

# A Feedback-based Approach to Reduce Duplicate Messages in Unstructured Peer-to-Peer Networks

Charis Papadakis<sup>1</sup>, Paraskevi Fragopoulou<sup>1</sup>, Elias Athanasopoulos<sup>1</sup>, Marios Dikaiakos<sup>2</sup>, Alexandros Labrinidis<sup>3</sup> and Evangelos Markatos<sup>1</sup>

<sup>1</sup> Institute of Computer Science, Foundation for Research and Technology-Hellas  
P.O. Box 1385, 71 110 Heraklion-Crete, Greece  
{adanar, fragopou, elathan, markatos}@ics.forth.gr

<sup>2</sup> Department of Computer Science, University of Cyprus, P.O. Box 537, CY-1678 Nicosia, Cyprus  
[mdd@ucy.ac.cy](mailto:mdd@ucy.ac.cy)

<sup>3</sup> Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA  
[labrinid@cs.pitt.edu](mailto:labrinid@cs.pitt.edu)

**Abstract.** Unstructured P2P systems have used flooding as their prevailing resource location method. Flooding dictates that each node should forward each incoming query messages to all of its neighbours until the query propagates up to a predefined maximum number of hops away from its origin. Although this algorithm has excellent response time and is very simple to implement, it creates a large volume of unnecessary traffic in today's Internet because each node may receive the same queries several times through different paths. In this paper, we propose an innovative technique, namely the *feedback-based approach* that aims to improve the scalability of flooding. The main idea behind our feedback-based algorithm is to monitor the number of duplicate messages transmitted over each network connection, and to forward query messages preferably over connections which do not produce excessive number of duplicates. During an initial and relatively short warm-up phase, a feedback message is returned for each duplicate message to the upstream node. Following the warm-up phase, each node decides as to whether to forward incoming query messages on each of its outgoing connections based on whether the percentage of duplicates on that connection during the warm-up phase does not exceed some predefined threshold. Through extensive simulation we show that this algorithm exhibits significant reduction of traffic in random and small-world graphs, the two most common types of graph that have been studied in the context of P2P systems, while conserving network coverage.

## 1 Introduction

In unstructured P2P networks, such as Gnutella and KaZaA, each node is directly connected to a small set of other nodes, called neighbors. Most of today's commercial P2P systems are unstructured and rely on random overlay networks [7,9]. Unstructured P2P systems have used flooding as their prevailing resource location method [7,9]. A node looking for a file issues a query which is broadcasted in the network. An important parameter in the flooding algorithm is the Time-To-Live or TTL. The TTL indicates the number of hops away from its source a query should propagate. The node that initiates the flooding sets the query's TTL to a small positive integer, smaller than the diameter of the network. Each receiving node decreases by one the query TTL value before broadcasting it to its neighbors. The query propagation terminates when its TTL reaches zero.

The basic problem with the flooding mechanism is that it creates a large volume of unnecessary traffic in the network mainly because a node may receive the same queries multiple times through different paths. The reason behind the duplicate messages is the existence of cycles in the underlying network topology. Duplicates constitute a large percentage of the total number of messages generated during flooding. In a network of  $N$  nodes and average degree  $d$  and for TTL value equal to the diameter of the graph, there are  $N(d-2)$  duplicate messages for a single query while only  $N-1$  messages are needed to reach all network nodes. The TTL was incorporated in the flooding algorithm in order to reduce the number of messages produced thus reducing the overall network traffic. Since the paths traversed by the flooding messages are short, there is a small probability that those paths will form cycles and thus generate duplicates. However, as we will see below, even this observation is not valid for small-world graphs. Furthermore, a small TTL value can reduce the *network coverage* defined as the percentage of network nodes that receive a query.

In an effort to alleviate the large volumes of unnecessary traffic produced during flooding several variations have been proposed in the literature [12]. Most of these rely on randomly or selectively propagating the query messages to a small number of each node's neighbours. The neighbour selection criteria is the number of responses received, the node capacity, or the link latency. Although these methods succeed in reducing excessive network traffic, they usually incur significant loss in network coverage, meaning that only a small part of the network's nodes are queried, thus a much smaller number of query answers are returned to the requesting node. This can be a problem especially when the search targets rare items for which often no response is returned. Other search methods such as random walkers or multiple random walkers suffer from slow response time.

Aiming to alleviate the excessive network traffic problem while at the same time maintain high network coverage, in this paper, we devise an innovative technique, the feedback-based algorithm, that attacks the problem by monitoring the number of duplicates on each network connection and trying to forward queries over connections that do not produce an excessive number of duplicates. During an initial and relatively short warm-up phase, a feedback is returned for each duplicate that is encountered on an edge to the upstream node. Following the warm-up phase each node decides to forward incoming query messages on each of its incident edges based on whether the percentage of duplicates on that edge during the warm-up phase does not exceed some predefined threshold value. We show through extensive simulation, for different values of the parameters involved, that this algorithm is very efficient in terms of traffic reduction in random and small-world graphs, the two most common types of graph that have been studied in the context of P2P systems, while the algorithm exhibits minor loss in network coverage. Furthermore, a restricted version of the algorithm which gives the best results does not require any protocol modification.

The remainder of this paper is organized as follows: Following the related work section, the feedback-based algorithm is presented in Section 3. The two most common types of graphs that were studied in the context of P2P systems, and on which we conducted our experiments, are presented in Section 4. The simulation details and the experimental results on static graphs are presented in Section 5. Finally, the algorithm's behavior on dynamic graphs, assuming that nodes can leave the network and new nodes can enter at any time, is presented in Section 6. We conclude in Section 7 with a summary of the results.

## 2 Related Work

Many algorithms have been proposed in the literature to alleviate the excessive traffic problem and to deal with the traffic/coverage trade-off [12]. One of the first alternatives to be proposed was *random walk*. Each node forwards each query it receives to a single neighboring node chosen at random. In this case the TTL parameter designates the number of hops the walker should propagate. Random walks produce very little traffic, just one query message per visited node, but reduce considerably network coverage and have long response time. As an alternative *multiple random walks* have been proposed. The node that originates the query forwards it to  $k$  of its neighbors. Each node receiving an incoming query transmits it to a single randomly chosen neighbor. Although compared to the single random walk this method has better behavior, it still suffers from low network coverage and slow response time. Hybrid methods that combine flooding and random walks have been proposed in [5].

In another family of proposed algorithms query messages are forwarded not randomly but rather selectively to part of a node's neighbors based on some criteria or statistical information. For example, each node selects the first  $k$  neighbors that returned the most query responses, or the  $k$  highest capacity nodes, or the  $k$  connections with the smallest latency to forward new queries [6]. A somewhat different approach named *forwarding indices* [2] builds a structure that resembles a routing table at each node. This structure stores the number of responses returned through each neighbor on each one of a pre-selected list of topics. Other techniques include query caching, and the incorporation of semantic information in the network [3,10,14].

The specific problem we deal with in this paper, namely the problem of duplicate messages, has been identified and some results appear in the literature. In [13] a randomized and a selective approach is adopted and each query message is sent to a portion of a node's neighbors. The algorithm is shown to reduce the number of duplicates and to maintain network coverage. The performance of the algorithm is demonstrated on graphs of limited size. In another effort to reduce the excessive traffic in flooding, Gkatsidis and Mihail [5] proposed to direct messages along edges which are parts of shortest paths. They rely on the use of PING and PONG messages to find the edges that lie on shortest paths. However, due to PONG caching is this not a reliable technique. Furthermore, their algorithm degenerates to simple flooding for random graphs, meaning that in this case no duplicate messages are eliminated. Finally, in [8] the authors propose to construct a shortest paths spanning tree rooted at each

network node. However, this algorithm is not very scalable since the state each network node has to keep is in the order of  $O(Nd)$ , where  $N$  is the number of network nodes and  $d$  its average degree.

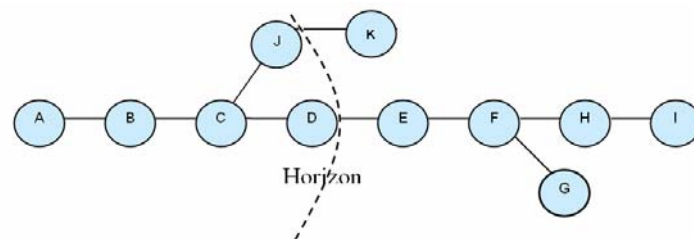
### 3 The Feedback-based Algorithm

The basic idea of the feedback based algorithm is to identify edges on which an excessive number of duplicates are produced and to avoid forwarding query messages over these edges. In the algorithm's warm-up phase, during which flooding is used, a feedback message is returned to the upstream node for each duplicate message. The objective of the algorithm is to count the number of duplicates produced on each edge during this phase and subsequently, during the execution phase, to use this count to decide whether to forward a query message over an edge or not.

In a static graph, a query message transmitted over an edge is a duplicate if this edge is not on the shortest path from the origin to the downstream node. One of the key points in the feedback-based algorithm is the following: Each network node  $A$  forms groups of the other nodes, and a different count is kept on each one of  $A$ 's incident edges for duplicate messages originating at nodes of each different group. The objective is for each node  $A$  to group together the other nodes so that messages originating at nodes of the same group either produce many duplicates or few duplicates on each one of  $A$ 's incident edges. An incident edge of some node  $A$  that produces only a few duplicates for messages originating at nodes of a group belongs to many shortest paths connecting nodes of this group to the downstream node. An incident edge of node  $A$  that produces many duplicates for messages originating at nodes of a group belongs to few shortest paths connecting nodes of this group to the downstream node. Notice that if all duplicate messages produced on an edge were counted together (independent of their origin), then the algorithm would be inconclusive. In this case the duplicate count on all edges would be the almost the same since each node would receive the same query though all of its incident edges. The criteria used by each node to group together the other nodes are critical for the algorithm's performance and the intuition for their choice is explained below.

A sketch of the feedback-based algorithm is the following:

- Each node  $A$  groups together the rest of the nodes according to some criteria.
- During the warm-up phase, each node  $A$  keeps a count of the number of duplicates on each of its incident edges, originating at nodes of each different group.
- Subsequently, during the execution phase, messages originating at nodes of a group are forwarded over an incident edge  $e$  of node  $A$ , if the percentage of duplicates for this group on edge  $e$  during the warm-up phase is below a predefined threshold value.



**Fig. 1.** Illustration of the horizon criterion for node  $A$  and for horizon value 3

Two different grouping criteria, namely, the hops, the horizon, and a combination of them horizon+hops are used that lead to three variations of the feedback-based algorithm.

- **Hops criterion:** Each node  $A$  keeps a different count on each of its incident edges for duplicates originating  $k$  hops away ( $k$  ranges from 1 up to the graph diameter). The intuition for this choice is that, as we will see below, in random graphs small hops produce few duplicates and large hops produce mostly duplicates. Thus, messages originating at close by nodes are most probably not duplicates while most messages originating at distant nodes are duplicates. In order for this grouping criterion to work each query message should store the number of hops traversed so far.
- **Horizon criterion:** The horizon is a small integer, smaller than the diameter of the graph. A node is in the horizon of some node  $A$  if its distance in hops from  $A$  is less than the horizon value, while all other nodes are outside  $A$ 's horizon, Fig. 1. For each node inside  $A$ 's horizon a different count is kept by  $A$  on each of its incident edges. Duplicate messages originating at nodes outside  $A$ 's horizon

are added up to the count of their entry node in A's horizon. For example, in Fig. 1, duplicates produced by queries originating at node K are added up to the counters kept for node J, while duplicates produced by queries originating at nodes E,F,G,H,I are added up to the counters kept for node D. The intuition for the choice of this criterion is that shortest paths differ in the first hops and when they meet they follow a common route. For this criterion to be effective a message should store the identities of the last k nodes visited, where k is the horizon value.

- **Horizon+Hops criterion:** This criterion combines the two previous. Duplicates are counted separately on each one of A's incident edges for each node in A's horizon. Nodes outside A's horizon are grouped together according (1) to their distance in hops from A and (2) to the entry node of their messages in A's horizon.

Three variations of the feedback-based algorithm are presented based on the grouping criteria used. The algorithm using the hops criterion is show below:

### Feedback-based algorithm using the Hops criterion

#### 1. Warm-up phase

- Each incoming non-duplicate query message is forwarded to all neighbors except the upstream one.
- For each incoming duplicate query message received, a duplicate feedback is returned to the upstream node.
- Each node A, for each incident edge e, counts the percentage of duplicate feedbacks produced on edge e for all queries messages originating k hops away. Let us denote this count by  $C_{e,k}$

#### 2. Execution phase

- Each node A forwards an incoming non-duplicate query message that originates k hops away over its incident edges e if the count  $C_{e,k}$  does not exceed a predefined threshold.

For the hops criterion to work each query message needs to store the number of hops traversed so far. The groups formed by node A in the graph of Fig. 1 according to the hops criterion are shown in Table 1.

The algorithm using the horizon criterion is shown below:

### Feedback-based algorithm using the Horizon criterion

#### 1. Warm-up phase

- & b. Same as in Hops criterion
- Each node A, for each incident edge e, counts the percentage of duplicates produced on edge e for all query messages originating at a node B inside the horizon, or entered the horizon at node B. Let us denote this count by  $C_{e,B}$ .

#### 2. Execution phase

- Each node A forwards an incoming non-duplicate query message that originates at a node B inside the horizon, or which entered the horizon at node B over its incident edges e if the count  $C_{e,B}$  does not exceed a predefined threshold value.

For the horizon criterion to work each query message needs to store the identity of the last k nodes visited. The groups formed by node A in the graph of Fig. 1 according to the horizon criterion are shown in Table 2.

**Table 1.** Groups of the Hops criterion based on the example of example of Fig. 1

Hops	1	2	3	4	5	6	7
Groups of nodes formed by node A	B	C	D, J	E, K	F	G, H	I

**Table 2.** Groups of the Horizon criterion based on the example of example of Fig. 1

Node in A's horizon	B	C	D	J
Groups of nodes formed by node A	B	C	D, E, F, G, H, I	J, K

The algorithm using the combination of the two criteria described above, namely the horizon+hops, is shown below. For this criterion each message should store the number of hops traversed and the identity of the last  $k$  nodes visited.

### Feedback-based algorithm using the Horizon+Hops criterion

#### 1. Warm-up phase

- a. & b. Same as in Hops criterion
- c. Each node  $A$ , for each incident edge  $e$ , counts the percentage of duplicates produced on edge  $e$  for all queries messages originating at a node  $B$  inside  $A$ 's horizon, or which entered  $A$ 's horizon at node  $B$  and originated  $k$  hops away. Let us denote this count by  $C_{e,B,k}$ .

#### 2. Execution phase

- a. Each node  $A$  forwards an incoming non-duplicate query message originating at some node  $B$  inside  $A$ 's horizon, or which entered  $A$ 's horizon at node  $B$  and originated  $k$  hops away, over its incident edges  $e$  if the count  $C_{e,B,k}$  does not exceed a predefined threshold.

The groups formed by node  $A$  in Fig. 1 for the horizon+hops criterion are shown in Table 3.

We should emphasize that in order to avoid increasing the network traffic due to the feedback messages, a single collective message is returned to each upstream node at the end of the warm-up phase.

Table 3. Groups of the Horizon+Hops criterion based on the example of example of Fig. 1

Node in A's horizon	B	C	D					J	
Hops	1	2	3	4	5	6	7	3	4
Groups of nodes formed by node A	B	C	D	E	F	G, H	I	J	K

## 4 Random vs. Small-World Graphs

Two types of graphs have been mainly studied in the context of P2P systems. The first is random graphs which constitute the underline topology in today's commercial P2P systems [7,9]. The second type is small-world graphs which emerged in the modelling of social networks [4]. It has been demonstrated that P2P resource location algorithms could benefit from small-world properties. If the benefit proves to be substantial then the node connection protocol in P2P systems could be modified so that small-world properties are intentionally incorporated in their network topologies.

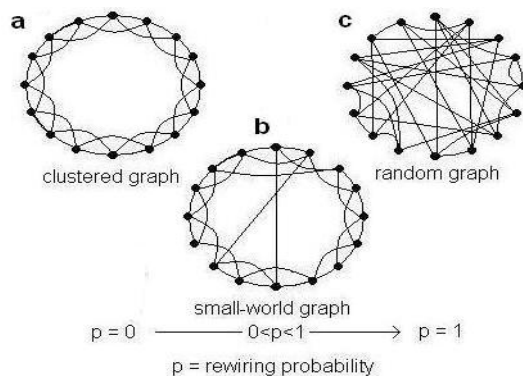


Fig. 2. (a) A clustered graph with no rewired edges (rewiring probability  $p=0$ ). (b) A small-world graph produced from the clustered graph with a small rewiring probability (c) A random graph produced if every edge is rewired to a random node (rewiring probability  $p=1$ )

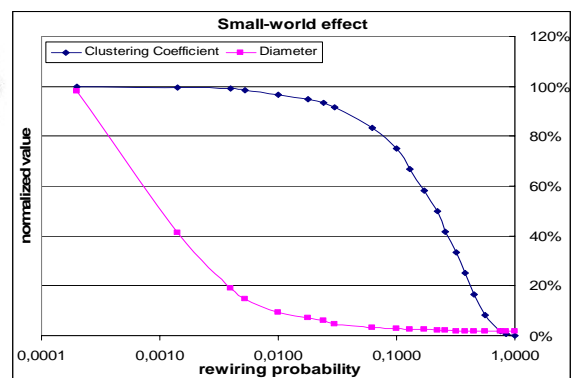


Fig. 3. By rewiring a few edges of the initial clustered graph to random nodes the *average diameter* of the graph is greatly reduced, without significantly affecting the *clustering coefficient*

In *random graphs* each node is randomly connected to a number of other nodes equal to its degree. Random graphs have small diameter and average diameter. The *diameter* of a graph is the length

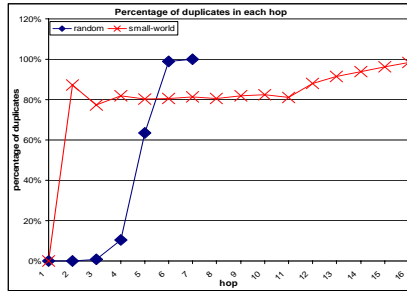
(number of hops for un-weighted graphs) of the longest among the shortest paths that connect any pair of nodes. The *average diameter* of a graph is the average of all longest shortest paths from any node to any other node.

A *clustered graph* is a graph that contains densely connected “neighborhoods” of nodes, while nodes that lie in different neighborhoods are more loosely connected. A metric that captures the degree of clustering that graphs exhibit is the clustering coefficient. Given a graph  $G$ , the *clustering coefficient of a node*  $A$  of  $G$  is defined as the ratio of the number of edges that exist between the neighbors of  $A$  over the maximum number of edges that can exist between its neighbors (which equals  $k(k-1)$  for  $k$  neighbors). The *clustering coefficient of a graph*  $G$  is the average of the clustering coefficients of all its nodes. Clustered graphs have, in general, higher diameter and average diameter than their random counterparts with about the same number of nodes and degree.

A small-world graph is a graph with high clustering coefficient yet low average diameter. The small-world graphs we use in our experiments are constructed according to the Strogatz-Watts model. Initially, a regular, clustered graph of  $N$  nodes is constructed as follows: each node is assigned a unique identifier from 0 to  $N-1$ . Two nodes are connected if their identity difference is less than or equal to  $k$  (in mod  $N$  arithmetic). In Fig. 2(a) such a graph is shown for  $N=16$  and  $k=2$ . Subsequently, each edge of the graph is rewired to a random node according to a given rewiring probability  $p$ . If the rewiring probability of edges is relatively small, a small-world graph is produced (high clustering coefficient and small average diameter), as shown in Fig. 2(b). As the rewiring probability increases the graph becomes more random (the clustering coefficient decreases). For rewiring probability  $p=1$ , all graph edges are rewired to random nodes, and this results to a random graph, Fig. 2(c). In Fig. 3, we can see how the clustering coefficient and the average diameter of graphs vary as the rewiring probability  $p$  increases. Small-world graphs are somewhere in the middle of the  $x$  axis ( $p=0.01$ ).

The clustering coefficient of each graph is normalized with the respect to the maximum clustering coefficient of a graph with the same number of nodes and average degree. In what follows, when we refer to the clustering coefficient of a graph with  $N$  nodes and average degree  $d$ , denoted by  $CC$ , we refer to the percentage of its clustering coefficient over the maximum clustering coefficient of a graph with the same number of nodes and average degree. The maximum clustering coefficient of a graph with  $N$  nodes and average degree  $d$  is the clustering coefficient of the clustered graph defined according to the Strogatz-Watts model, Fig. 2(a), before any edge rewiring takes place.

Fig. 4 shows the percentage of duplicate messages generated per hop over the messages generated on that hop on a random and on a small-world graph of 2000 nodes and average degree 6. We can see from this figure that in a random graph there are very few duplicate messages in the first few hops (1-4), while almost all messages in the last hops (6-7) are duplicates. On the contrary, in small-world graphs duplicate messages appear from the first hops and their percentage (over the total number of messages per hop) remains almost constant till the last hops.



**Fig. 4.** Percentage of messages generated per *hop*, which are *duplicates*, in random and small-world graphs. In small-world graphs the percentage of duplicates in hops 2 to 11 is almost constant, while in random graphs, in small hops there are no duplicates and in large hops almost all messages are duplicates

## 5 Experimental Results on Static Graphs

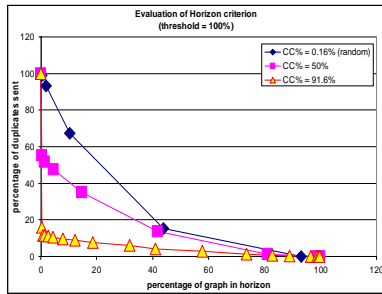
The simulation was performed using **sP2Ps** (simple P2P simulator) developed at our lab. The experiments were conducted on graphs with 2000 nodes and average degree 6. The clustering coefficient ranged from 0.0001 to 0.6, which is the maximum clustering coefficient of a graph with  $N=2000$  and  $d=6$ . We shall refer to  $CC$  values from now on, as percentages of that max value. We conducted experiments for different values of the algorithm’s parameters. The horizon value varied from 0 (were practically the horizon criterion is not used) up to the diameter of the graph. Furthermore,

we used two different threshold values, namely 75% and 100%, to select the connections over which messages are forwarded. For example a threshold of 75% indicates that if the percentage of duplicates on an edge  $e$  during the warm up phase exceeds 75% for messages originated at the nodes of a group, in the execution phase no query message from this group is forwarded over edge  $e$ . The TTL value is set to the diameter of the graph.

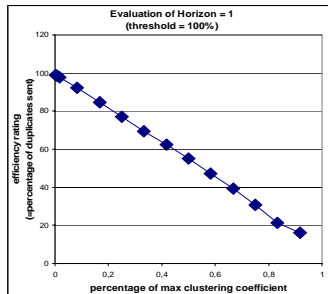
The efficiency of our algorithm is evaluated based on two metrics, firstly the percentage of duplicates sent by the algorithm, in relation to the naive flooding and secondly the network coverage (defined as the percentage of network nodes reached by the query). Thus, the lower the duplicates percentage and the higher the coverage percentage, the better. Notice that a threshold value of 100% indicates that messages originating at the nodes of a group are not forwarded only over edges that produce exclusively (100%) duplicates for all nodes of that group during the warm-up phase. In this case we do not experience any loss in network coverage but the efficiency of the algorithm in duplicate elimination could be limited. In all experiments on static graphs, the warm-up phase included one flooding from each node. In the execution phase, during which the feedback-based algorithm is applied, again one flooding is performed from each node in order to gather the results of the simulation experiment.

In Figs 5-10 we can see the experimental results for the feedback-based algorithm with the horizon criterion. In Fig. 5 we can see the percentage of duplicates produced as a function of the percentage of graph nodes in the horizon for three graphs (random with  $CC=0.16$ , clustered with  $CC=50$ , and small-world with  $CC=91.6$ ) and for threshold value 100%, which means that there is no loss in network coverage. We can deduce from this figure that the efficiency of this algorithm is high for clustered graphs and increases with the percentage of graph nodes in the horizon. Notice that in clustered graphs, with a small horizon value a larger percentage of the graph is in the horizon as compared to random graphs. In Fig. 6 we plot the percentage of duplicates produced by the algorithm as a function of the clustering coefficient for horizon value 1 and threshold 100%. We can see that even for such a small horizon value the efficiency of the algorithm increases linearly with the clustering coefficient of the graph. We can thus conclude that the feedback-based algorithm with the horizon criterion is efficient for clustered and small-world graphs.

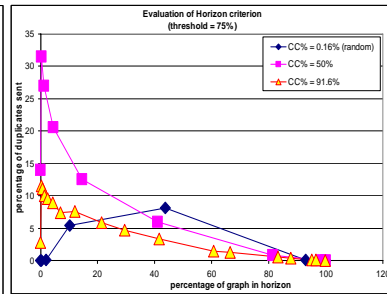
Even if the percentage of graph nodes in the horizon decreases, in case the graph size increases and the horizon value remains constant, the efficiency of the algorithm will remain unchanged, because in clustered graphs the clustering coefficient does change significantly with the graph size. Thus, the horizon criterion is scalable for clustered graphs. In contrast, in random graph, in order to maintain the same efficiency as the graph size increases, one would need to increase the horizon value, in order to maintain the same percentage of graph nodes in the horizon. Thus the horizon criterion is not scalable on random graphs.



**Fig. 5.** Percentage of *duplicates* as a function of the percentage of graph nodes in the horizon for three graphs with clustering coefficients 0.16, 50, and 91.6, and threshold value 100%



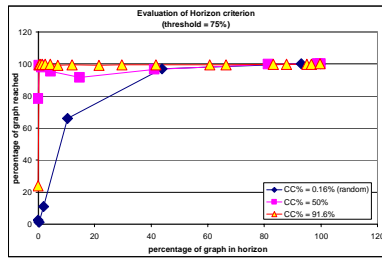
**Fig. 6.** Percentage of *duplicates* as a function of the clustering coefficient for horizon value 1 and threshold value 100%



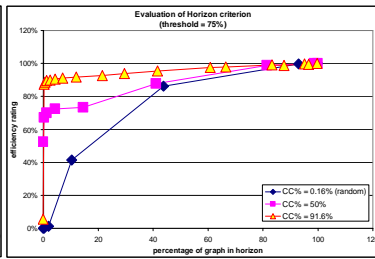
**Fig. 7.** Percentage of *duplicates* as a function of the percentage of graph nodes in the horizon for three graphs with different clustering coefficients (0.16, 50, and 91.6) and threshold value 75%

Figs 7-10 show the efficiency of the algorithm with the horizon criterion in duplicate elimination for threshold 75%. In Figs 7 and 8 we can see that the algorithm is very efficient on clustered graphs. From the same figures we can see that with this threshold value in random graphs ( $CC=0.16$ ) most duplicate messages are eliminated but there is loss in network coverage. Thus, even if we lower the threshold value, the horizon criterion does not work well for random graphs. The algorithm's behavior is summarized in Fig. 9, where duplicate elimination, denoted by  $D$ , and network coverage, denoted by  $C$ , are combined into one simple metric, defined as  $C^2D$ .

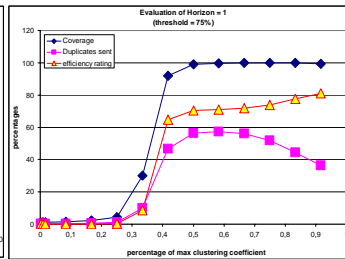
In Fig. 10 we can see again the efficiency of the algorithm for horizon value 1 (as in Fig. 6) but for a threshold of 75%. Notice that the algorithm's efficiency is not linear to the percentage of the clustering coefficient of the graph. This arises because the threshold value of 75% is not necessarily the best choice for any clustering coefficient.



**Fig. 8.** Network coverage as a function of the percentage of graph nodes in the horizon for three graphs with clustering coefficients 0.16, 50, and 91.6 and threshold 75%



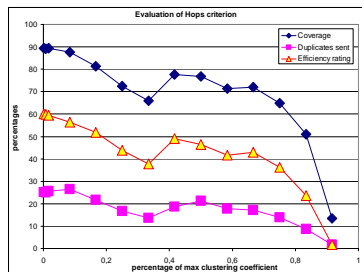
**Fig. 9.** Efficiency of the feedback based algorithm as a function of the percentage of graph nodes in the horizon for three graphs with clustering coefficients 0.16, 50, and 91.6 and threshold 75%



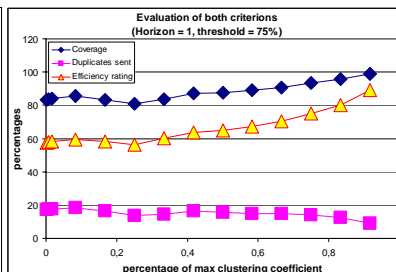
**Fig. 10.** Network coverage, percentage of duplicates, and efficiency as a function of the clustering coefficient for horizon value 1 and threshold 75%

In Fig. 11 we can see the experimental results for the algorithm with the hops criterion for a graph with 2000 nodes and average degree 6 while varying the clustering coefficient. We can see in this figure that the hops criterion is very efficient in duplicate elimination, while maintaining high network coverage, for graphs with small clustering coefficient. This means that this criterion exhibits very good behaviour on random graphs. As the clustering coefficient increases the performance of the algorithm with the hops criterion decreases. This behaviour can be easily explained from Fig. 4, where the percentage of duplicates per hop is plotted for random and small-world graphs. We can see from this figure that in random graphs, the small hops produce very few duplicates, while large hops produce too many. Thus, based on the hops criterion only, we were able to eliminate a large percentage of duplicates without greatly sacrificing network coverage.

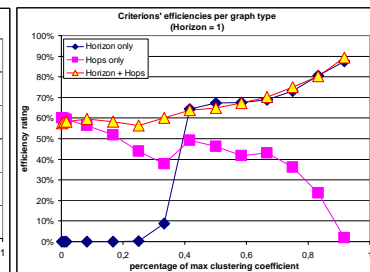
As mentioned before, the hops criterion works better for random graphs. In case the graph size increases, the number of hops also increases (recall that the diameter of a random graph with  $N$  nodes and average degree  $d$  is  $\log(N)/\log(d)$ ). Thus, the hops criterion is scalable on random graphs.



**Fig. 11.** Network coverage, percentage of duplicates, and efficiency of the algorithm with the hops criterion as a function of the clustering coefficient



**Fig. 12.** Network coverage, percentage of duplicates, and efficiency of the algorithm with the horizon+hops criterion as a function of the clustering coefficient



**Fig. 13.** Efficiency of algorithms with the horizon, hops, and horizon+hops criteria as a function of the clustering coefficient and for horizon value 1

In Fig. 12, we see the efficiency of the algorithm for the horizon+hops criterion. As we can see from this figure this combination of criteria constitutes the feedback based algorithm efficient in graphs with all clustering coefficients, random and small-world. In Fig. 12, three different metrics are plotted, the network coverage, the percentage of duplicates, and the efficiency as a function of the clustering coefficient of the graph. We can see that for any clustering coefficient network coverage is always above 80%, while the percentage of duplicate messages not eliminated is always less than 20%. This behavior is achieved for random and small-world graphs for horizon value of only 1. Thus the horizon+hops criterion is scalable on all types of graphs.

In Fig. 13 we compare the efficiencies of the hops, horizon, and horizon+hops and we see that their combination, horizon+hops works better than each criterion separately.

## 6 Experimental Results on Dynamic Graphs

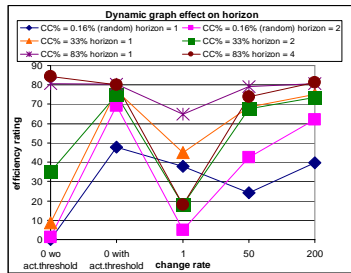
In what follows, we introduce dynamic changes to the graph, meaning that a graph node can leave and some other node can enter the graph, and we monitor how these changes influence the algorithm's efficiency. We introduced a new parameter to our experiments in order to capture the rate of graph change. This parameter measures in query-floods the lifetime of a node in the graph. A graph rate change of  $r$  means that each node will initiate, on the average,  $r$  query-floods before leaving the network. Insertion of new nodes is performed so as to preserve the clustering coefficient of the graph.

We also introduce a dynamic way to determine when the warm-up phase can terminate, meaning that we have collected enough measurements. The warm-up phase for a group of nodes terminates after the percentage of duplicates seen on an edge for messages originating at nodes of the group stops to oscillate significantly. More specifically, the warm-up phase terminates on an edge for a group of nodes, if in each of the last 20 rounds the change in the count (percentage of the number of duplicates seen on that edge for messages originating at nodes of the that group) was smaller than 2% and the total change over the last 20 rounds was smaller than 1%.

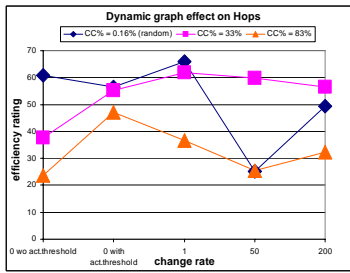
We perform experiments for random graphs and for small-world graphs with clustering coefficient  $CC=33$  and  $CC=84$ . For each of these graphs, the value of the change rate equals 0 (static graph), 1, 50, and 200. A change rate of 200 indicates that each node will make 200 query-floods before leaving the network, which is a reasonable assumption for Gnutella 2 [7]. This is because each Ultrapeer contains, on the average, 30 leaves. A leaf node has in general much smaller average lifetime than an Ultrapeer, which means that each Ultrapeer will "see" more than 30 unique leaves in its lifetime. If we assume that each leaf node will send one query through the Ultrapeer, this explains the fact that real-world measures with an Ultrapeer show that each Ultrapeer sends about 100 queries per hour. For each of these graphs and change rates, we run experiments with the following Horizon values:

- Horizon values = {1|2} for random graphs and for small-world graphs with  $CC = 33$ .
- Horizon values = {1|4} for small-world graphs with  $CC = 84$ .

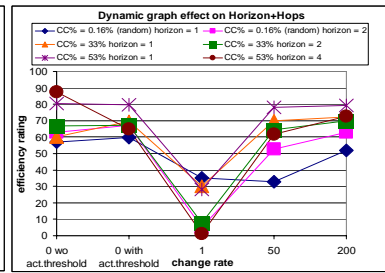
We performed two experiments with the same horizon value, one using the hops criterion and one without the hops criterion. The threshold value was set to 75%. Each experiment performed  $25 \times 2000$  floods. The difference between the values "0 wo act. threshold" and "0 with act. threshold" in the x axis in the figures indicates that in both cases the change rate is 0 (static graph), but in the first case, the numbers are taken from the experiments described in the previous section, while in the second case the activation threshold was used to terminate the warm-up phase. This enables us to clearly see the benefit of the activation threshold.



**Fig. 14.** Performance (efficiency) of the algorithm on a dynamic graph for the horizon criterion



**Fig. 15.** Performance (efficiency) of the algorithm on a dynamic graph for the hops criterion



**Fig. 16.** Performance (efficiency) of the algorithm on a dynamic graph for the horizon + hops criterion

Fig. 14 shows how the algorithm performs on dynamic graphs for the horizon criterion. We should first note that the use of the activation threshold increases the efficiency of the algorithm significantly. This happens because nodes gradually start eliminating traffic for certain groups of nodes instead of all of them starting eliminating duplicates for all groups simultaneously.

We can see that the efficiency of the algorithm decreases when the change rate is 1. The main reason for this is not that the measurements for each group quickly become stale, but rather because each node needs some warm-up period to learn the topology of the network. A certain amount of traffic needs to be "seen" by any node, to make the necessary measurements. If that time is a large fraction of the node's lifetime, it means that it will spend most of its time measuring instead of regulating traffic according to the measurements.

Finally and most importantly, we can see that the results for a change rate of 200 are the same as those of a change rate of 0 with activation threshold, which shows that, given that the warm-up phase is

shorter than the time during which the nodes use the algorithm (execution phase), the changes of the graph do not affect the algorithm's efficiency.

In Fig. 15 we can see that the activation threshold is beneficial to the algorithm with the hops criterion. Furthermore, from the same figure, it becomes clear that the efficiency of the feedback-based algorithm with the hops criterion is not greatly affected by the dynamic changes in the graph. We should however point out that it seems to lightly affect the efficiency of the algorithm in highly clustered graphs.

In Fig. 16 we finally see the efficiency of the algorithm for the horizon+hops criterion. We should notice again that the use of the activation threshold does not harm the algorithm, except in the case of the graph with high clustering coefficient and for a horizon value greater than 1. However, as we have seen before, there is not reason to use a horizon value larger than 1. Again, the change rate does not affect the measurements for groups of nodes, since the reason for the low efficiency at high change rates is the fact that the nodes spent most of their lifetime in the warm-up phase.

## 7 Conclusions

We presented the feedback-based algorithm, an innovative method which reduces significantly the number of duplicate messages produced by flooding while maintaining high network coverage. The algorithm monitors the percentage of duplicates on each connection during a warm-up phase, and directs traffic to connections that do not produce excessive number of duplicates during the execution phase. In order for this approach to work, each network node groups together the rest of the nodes according to some criteria, so that nodes that produce many duplicates on its incident edges are in different groups than those that produce only few duplicates. The efficiency of the algorithm was demonstrated through extensive simulation on random and small-world graphs, the two most common types of graphs that have been studied in the context of P2P systems. The experiments involved graphs of 2000 nodes. The feedback-based algorithm was shown to reduce to less than 20% the number of duplicates of flooding while conserving network coverage above 80%. The memory requirements in each node are much less compared to the algorithm that constructs shortest paths trees from each network node. The efficiency of our algorithm was demonstrated on static and dynamic graphs.

## References

1. Y. Chawathe, S. Ratnasamy, and L. Breslau. *Making Gnutella-like P2P Systems Scalable*. ACM SIGCOMM, 2003.
2. Crespo and H. Garcia-Molina. *Routing Indices for Peer-to-Peer Systems*. International Conference on Distributed Computing Systems, 2002.
3. Crespo and H. Garcia-Molina. *Semantic Overlay Networks for P2P Systems*. 2002.
4. Duncan, J. Watts, and S. H. Strongatz. *Collective Dynamics of Small-world Networks*. Nature, Vol. 393, pp. 440-442, 1998.
5. C. Gkantsidis, M. Mihail, and A. Saberi. *Hybrid Search Schemes for Unstructured Peer-to-Peer Networks*. IEEE INFOCOM, 2005.
6. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. *Search and Replication in Unstructured Peer-to-Peer Networks*. International ACM Conference on Supercomputing, 2002.
7. R. Manfredi and T. Klingberg. Gnutella 0.6 Specification, [http://rfc-gnutella.sourceforge.net/src/rfc-0\\_6-draft.html](http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html)
8. M. Ripenau, I. Foster, A. Iamnitchi, and A. Rogers. *UMM: A Dynamically Adaptive, Unstructured, Multicast Overlay*. In *Service Management and Self-Organization in IP-based Networks*, 2005, editors M. Bossardt, G. Carle, D. Hutchison, H. de Meer, and B. Plattner, Dagstuhl Seminar Proceedings.
9. Sharman Industries. Kazaa, <http://www.kazaa.com>
10. K. Sripanidkulchai, B. Maggs, and H. Zhang. *Efficient Content Location using Interest-Based Locality in Peer-to-Peer Systems*. IEEE INFOCOM, 2003.
11. D. Stutzbach and R. Rejaie. *Characterizing Today's Gnutella Topology*. Technical Report CIS-TR-04-02, Department of Computer Science, University of Oregon, Dec. 2004.
12. D. Tsoumakos and N. Roussopoulos. *A Comparison of Peer-to-Peer Search Methods*. International Workshop on the Web and Databases, 2003.
13. Z. Zhuang, Y. Liu, L. Xiao, and L.M. Ni. *Hybrid Periodical Flooding in Unstructured Peer-to-Peer Networks*. International Conference on Parallel Computing, 2003.
14. D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos. *Exploiting Locality for Scalable Information Retrieval in Peer-to-Peer Systems*. Information Systems Journal, Vol. 30, No. 4, pp. 277-298, 2005.