*Proceedings of 2003 IEEE Globecom Confernece, (Globecom'03),* San Fransisco, California, USA, December 1-5, 2003.

# FloodTrail: an Efficient File Search Technique in Unstructured Peer-to-Peer Systems

Song Jiang and Xiaodong Zhang

Department of Computer Science,
College of William and Mary
Williamsburg, VA 23187, USA

***Abstract*— Searching efficiency is a decisive factor concerning scalability in large-scale peer-to-peer (P2P) file sharing systems. While flooding is the most commonly used and user-performance oriented method to broadcast query across an unstructured P2P network, it generates a large number of redundant messages. Our study shows that more than 70% of messages are redundant using flooding in a moderately connected network, which imposes an increasingly excessive burden on the underlying infrastructure, hindering the growth and scalability of P2P systems. To reduce the use of flooding as well as its associated overhead, we utilize the access trail left by a standard flooding, which is a collection of P2P links used by non-redundant messages. Thus the multiple queries following the flooding can be broadcasted along the trail to achieve two goals: (1) The ability of flooding to achieve short response time is maintained; (2) the cost of a broadcast is minimized. Though the trail can be partially damaged in an ad hoc system with constant peer arrivals and departures, we used repeated trail refreshings and additional trail links to make a trail consistently available for query broadcast. We call this trail-based technique *FloodTrail*.**

**We evaluated the performance of FloodTrail for P2P systems for Web Contents sharing. Simulation results show that FloodTrail could reduce flooding traffic by up to 57%, while maintaining almost the same coverage as flooding.**

## I. Introduction

Recent music file sharing systems have spurred tremendous interests in peer-to-peer (P2P) distributed systems. While fully decentralized architectures such as what is adopted in Gnutella provide potentially highly scalable systems, searching efficiency is a decisive factor concerning scalability. Though the query routing by flooding is very naive and often leads to inefficient use of bandwidth, flooding is currently the most widely used method to reach a large amount of peers in a short period of time.

In a flooding, a peer sends a message to its neighbors, which in turn forward it to all their neighbors except the message sender. Each message has a globally unique message ID. A message received by a peer that has the same message ID as the one received previously is considered a redundant message and will be discarded. Flooding is conducted in a hop by hop fashion counted by Time-to-Live (TTL). A message starts off with its initial TTL, which is decremented by one when it travels across one hop. A message comes to its end either because it becomes a redundant message or because its TTL is decremented to 0. Excessive traffic overhead is a major limit of flooding. When multiple messages with the same message ID are sent to a peer by its multiple neighbors, all but the first messages are considered as redundant messages. These redundant messages are pure overhead: they increase the network transferring and peer processing burden without enlarging the propagating scope. However, because there are no controls and no accurate information on network topologies and locations of desired files in ad hoc, unstructured P2P systems, the role of indiscriminate flooding is essential for a rapid and reliable query broadcasting with a large coverage.

A flooding not only provides a large scope broadcasting, but also leaves valuable *trail* information, which is largely ignored in the previous research. Trail of a flooding is a collection of P2P links, along which a message reaches a new peer, which sees the query for the first time, during a flooding. Those links that are used to transmit redundant messages in flooding are excluded from trail. A trail generated by a flooding provides an optimal multicast tree for a query broadcast from the same peer as the one where the flooding is initiated, through which a query can reach each peer in the shortest time and without any redundant messages. So the trail can be used for broadcasting the subsequent queries. We call the trail-based broadcast technique *FloodTrail*. Actually, history information about a flooding has been effectively utilized – a globally unique message ID is kept in every peer that saw the query for a certain period of time to recognize and discard redundant messages. We assume the trail information can also be recorded across the peers that have seen a query for its future use. The main concern on the use of trail for broadcasting comes from the transiency of P2P systems. A peer could join in or depart from the ad hoc system, which could disrupt a trail and make it unavailable for query broadcasting. In this paper we will answer the following questions:

1) How to mend a trail in the face of constant peer arrivals and departures?
2) How does the transiency of P2P systems affect the performance of trail in terms of coverage and redundant message reduction?
3) How to set the time length during which a trail is usable?

4) Is the additional space overhead for storing trail information in a peer acceptable?

## II. Related Work

To address the flooding problem in Gnutella style P2P systems, researchers and practitioners have proposed some solutions. For example, the expanding ring [2] (or iterative deepening [10]) initiates several successive flooding searches with increasing TTLs. After each flooding, the requesting peer has to wait for some period of time to collect response messages. The continuation of a flooding depends on whether enough responses have been received so far. These schemes only work for queries for highly popular and widely duplicated files. They even could generate more traffic than standard flooding for less popular files due to its repeated use of floodings. In some schemes, interest-based locality has been assumed and exploited by abstracting hints from previous (flooding) search [8], [6]. Their effectiveness depends on the content distribution and user access patterns. Another prevalent solution adopts super-peers to provide a partially centralized location service [9]. These super-peers connect to each other forming a pure Gnutella style network, where flooding is used for query search. In summary, flooding is still a major component in all these schemes, and its efficiency remains as an issue to be addressed. $LightFlood$ [1] directs queries with small TTLs routing across a tree-like sub-overlay to reduce excessive flooding traffic. However, $LightFlood$ could increase the broadcast delay, and it can not differentiate the various bandwidth of links.

Our $FloodTrail$ technique utilizes the temporal locality of floodings to directly reduce flooding cost. Many schemes aiming at system scalability are expected to benefit from the technique by integrating it into these schemes.

In the following section, we will describe FloodTrail technique in detail. In Section 4, we evaluate the technique through real-life topology and content request traces. We also study the performance implication of system transiency, and the length of the period of time during which a trail is usable. Then we discuss its additional overhead. We conclude the paper in Section 5.

## III. The FloodTrail Technique

In the FloodTrail technique, each peer has a globally unique peer ID. A query sending from a peer carries its peer ID. Initially a peer floods its query across P2P network with a flag called *Flood*. Any peer receiving a message with the flag forwards the message to each of its neighbors except the one from which it received the messages, just as what it does in a standard flooding. And it marks each of its links to those neighbors as a trail link. When one of its neighbors gets the message and it is not the first one it received for the same query, it sends back a small *trail link invalidation* message, thus the peer receiving the returned message invalidates its previously marked trail link. If the neighbor receives the message and it is the first one for the query, it also marks the link from which it gets the message as a *trail link*. In this way the flooding generates a trail composed by trail links, which is actually a tree rooted from the peer initiating the query and covers all the peers covered by the flooding. Any links transmitting redundant messages are excluded from the trail. Because the links transmitting first arriving messages are selected as trail links, the path from the root peer to a peer along trail links is the shortest one between the two peers in terms of transmitting time. Each peer can built a trail tree rooted from itself through an initial flooding for its future query broadcast. Figure 1 illustrates the construction of a trail from a specific peer. From the figure we can see that (1) The query broadcast through a trail takes only $N$ messages to reach $N$ peers, and eliminates all the redundant messages in the ideal case. (2) The shortest paths are selected for each destination peers, and slow links are dynamically avoided.

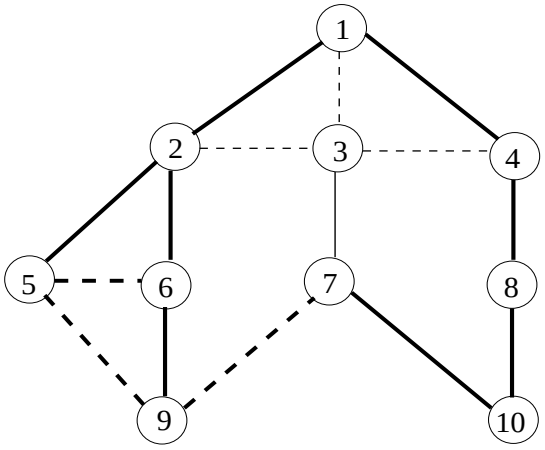


Fig. 1. Illustration of the construction of a trail. Node 1 is the source node. Solid lines are for trail links. Suppose that node 3 is connected into the system via low-bandwidth connection (e.g. dial-up connection). Those lines which are not darkened represent the low-bandwidth links. The messages passing through these slow links subsequently become redundant ones, and are excluded from the trail.

However, in an ad hoc system, peers could constantly join in or depart from the system network. For a relatively long period of time, these moving peers are accumulated and could significantly change the connections among peers. This leaves many peers unreachable either because they are newly added peers and not on the trail or because some peers on the path from the root to those peers leave. So we set a timer at each peer, which is reset right after the initiation of a flooding for a trail construction. A peer's trail is only valid for query broadcast before the timer of the peer expires. The timer expires after a certain period of time called *window time*. A query will be flooded with a *flood* flag to construct a new trail to replace the current one once the timer expires, which we call *trail refreshing*. Repeated trail refreshings help reduce the impact of major accumulated topology changes on the use of trail. For minor topology changes within a window time, we do the following maintenance work: (1) When a peer leaves, all of its neighbors, which we call *lost peers*, designate all of their links to their neighbors as trail links, and their corresponding neighbors are notified. Thus these lost peers try their best

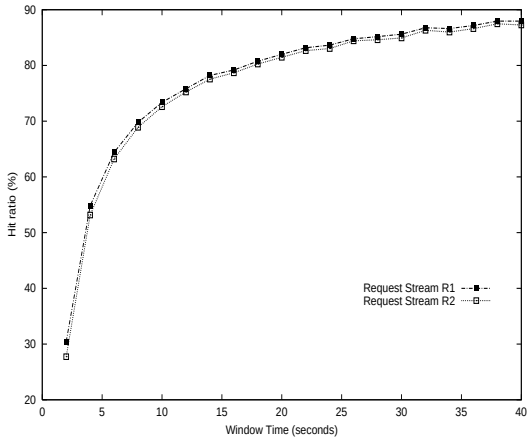


Fig. 2. The hit ratios (the number of requests that can broadcasted across the trail among the total requests) with various window times for request stream R1 and R2.

to re-connect to other peers, possibly introducing redundant messages. (2) When a peer joins, it is treated as a lost peer and the link to each of its neighbors is marked as a trail link. In this way any temporarily lost peers are re-connected to the trail and are able to receive queries from the root peer.

## IV. Performance Evaluation

We use trace-driven simulations for performance evaluation. In order to investigate the feasibility and performance potential of the FloodTrail technique, we used Web proxy workloads which reflect user access patterns for Web content, which is envisioned being shared in a P2P Web content distribution system [8]. The Web proxy trace we used is composed of one-day traces from five of Boeing's firewall proxies [12]. We selected two one-hour traces for our experiments, which is the median session duration reported for P2P systems [7]. The first request trace, R1, was collected from 5PM to 6PM, and contains 8,283 client nodes and 838,150 requests. The second trace, R2, was collected from 10AM to 11AM, and contains 22,126 client nodes and 1,989,785 requests. Each entry of the traces contains a request time and a request client. To simulate the performance of Gnutella, we used Gnutella connectivity graphs collected in early 2001 [11]. All graphs have a bimodal power-law degree distribution. We selected two graphs with different average degrees and each with its number of peers similar to the number of clients in a request stream trace. The first one has 8,351 peers with an average degree of 2.78. The second one, T2, has 22,509 peers with an average degree of 4.77. We randomly map the clients in stream R1 to graph T1, and the clients in stream R2 to graph T2. In all broadcasts, we used a maximum query TTL of 7, which is the application default for many Gnutella clients.

### A. The Impact of Window Time on Trail Broadcast

As we have mentioned, trail broadcast is used within a window time, which is after a trail is refreshed by an initial flooding and before a timer is expired. Thus window time is directly related to the number of times for which trail broadcasts can be used. We denote the ratio of the number of requests for which trail broadcasts can be used and total requests in a stream as the hit ratio for a given window time. Figure 2 shows the hit ratios with a large range of window times from 2 seconds to 40 seconds for request stream R1 and R2. From the figure we see that hit ratio rises rapidly with small window times. For example, a window time of 10 seconds makes 73% of requests eligible for trail broadcasts, and only 27% of requests have to use standard floodings in stream R1. In the current music file sharing P2P systems, users usually experience the cycle like multiple trial queries, file selection and download, and listening. So we expect clustering of query requests during the live session of a user, which would generate a high hit ratio for a moderate window time. High hit ratio provides a large potential for our FloodTrail technique to exploit for broadcast cost reduction.

### B. The Impact of Transiency on Trail Broadcast

In this section we study the impact of peer arrivals/departures on the performance of trail broadcasts. Even though we add trail links for lost peers due to peer departures, some lost peers could be beyond the coverage of a 7-hop broadcast. Thus the coverages of trail broadcasts could be reduced with peer departures. The number of redundant messages could increase due to the introduction of trail links for lost peers and newly arriving peers.

To simulate the behavior of peer moving and keep the size of a graph fixed, we add a peer with the same degree as the one removed into the graph once we remove a peer in the graph. The neighbors of the added peer are randomly selected from the peers in the graph. For each peer in a graph, we first generated a trail rooted from the peer through a flooding, then broadcasted a query across the trail, which could be updated thereafter due to peer arrivals/departures. We collected the coverage of the broadcasts initiated from every peers and their traffic, which is the number of messages forwarded, and reported their averages in Figure 3 for graphs T1 and T2 with various number of moving peers. We have tested a large range of number of moving peers, from 0.1% to 4.9% of total peers in a graph. Actually the upbound of 4.9% is largely exaggerated, and we would not expect a stable P2P system changed that much in a window time of a couple of seconds.

The coverage of a trail broadcast without a moving peer is the same as that of a standard flooding. From the figures we can see that the coverage reduction due to moving peers is very limited. For example, the coverage is reduced by only 4.8%, 8.8% and 12.3%, respectively for graph T1, and by only 3.9%, 6.8% and 9.8%, respectively for graph T2, when 1%, 2%, and 3% of total peers are moved. The reason for the small coverage reductions is that moving peers are randomly selected, a peer close to the trail tree leaves has much higher probability to be chosen than a peer close to the tree root, because the number of peers close to a root is small. The traffic of a trail broadcast without a moving peer is the same as the number of its covered peers, because there are no redundant messages in the process of broadcast. We observed that the traffic could

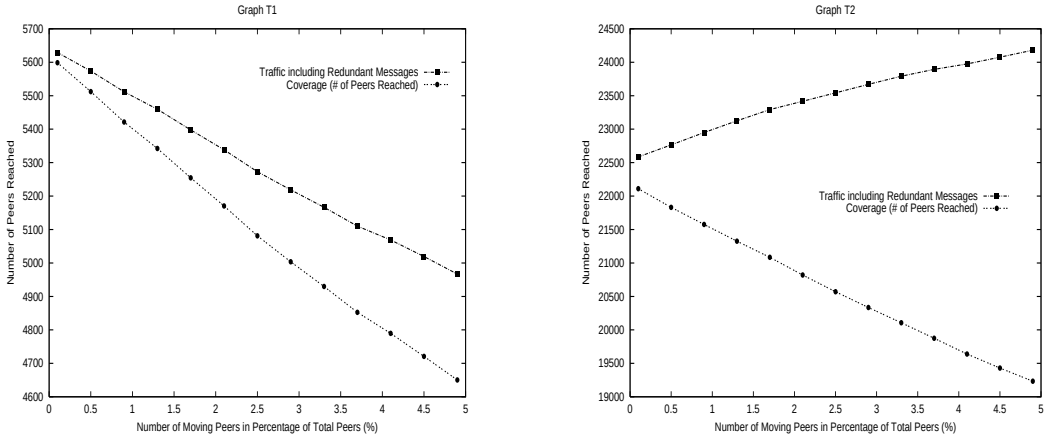


Fig. 3. Average coverage and corresponding traffi c for trail broadcasts with various number of moving peers for connection graphs T1 and T2. For comparison, the average coverage and traffi c of a standard fboding are 5,925 peers and 10,719 messages for T1, 22,426 peers and 84,136 messages for T2.

be increased with the increasing of moving peers. If we use ratio of coverage and traffic to characterize traffic efficiency, the traffic efficiency is 98.6%, 97.0% and 98.1%, respectively for graph T1, and is 93.6%, 89.7% and 84.9%, respectively for graph T2, when 1%, 2%, and 3% of total peers are moved. Though the efficiency is decreased with the increase of moving peers, they are much better than the efficiency for a standard flooding, namely 55.3% for T1 and 26.7% for T2. We obseverd that the efficiency of T2 is lowered more than that for T1, this is because of its high average degree and more trail links added for lost and new peers.

In summary, trail broadcast is robust in retaining its significant performance merits over a standard flooding even with a largely transient P2P system, and significantly reduces broadcast traffic.

### C. The Performance of FloodTrail for Web Content Requests

We ran the Web content request streams R1 and R2 on connection graphs T1 and T2, respectively. Because the reported median session duration for P2P systems is one hour, and both R1 and R2 are one-hour traces, we assume that during each request stream, a peer would arrive once and depart once. We also assume the arrivals and departures are randomly distributed along the one-hour period. We define moving rate as the number of peers moved in a second in percentage of the number of total peers. As before we add a peer into the network once a peer departs to maintain a graph of fixed size. A peer moving includes a peer departure and a peer arrival. So we estimate the moving ratio is $1/3600 \times 100\% = 0.028\%$ for a second in the scenario. To stress the robustness of the FloodTrail technique, we also tested moving ratios of 0.05%, 0.1%, 0.15%, 0.2%, and 0.25% for a second. We selected three representative window times: 5, 10 and 15 seconds. The experimental results are depicted in Table I.

In the table, we observed that (1) Increasing window times could increase hit ratios and the use of trail floodings, which could provide a smaller coverage with peer moving in/out. However, the reduction of coverage is trivial. Meanwhile the reduction of traffic is significant. For example, the average coverage of R1 on T1 is reduced by only 0.4%, but its traffic is reduced by 11.4% when the window time is increased from 5 to 15 seconds for the moving rate of 0.028%. (2) The coverage reduction due to the increasing of moving rates is moderate, which shows the robustness of the FloodTrail technique in the face of the transiency of ad hoc systems. (3) The coverages are very close to those of the 7-hop standard floodings, which are 5,925 and 22,426 in graphs T1 and T2, respectively. However, the traffic is significantly reduced, and the reduction for high degree graphs is even large. For example, the average traffic of 7-hop standard flooding is 10,719 and 84,136 in graphs T1 and T2, respectively. With the moving rate of 0.028% and the window time of 10 seconds, traffic is reduced by 27.1% for R1 on T1, and by 53.2% for R2 on T2.

### D. Space Overhead of Trail Flooding

While trail flooding can greatly reduce traffic overhead caused by redundant messages, it needs peers in the system to store trail information. Actually the additional space overhead distributed among all peers is small and acceptable in practice. If one bit is used to indicate whether a link is on the trail of a root peer, it takes only 4 bytes for a peer with a degree of 32 to remember trail information for a root peer. For a system whose 7-hop flooding coverage is around 40K peers, the peer takes only about 160KByte to remember trail information for peers in its neighborhood. In fact, we expect this space overhead is comparable with that used for message ID in a standard flooding, which practically imposes little burden on P2P system participants.

## V. Conclusion

By utilizing the trail a standard flooding leaves, and carefully managing the trail in the face of constant peer arrivals and departures, we designed the FloodTrail technique to reduce the large overhead of popular flooding searches. We evaluated it through simulations with real-life topology and request stream

TABLE I

COVERAGE AND TRAFFIC VARIANCE

| | Window Time(s) | Hit Rate | 0.028% | 0.05% | 0.10% | 0.15% | 0.20% | 0.25% |
|---|---|---|---|---|---|---|---|---|
| T1+R1 | 5 | 60.4% | *5916/7817* | 5910/7813 | 5894/7804 | 5876/7792 | 5860/7781 | 5843/7770 |
| T1+R1 | 10 | 73.2% | 5903/7187 | 5886/7176 | 5844/7149 | 5847/7978 | 5763/7096 | 5720/7066 |
| T1+R1 | 15 | 78.4% | 5890/6924 | 5859/6903 | 5794/6860 | 5832/8304 | 5661/6771 | 5592/6722 |
| T2+R2 | 5 | 59.0% | 22409/47689 | 22388/47713 | 22339/47759 | 22298/47808 | 22258/47860 | 22217/47908 |
| T2+R2 | 10 | 72.4% | 22374/39404 | 22322/39454 | 22220/39575 | 22211/39624 | 22009/39797 | 21902/39898 |
| T2+R2 | 15 | 77.9% | 22336/36012 | 22261/36097 | 22097/36286 | 22086/36365 | 21755/36615 | 21589/36763 |

Note: Average coverage in number of peers and average traffic in number of forwarded messages of a query in FloodTrail for various peer moving rates, which are shown as a pair of numbers in that order. For example, "5916/7817" means the average coverage is 5916 peers and the average traffic is 7817 messages for request stream R1 on graph T1 when there are 0.028% of total peers moving in a second and the window time is 5 seconds.

traces. Simulation results show that the FloodTrail technique is effective and consistent to significantly reduce flooding traffic, while keeping the merits of standard flooding such as large coverage and low response time across a large range of moving rates and window times. We believe FloodTrail can effectively serve as a basic flooding technique for file search in unstructured P2P systems.

Currently, we are investigating the feasibility of adaptively customizing window time to each peer's request behavior pattern, as well as its performance potentials.

## REFERENCES


[1] S. Jiang and X. Zhang, "LighFlood: an efficient floding scheme for file search in unstructured peer-to-peer systems?, *Proceedings of 2003 International Conference on Parallel Processing, (ICPP'03)*, October, 2003.

[2] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer networks", *Proceedings of the 16th ACM International Conference on Supercomputing*, June 2002

[3] "Why Gnutella Can't Scale. No, Really", *http://www.darkridge.com/ jpr5/doc/gnutella.html*

[4] M. Ripeanu and I. Foster, "Mapping Gnutella Network" *Proceedings of first International Workshop on Peer-to-Peer Systems (IPTPS'02)*, March 2002.

[5] J. Ritter, "Why Gnutella Can't Scale. No, Really.", http://www.monkey.org/ dugsong/mirror/gnutella.htm, February, 2001

[6] M. K. Ramanathan, V. Kalogeraki, J. Pruyne, "Finding Good Peers in Peer-to-Peer Networks", *Symposium of International Parallel and Distributed Processing*, April, 2002

[7] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-Peer file Sharing Systems", *Proceedings of Multimedia Computing and Networking*, 2002.

[8] K. Scipanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems", *Proceedings of IEEE INFOCOM 2003*.

[9] B. Yang, H. Garcia-Molina, "Designing a Super-peer Network", *Proceedings of the 19th International Conference on Data Engineering*, March 2003.

[10] B. Yang, H. Garcia-Molina,"Improving Search in Peer-to-Peer Systems", *Proceedings of the 22nd International Conference on Distributed Computing Systems*, , July 2002.

[11] Clip2.com, "Clip2 Gnutella crawl files"

[12] J. Meadows, "boeing proxy logs," Available at ftp://reaserachs.cc.vt.edu/pub/boeing/, March 1999.