0.26 | 0.22 | 0.19 |
| Zipf-like / Proportional | #hops | 2.51 | 4.03 | 9.12 | 6.66 |
| | #msgs/node | 1.86 | 0.05 | 0.03 | 0.02 |
| | #nodes probed | 7847 | 396 | 132 | 150 |
| | peak msgs | 5.23 | 0.20 | 0.17 | 0.14 |
| Zipf-like / Square root | #hops | 2.70 | 4.24 | 5.74 | 4.43 |
| | #msgs/node | 2.31 | 0.03 | 0.02 | 0.02 |
| | #nodes probed | 8983 | 269 | 89 | 109 |
| | peak msgs | 6.14 | 0.12 | 0.17 | 0.16 |

**Table 1: Simulation results for Random Graph.**

However, expanding ring does not address the message duplication issue inherent in flooding. So we try a different approach, *Random Walk*, where a query message is forwarded to a randomly chosen neighbor at each step until the object is found. We call this message a *walker*. The standard random walk technique uses only one walker, which greatly reduces the message overhead, but has much longer search delay. To decrease the delay, we increase the number of walkers. That is, the requesting node sends out $k$ query messages, and each query message takes its own random walk. The expectation is that $k$ walkers after $T$ steps should reach roughly the same number of nodes as 1 walker after $kT$ steps. Therefore, by using $k$ walkers, we expect to cut the delay down by a factor of $k$. To terminate the walks, each walker periodically checks with the requesting node to decide if it should continue searching. Nodes can also keep states so that different walkers for the same query can be sent to different neighbors.

We simulate various search methods under different query and replication models for different network topologies (random, power-law, Gnutella, grid). Table 1 shows the simulation results for a 9836-node random graph. "check" is 32-walker random walk with checking, and "state" is 32-walker random walk with checking and state keeping. Other results are omitted due to space constraints. Our results show that $k$-walker random walk is a much more scalable search method than flooding.

The key to scalable searches in unstructured network is to cover the right number of nodes as quickly as possible and with as little overhead as possible. However, in reaching the required node coverage, one must pay attention to the following:

- *Adaptive termination is very important.* TTL-based mechanisms do not work. Any adaptive/dynamic termination mechanism must avoid the implosion problem at the requester node.
- *Message duplication should be minimized.* Preferably, each query should visit a node just once. More visits are wasteful in terms of the message overhead.
- *Granularity of coverage should be small.* Each additional step in the search should not significantly increase the number of nodes visited. This is perhaps the fundamental difference between expanding ring flooding and multiple-walker random walk.

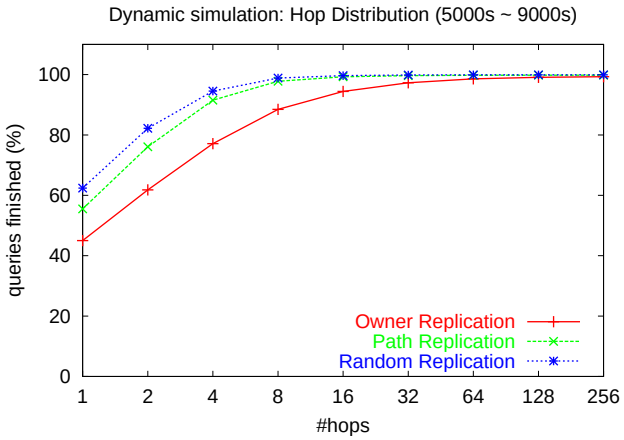


**Figure 1: Cumulative distribution of #hops.**

## 3. REPLICATION

In certain P2P systems like Gnutella, only the requester stores a copy of the requested object (*owner* replication). We consider more proactive replication strategies: *path* replication (replicate along the path from the requester to the provider); and *random* replication (same number of replicas as in path replication, but replicas are placed randomly among the sites probed). We can see from Figure 1 that they both outperform owner replication. Our analysis shows that they are close-to-optimal.

## 4. CONCLUSIONS

Through simulations and modeling studies, we have learned that scalable search algorithm designs for unstructured peer-to-peer networks should consider three properties: adaptive termination, minimizing message duplication, and small granularity of coverage. We also show that path replication and random replication are close-to-optimal.

## 5. REFERENCES


[1] Clip2.com. The gnutella protocol specification v0.4. In *http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf*, 2000.

[2] OpenSourceCommunity. The free network project. In *http://freenet.sourceforge.net/*, 2001.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM'2001*, Aug. 2001.

[4] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of SOSP'01*, 2001.

[5] 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 SIGCOMM'2001*, Aug. 2001.

[6] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.