DCMP: A Distributed Cycle Minimization Protocol for Peer-to-Peer Networks

Zhenzhou Zhu, Panos Kalnis, and Spiridon Bakiras, *Member, IEEE*

***Abstract*— Broadcast-based Peer-to-Peer (P2P) networks, including flat (e.g., Gnutella) and two-layer super-peer implementations (e.g., Kazaa), are extremely popular nowadays due to their simplicity, ease of deployment and versatility. The unstructured network topology, however, contains many cyclic paths which introduce numerous duplicate messages in the system. While such messages can be identified and ignored, they still consume a large proportion of the bandwidth and other resources, causing bottlenecks in the entire network.**

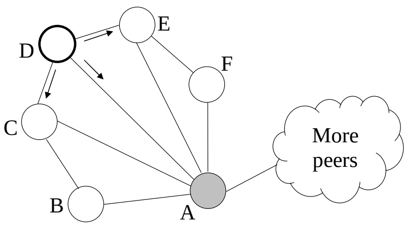
In this paper we describe DCMP, a dynamic, fully decentralized protocol which reduces significantly the duplicate messages by eliminating unnecessary cycles. As queries are transmitted through the peers, DCMP identifies the problematic paths and attempts to break the cycles, while maintaining the connectivity of the network. In order to preserve the fault resilience and load balancing properties of unstructured P2P systems, DCMP avoids creating a hierarchical organization. Instead, it applies cycle elimination symmetrically around some powerful peers to keep the average path length small. The overall structure is constructed fast with very low overhead. With the information collected during this process, distributed maintenance is performed efficiently even if peers quit the system without notification. The experimental results from our simulator and the prototype implementation on PlanetLab, confirm that DCMP improves significantly the scalability of unstructured P2P systems without sacrificing their desirable properties. Moreover, due to its simplicity, DCMP can be easily implemented in various existing P2P systems and is orthogonal to the search algorithms.

***Index Terms*— Network protocols, distributed systems, peer-to-peer.**

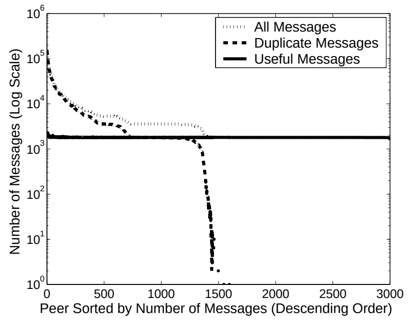
I. Introduction

PEER-TO-PEER (P2P) technology is attracting a lot of attention since it simplifies the implementation of large, ad-hoc, distributed repositories of digital information. In a P2P system numerous nodes are interconnected and exchange data or services directly with each other. There are two major categories of P2P architectures: (i) Hash-based systems (e.g., CAN [1], Chord [2]), which assign a unique key to each file and forward queries to specific nodes based on a hash function. Although they guarantee locating content within a bounded number of hops, they require tight control of the data placement and the topology of the network. (ii) Broadcast-based systems (e.g., Gnutella [3]), which use message flooding to propagate queries. There is no specific destination; hence, every neighbor peer is contacted and forwards the message to its own neighbors until the message's lifetime expires. Such systems have been successfully deployed in practice to form world-wide ad-hoc networks, due to their simplicity and versatility. Here we focus on broadcast-based P2P architectures. Our methods are also applicable to two-layer networks based on super-peers (e.g., Kazaa [4]), since the super-peer layers resemble Gnutella-style protocols.

Z. Zhu and P. Kalnis are with the Department of Computer Science, National University of Singapore, 3 Science Drive 2, Singapore 117543.
E-mail: {zhuzhenz, kalnis}@comp.nus.edu.sg.
S. Bakiras is with the Department of Mathematics and Computer Science, John Jay College of Criminal Justice, City University of New York, 445 West 59th Street, New York, NY 10019.
E-mail: sbakiras@jjay.cuny.edu.



(a) Example network



(b) Total vs. duplicate messages

Fig. 1. Cycles introduce numerous duplicate messages

Assume the network topology of Fig. 1(a) and let peer D initialize a query message msg. D broadcasts msg to A, C and E. C returns any qualifying results and propagates msg to A and B. Similarly, E propagates msg to A and F; this procedure continues until the maximum number of hops (typically 7 or 8) is reached. Note that A receives the same message five times. Existing systems tag query messages with a unique identifier and each peer maintains a list of recently received messages. When a new message arrives, the peer checks whether it has already been received through another path. If this is the case, it simply ignores the incoming message. We call this method *Naïve Duplicate Elimination* (NDE).

The motivation of this work is that most real-life networks exhibit power-law topology [5]; there is a small number of peers with many neighbors (A in our example), while most peers have fewer neighbors. If we employ NDE in our example, most of the overhead due to duplicate elimination will occur in A. Overloading A is likely to affect many other nodes since A is the hub between the two parts of the network[1]. To verify

[1]Obviously, our methods apply to any topology that contains cycles. We focus on power-law topologies because they are more common and the gain is more prominent.

this claim, we deployed a 3000-node Gnutella-style power-law network and counted the number of duplicates, useful messages and total messages (see Section V for details). The data are sorted by total messages first and then by useful messages, as shown in Fig. 1(b). Nodes appear in descending workload order; therefore, $x = 0$ corresponds to the node which receives the most messages. It is clear from the graph that a large proportion of the transmitted messages are duplicates which will be ignored; similar results appear in Ref. [6]. Observe that several low-degree nodes (i.e., peers with few neighbors) do not receive any duplicates because they do not participate in any cycle. On the other hand, our investigation revealed that the high-degree nodes (i.e., peers with many neighbors) receive most of the useless messages (the graph is in logarithmic scale), since the probability of being involved in cycles is higher.

Duplicate messages affect severely the response time and scalability of P2P systems, since they consume bandwidth and system resources, primarily from high-degree peers which are crucial for the connectivity of the network. In this paper, we describe the *Distributed Cycle Minimization Protocol* (DCMP) which aims at cutting the cyclic paths at strategic locations, in order to avoid introducing duplicate messages in the network. In DCMP any peer which detects a duplicate message can initiate the cutting process. This involves two steps: First, the peers in the cycle elect a leader, called *GatePeer*. At the second step, the cycle is cut at a well-defined point with respect to the GatePeer. GatePeers are also important for maintaining the connectivity and optimal structure of the network when peers enter or quit without notification. Since any peer can become GatePeer via a distributed process, the system is resilient to failures.

The main characteristics of DCMP are: (i) it reduces duplicate messages by as much as 90%, (ii) it requires few control messages, therefore the overhead is minimal, (iii) DCMP is suitable for dynamic networks with frequent peer arrivals and departures/failures, since it is fully distributed and requires only localized changes to the network's structure, and (iv) there is a tradeoff between eliminating the cycles and maintaining the connectivity of the network. DCMP performs symmetric cuts and includes mechanisms to detect network fragmentation. As a result, the connectivity and average path length remain relatively unaffected.

We performed an extensive experimental evaluation of our protocol in a simulator using flat and super-peer network topologies. We also implemented a prototype which was deployed on PlanetLab [7]. Our experiments indicate that DCMP achieves substantial reduction in network traffic and response time, hence improving the scalability of broadcast-based P2P systems. Due to its simplicity, DCMP can be implemented in many existing P2P systems such as Kazaa or Gia [8]. Moreover, DCMP is orthogonal to the search algorithms.

The rest of the paper is organized as follows: Section II presents the related work. Next, in Section III we describe the main aspects of DCMP, whereas in Section IV we discuss how DCMP deals with dynamic networks. In Sections V and VI we present the experimental results from our simulation and the Planetlab implementation, respectively. Finally, Section VII concludes the paper and discusses directions for future work.

II. Related Work

Research in the P2P area was triggered by the success of systems like Gnutella [3] and Kazaa [4]. Gnutella is a pure P2P system which performs searching by Breadth-First-Traversal (BFT) of the nodes around the initiator peer. Each peer that receives a query propagates it to all of its neighbors up to a maximum of d hops. By exploring a significant part of the network, it increases the probability of satisfying the query. BFT, however, overloads the network with unnecessary messages; moreover, slow peers become bottlenecks. To overcome these problems, Kazaa implements a two-layer network. The upper layer contains powerful peers, called *SuperPeers* (or UltraPeers); slower peers connect only to SuperPeers. The upper layer forms a Gnutella-like network among SuperPeers. Searching is performed by BFT at the upper layer only, since SuperPeers contain the indices of their children. Ref. [9] contains a detailed analysis of such configurations.

Yang and Garcia-Molina [10] observed that the Gnutella protocol could be modified in order to reduce the number of nodes that receive a query, without compromising the quality of the results. They proposed three techniques: (i) Iterative Deepening, where multiple BFTs are initiated with successively larger depth. (ii) Local Indices, where each node maintains an index over the data of all peers within r hops of itself, allowing each search to terminate after $d - r$ hops. (iii) Directed BFT, where queries are propagated only to a beneficial subset of the neighbors of each node. Several heuristics for deciding these neighbors are described. This method is extended in Ref. [11], [12], where the network is reconfigured dynamically based on the query statistics. A similar technique, called Interest-based Locality [13], contacts directly the most promising peer which is not necessarily a neighbor of the query initiator. If this process does not return enough results, a normal BFT is performed. Note that none of these techniques deals with duplicate elimination, but they are orthogonal to our protocol.

Limewire [14] maintains a table where it stores the IDs of duplicate messages and the directions (i.e., neighbor peers) from where they arrive. Once a message is identified as duplicate, it is discarded. Further message propagation avoids the directions from where duplicates have arrived. Keeping (*ID*, *Direction*) information for each duplicate message requires additional memory, especially in high-degree peers as they tend to receive a lot of duplicates. Therefore, Limewire also implements a simplified version which disables those connections from where "a lot" of duplicates are arriving. In practice, it is difficult to define unambiguously the disconnection threshold. Moreover, this method may compromise the connectivity of the network, as we show in our experiments.

Gia [8] improves the scalability of Gnutella by using a combination of topology adaptation, flow control and one-hop replication. Topology adaptation means that a node will prefer to connect to high capacity peers (capacity depends on bandwidth, processing power, etc.), even by rejecting some of its current neighbors. Gia performs search by biased random walks, where each peer forwards the query to the neighbor with the highest capacity. Nevertheless, the possibility of duplicates still exists. Consider, for instance, the network of Fig. 2, where the order of the peers based on capacity is: A, B, C, D (A has the highest capacity). Let peer A receive a query message. Gia routes the message as follows: $A \rightarrow B \rightarrow C \rightarrow A$. Therefore, A receives a duplicate. Since A knows that it has already sent the message to B, this time it chooses D. The message follows the path $A \rightarrow D \rightarrow B$, thus B also receives a duplicate. Although the message is propagated to one peer at a time, there may be many duplicates because the maximum number of hops d is much larger than in Gnutella. Gia also implements a flow control mechanism by assigning tokens to neighbors. The aim is to prevent overloading the peers beyond their capacity. Flow control, however, allows or blocks useful and duplicate messages without distinction. Our protocol, on the other hand, can be implemented on top of Gia in order to eliminate the cycles which cause duplicates.

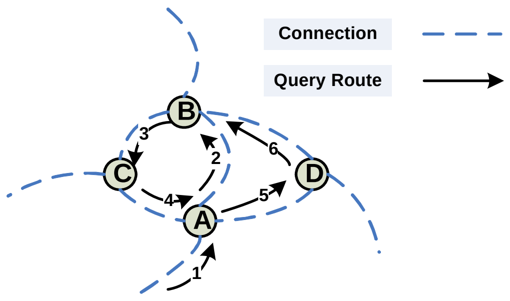


Fig. 2. Example of duplicate messages in Gia

In order to reduce the unnecessary traffic (i.e., duplicates), ACE [15] and LTM [16] use network delay as a metric to reconstruct the Gnutella network topology. In ACE, each peer uses the PRIM algorithm to build a multicast tree around itself. In LTM, each peer periodically broadcasts detection messages to discover and cut the connections which have the maximum delay. Disagreement of measured delay due to unsynchronized clocks, causes problems when deciding the cut positions, which can influence the network connectivity. Moreover, the network delay metric mainly focuses on disabling the connections between peers which are physically far away (e.g., different countries), without considering the shortcuts created by powerful peers, which are beneficial for exploiting large parts of the network.

The previous discussion applies to ad-hoc dynamic P2P networks without any guarantee on the availability of resources; the majority of P2P systems in use belong to this category. Alternatively, by allowing strong control over the topology of the network and the contents of each peer, Distributed Hash Table systems (e.g., CAN [1], Chord [2]) can answer queries within a bounded number of hops. Such configurations are outside the scope of this paper. Moreover, this paper focuses on the search process; we do not consider the downloading of files after they have been located.

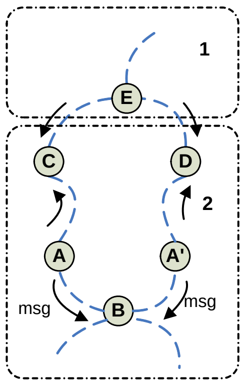
III. Protocol Design

In this section, we describe our protocol in details and explain why it is superior to existing approaches. To assist our discussion, first we present the notation we use throughout this paper.

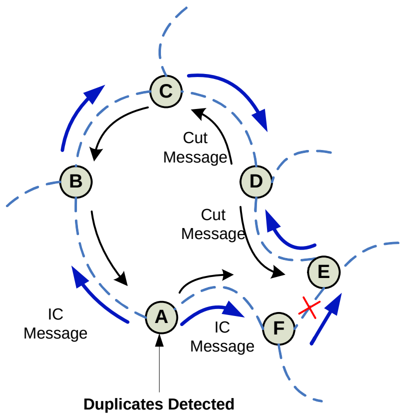
- When a node generates a query message msg, the message is assigned a globally unique ID denoted as: $GUID(msg)$
- Let A and B be two neighbor nodes (i.e., they have a direct overlay connection). The connection between them is denoted as: $\overline{AB}$
- Let a message travel from A to B. We denote the direction of the travelled path as: $A \rightarrow B$ and the reverse direction as $B \rightarrow A$
- Let A receive a message msg from its neighbor B. Then A places the following pair into the history table: $(GUID(msg), B \rightarrow A)$

A. Simplistic Cycle Elimination (SCE)

To motivate our approach, here we describe a straightforward method for eliminating cycles and explain its drawbacks. Consider Fig. 3(a) and let peer B receive the same message msg from A and A'. B identifies msg as duplicate by searching its $GUID$ in the history table. Both the direction $A \rightarrow B$ of the first msg (which is recorded in the table) and the direction of the duplicate msg, $A' \rightarrow B$, are parts of a cycle. A simplistic approach is to disable either connection $\overline{AB}$ or $\overline{A'B}$, in order to eliminate the cycle.



(a) Simplistic Cycle Elimination



(b) Distributed Cycle Minimization Protocol

Fig. 3. Cycle elimination methods

This approach, however, is prone to problems when multiple nodes in a cycle perform this cycle elimination operation simultaneously. Consider a different case, where nodes C and D receive duplicates and decide to eliminate the cycle at the same time by disabling $\overline{CE}$ and $\overline{DE}$, respectively; then regions 1 and 2 will be disconnected. The reduced connectivity has a negative effect on response time and on the ability of returning enough results. One way to tackle this problem is to force the disconnected pair of peers to continue exchanging information frequently about each other's status and reconnect, if necessary. Obviously, this poses a considerable overhead on the network.

B. DCMP: Distributed Cycle Minimization Protocol

In contrast to SCE, our protocol requires negotiation among all peers involved in a cycle about the optimal way to cut the cycle. Therefore, the probability of generating a disconnected network is minimized. The negotiation process is efficient, requiring only two messages per peer per cycle. Also, the information gathered during negotiation is used to repair the network with low overhead when peers join or fail/quit without notification.

The negotiation process can be initiated by any peer which receives a duplicate. Fig. 3(b) provides an example. Assume that peer A receives a message msg from $B \rightarrow A$ and, soon after, it receives the same message[2] (i.e., same $GUID$) from $F \rightarrow A$. Peer A identifies msg as duplicate by performing a lookup in its history table. The first step of our protocol is to gather information from all peers in the cycle. To achieve this, we introduce a new type of control message, called *Information Collecting* (IC) message.

Fig. 4 illustrates the structure of a typical IC message. Let icm be the IC message of our example. We set $GUID(icm)$ to be the same as the $GUID$ of the duplicate msg. This is done in order to facilitate the propagation of icm by the same mechanism which handles query answers in a Gnutella-style network. Note that if msg travels through many cyclic paths, multiple peers will detect the duplicates. To ensure that each IC message is unique we introduce another field, called $DetectionID$, which represents the direction of the connection where the duplicate was identified. In our example, $DetectionID(icm) \equiv F \rightarrow A$. The last field of the IC message is the *Node Information Vector* (NIV). NIV contains information about the peers which propagated the IC message. This includes the bandwidth of each peer, the processing power, the IP address and topology information about the peer's degree and its neighbors. In our example, the NIV of icm initially contains information only about peer A.

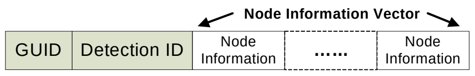


Fig. 4. Structure of the IC message

Peer A sends one copy of icm towards $A \rightarrow B$ and another towards $A \rightarrow F$. Each peer which receives icm appends its own information to the NIV field and then treats icm similarly to an answer message; therefore, icm is propagated following the reverse path of the original message msg. Since two copies of icm are sent, at some point, a peer will receive a duplicate of icm; in our example this happens at peer D. The algorithm for handling IC messages is shown in Fig. 5.

Observe that D is not necessarily the origin of msg. Assume that a node D' further away (not shown in the illustration) initiated msg. Also assume that icm arrives from $C \rightarrow D$ faster than from $E \rightarrow D$. Since D has not received a duplicate of icm yet, it will propagate icm towards $D \rightarrow D'$. Therefore, potentially there will be an overhead of at most $TTL-1$ messages per cycle[3]. Similarly, if for any reason the cycle ceases to exist (e.g., node failure), it is possible that no peer receives a duplicate icm. In this case icm is simply propagated towards the origin of msg. We could avoid both cases by using a more complicated protocol. However, TTL

[2]Note that msg and its duplicate are not shown in the illustration.

[3]TTL: Time To Live. It is synonymous to the maximum number of hops d.

```
Precondition: Node $N$ receives an IC message $icm$, from direction $M \rightarrow N$
1.  Search the history for a recent IC message $icm'$ which satisfies:
      $GUID(icm) = GUID(icm')$ and
      $DetectionID(icm) = DetectionID(icm')$
2.  if $icm'$ is found, then // a duplicate IC message is found
3.    Combine NIV of $icm$ and $icm'$ into a single vector $v$
        At this point, $v$ contains information about all the nodes in the cycle
4.    Using $v$, decide which connection in the cycle will be disabled
5.    Forward the decision to all the nodes in the cycle
6.  else // no duplicate IC found
7.    Append the node information of $N$ to the NIV field in $icm$
8.    Find in the history a message $msg$ such that
        $GUID(msg) = GUID(icm)$
9.    Assume that $icm$ is an answer message for $msg$
        Use Gnutella protocol to send $icm$ towards the reverse path of $msg$
```

Fig. 5. Algorithm for handling the IC message

is between 3 and 7 in practice, so the potential overhead is very low.

Recall that our protocol does not eliminate all cycles. Obviously, if the cycle contains more than $2 \cdot TTL$ edges it will not be detected, since there will be no duplicates. Moreover, we introduce a parameter TTL_d, where $0 < TTL_d \leq TTL$. If a duplicate msg is detected more than TTL_d hops away from the origin of msg then we do not eliminate the cycle. The intuition is that there is a tradeoff between preserving the connectivity of the network and minimizing the duplicates. Therefore, we allow some large cycles (some duplicates as a consequence) in the network. In Section V we will discuss how we select the TTL_d value. Note that the introduction of TTL_d does not require any modification of the Gnutella-style query message.

From the NIVs of the icm messages, D has information about all nodes in the cycle, namely A, B, C, D, E and F. Using this information D decides which connection should be disabled; we will discuss the exact criteria in the next section. For now assume that D decides to cut the $\overline{EF}$ connection. In order to inform the other peers in the cycle about the decision, we introduce one more message type called *Cut Message* (CM). CM contains the $GUID$ and $DetectionID$ which are set equal to the $GUID$ and $DetectionID$ of the corresponding IC message. Additionally, there is a field which identifies the connection to be cut. Direction is not important in this field since any of the two nodes in the pair can disable the connection. Peer D sends two copies of the cut message towards $D \rightarrow C$ and $D \rightarrow E$, respectively. These are the reverse directions from where icm arrived previously. Similarly, CM messages received by any peer, are propagated towards the reverse path of the corresponding IC. Eventually, the cut message will reach either E or F and one of these peers will cut the connection, thus eliminating the cycle. The algorithm for handing CM messages is presented in Fig. 6.

```
Precondition: Node $N$ receives a cut message $cm$
1.  if $N$ is involved in the connection to be disabled then
2.    if the corresponding connection is still active then disable it
3.  else
4.    Search the history for an IC message $icm$ such that
        $GUID(icm) = GUID(cm)$ and
        $DetectionID(icm) = DetectionID(cm)$
5.    if such $icm$ is found then forward $cm$ to the reverse direction of $icm$
6.    else ignore $cm$ // $N$ was the initiator of $icm$
```

Fig. 6. Algorithm for handling the CM message

Observe that D could initiate only one copy of the cut message to traverse the cycle. The reason for sending two copies is threefold: (i) Our approach uses the standard Gnutella protocol to envelope the messages. If one message was used, we would need to consider special cases for handling the CM messages, thus complicating the protocol. (ii) The delay until cutting the cycle is minimized, since the average number of hops for CM messages is reduced. (iii) The total number of transmitted messages is the same, since the cut message carries useful information for all the peers and must traverse the entire cycle, as we will discuss in the next section.

C. Deciding the Cutting Position

Here we explain how we choose the connection to disable, in order to cut a cycle. This desision is made at the peer which receives two copies of the same IC message (i.e., D in our example). This peer is the *coordinator*; in DCMP any peer can act as coordinator. A straightforward way is to eliminate randomly one edge of the cycle. However, our experiments indicate that this approach does not preserve the connectivity of the network. In order to achieve better results, we rely on the properties of the peers in the cycle. Recall that the IC messages which arrive at the coordinator, have gathered this information.

The following definitions are necessary:

Definition 1 (Opposite edge): Let S_N be the set of nodes which form a cycle. For a node $N \in S_N$, the edge opposite to it is an edge $\overline{MM'}$ such that: $M \in S_N$, $M' \in S_N$, and there is a path p from N to M and a path p' from N to M' such that $p \subset S_N$, $p' \subset S_N$ and:

$$|p| = \begin{cases} |p'| = \lceil |S_N|/2 \rceil & \text{if } |S_N| \text{ is odd} \\ |p'| - 1 = |S_N|/2 & \text{if } |S_N| \text{ is even} \end{cases} \quad (1)$$

Definition 2 (Peer power): The power $\mathcal{P}$ of a node N is given by the following formula:

$$\mathcal{P}(N) = c_1\mathcal{B} + c_2\mathcal{C} + c_3\mathcal{D} \quad (2)$$

where $\mathcal{B}$ is the bandwidth, $\mathcal{C}$ is the CPU processing power, $\mathcal{D}$ is the peer's degree (i.e., maximum number of simultaneous connections) and $c_{1\ldots3}$ are predefined constants.

It is obvious why the bandwidth and CPU power characterize how powerful a peer is. The degree factor is used, because a peer which accepts many neighbors is beneficial for low network diameter. There are several other factors which can influence the characteristics of the network. For example, Ref. [17] suggests that the distribution of the lifespan of peers follows the Pareto distribution and proposes several methods to improve the network stability according to this observation. Such factors can be easily incorporated in our protocol.

Definition 3 (GatePeer): The most powerful peer in a cycle is called GatePeer.

The heuristic we use in our protocol is to cut cycles by disabling the connection which is *opposite* to the corresponding GatePeer. The intuition is that our method minimizes the average number of hops from the GatePeer to any peer in the cycle. The GatePeer, in turn, will most probably be the hub which connects the cycle to many other peers; therefore, the connectivity will be largely preserved. Also, since the GatePeer can process messages fast, the response time will not suffer.

Recall that the GatePeer is elected by the coordinator. The coordinator is the only peer which knows the characteristics of all members in the cycle. All peers must be informed about their corresponding GatePeer, including the GatePeer itself which does not know its status yet; for this reason, the IP address of the GatePeer is appended in the CM messages. As we explain later, this is also beneficial for the fast recovery from failures. The algorithm for selecting a GatePeer is shown in Fig. 7.

```
Precondition: Node $N$ receives two IC messages $icm$ and $icm'$ which
    satisfy the conditions: $GUID(icm) = GUID(icm')$ and
    $DetectionID(icm) = DetectionID(icm')$
1. Calculate the $power\ P_i$ of each peer in NIVs using Definition 2
2. Let the peer with $Max(P_i)$ be the GatePeer
3. In case of a tie, the GatePeer is the one with the largest $GUID$
4. Find the position to be disabled based on the GatePeer and Definition 1
5. Generate $Cut$ message(s) accordingly
```

Fig. 7. Algorithm for selecting the GatePeer in a cycle

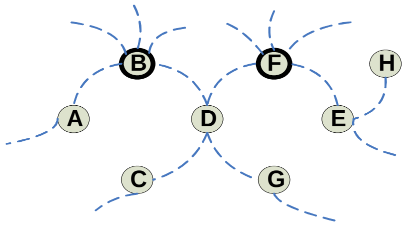
D. Disseminating GatePeer Information

GatePeers assist to recover from node failures and are used as entrance points in a dynamic network (refer to Section IV-A); therefore, it is beneficial for other peers outside the cycle to know which are the nearby GatePeers. To disseminate this information with minimal overhead, we use a piggyback technique. Each GatePeer appends the messages passing through it with the following information: $(NIV_{GP}, HopsNumber)$, where NIV_{GP} is the information vector of the GatePeer (including its IP address) and $HopsNumber$ is an integer indicating the distance (in hops) from the message origin to the GatePeer. We call this process *tagging*. While the overhead of tagging is only a few bytes per message, the GatePeer information remains relatively stable for most of the time. Therefore, we can achieve our goal by tagging messages periodically. Observe that immediately after a cycle is eliminated, most probably a new GatePeer is elected. In order to advertise fast its identity, the GatePeer performs tagging frequently. Later, the GatePeer tags messages infrequently to let peers up to TTL hops away realize that it is still alive. We investigated different values for the tagging frequency and length of the tagging process in the simulation. Our results suggest that the following settings provide a good tradeoff between cost and efficiency: for a period of 1 min after a new GatePeer is elected, a message is tagged every 5 sec; after that, the tagging frequency is lowered to 1 message every 10 min. Note that the exact values are not crucial and the overhead of tagging is small (refer to Section VI-A.3 for details).

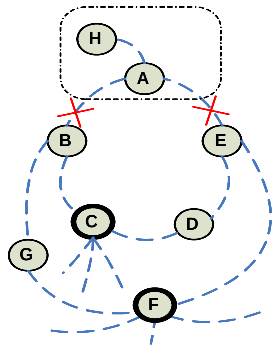
Definition 4 (Transitive peer): A peer that continuously receives tagged messages from more than one direction is called a *transitive peer*.

Peers may receive tagged messages from several GatePeers continuously. If the tagged messages do not come all from the same direction, it is possible that the peer is a hub. An example is shown in Fig. 8(a) where B, F are GatePeers and D receives messages tagged by both B and F; peer D is a transitive peer. Due to the strategic position of transitive peers, they are important for the connectivity of the network, should a node fail/quit. Therefore, transitive peers must also advertise their presence. To keep the protocol simple, transitive peers use the same tagging mechanism as GatePeers and are treated by other nodes as GatePeers.

Any peer which is not GatePeer or transitive peer, is called *normal* peer. Normal peers may receive tagged messages from multiple GatePeers (or transitive peers) but all come from the same direction. In the example of Fig. 8(a), peer H receives tagged messages from D and F but all arrive through $E \rightarrow H$. We



(a) Example of transitive peer D



(b) Example of network split in DCMP

Fig. 8. Transitive peers and network splits

call the closest GatePeer in this direction the *referred* GatePeer of the normal peer. Note that the referred GatePeer is not necessarily a neighbor of the normal peer.

Definition 5 (Primary Direction): Let N, M be two neighbor peers. Let the messages tagged by the referred GatePeer of N arrive from direction $M \rightarrow N$. The reverse of this direction (i.e., $N \rightarrow M$) is the *primary direction* of N.

Continuing our example, the referred GatePeer of H is F and the primary direction of H is $H \rightarrow E$. Note that both D and F are considered as GatePeers by H; however, F is closer.

E. Concurrent Cycle Elimination

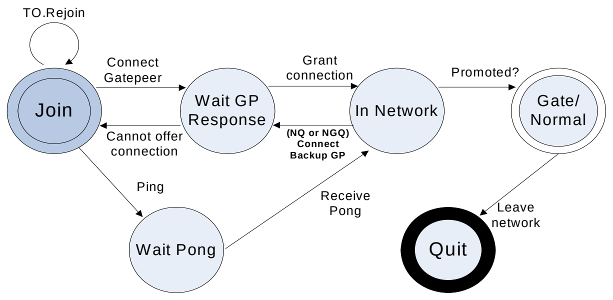
In Section III-A we demonstrated how SCE may split the network into two unconnected parts. In DCMP this problem is greatly reduced, mainly because the cutting position is defined deterministically. Nevertheless, as we show in Fig 8(b), it is still possible to split the network. For simplicity, in this example we measure the power $\mathcal{P}$ of a node only by its degree; therefore, the GatePeer in the cycle $ABCDEA$ is C, and the one in the cycle $ABGFEA$ is F. The connection opposite to C is $\overline{AE}$, whereas the one opposite to F is $\overline{AB}$. Hence, if the two connections are disabled *simultaneously*, nodes A and H are isolated from the network.

We propose an effective, yet simple solution to this problem. Immediately after a connection is disabled due to a cycle, the nodes at both ends of this connection start listening for a tagged message from their corresponding GatePeer[4]. For example, A and E will listen for a tagged message from C (similarly, A and B also expect a tagged message from F). Recall that after eliminating the cycle, C will tag messages frequently. If either A or E do not receive any tagged message from C for some time[5], they reestablish the $\overline{AE}$ connection. Then they start listening again for a message tagged by C. If they still cannot receive such a message (because, for instance, D failed in the meanwhile), both A and E attempt to connect directly to C. During this process new cycles may be formed. However, our experiments indicated that in practice this happens rarely. Moreover, even if a new cycle is generated, it will be identified and eliminated soon after.

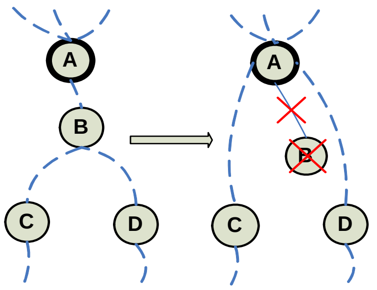
[4]Tagging is beneficial during peer failures (see next section). Concurrent cycle elimination is rare in our protocol, and by itself would not justify the tagging mechanism.

IV. Dynamic Networks

The previous discussion assumes a static snapshot of the network; here we explain the handling of node arrivals and departures. Node arrivals are easy to handle. The departure case, however, is more complex. To improve fault tolerance, our protocol allows nodes to depart without notification; therefore, both proper departures and failures are handled in the same way. DCMP uses the information about GatePeers to maintain the connectivity of the network without imposing additional overhead. The entire process is summarized in Fig. 9(a).



(a) State diagram to handle node quit/join. *GP* is GatePeer. *NGQ* means Neighbor GatePeer Quit. *NQ* means departing node is in the primary direction of a peer. *TO*: timeout period



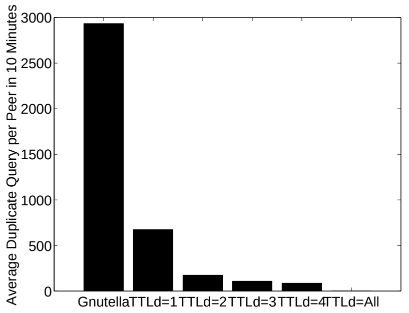
(b) Failure of normal peer B

Fig. 9. Join and quit in dynamic networks

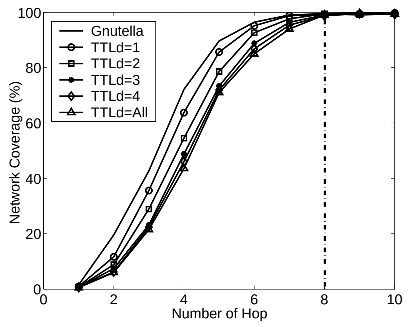
A. Peer Arrival

In existing Gnutella-style networks, joining nodes first contact some well-known peers and send ping messages which are broadcasted in the network. Peers willing to accept the new connection, reply with a pong message. Unfortunately, there is a considerable overhead due to ping messages. For this reason, DCMP uses a slightly different technique. First, the node attempts to connect to some GatePeers (from previous cache and/or well-known peers). Only if this process fails, it uses the ping/pong protocol.

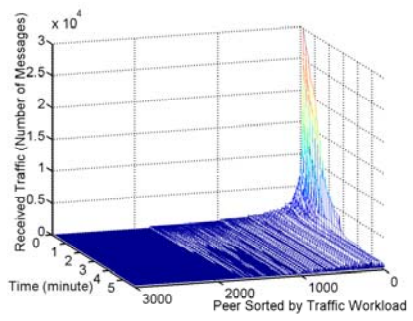
[5]The waiting period is set to 30 sec in our prototype, but the exact timing is not crucial.



(a) Duplicate messages



(b) Connectivity



(c) Received traffic (DCMP)

Fig. 10. Analysis of traffic workload and effect of using different TTL_d (static networks)

Assuming that the newcomer peer N was in the network before, it is possible that it has cached the IP addresses of some GatePeers. N attempts to contact the GatePeers, hoping they are still in the network. The intuition is that GatePeers are powerful and most probably can accept the new connection. Even if there are no free resources at the moment, a GatePeer G can recommend to N a new set of GatePeers in G's vicinity. Given that this process succeeds, N is able to join the network without the overhead of broadcasting a large number of ping messages. The savings can be substantial if nodes join/leave the network frequently.

B. GatePeer Departure

All peers, including GatePeers, receive tagged messages periodically; therefore, they have a list of nearby GatePeers (recall that transitive peers are also handled as GatePeers). From this information, a GatePeer G knows its distance to each of the nearby GatePeers. Taking into account the distance and power of these GatePeers, G generates an ordered list of *backup* GatePeers. Then G broadcasts this list to its direct neighbors (i.e., only one hop away). The guideline for selection is that the backup GatePeers should be powerful enough to accept the direct neighbors of G, in case G quits. In our experiments, we found two to five backup GatePeers were usually selected, depending on the degree of G and the capacity of its neighboring GatePeers. To maintain the backup list up-to-date, backup GatePeers selection is performed periodically and information broadcasting is only needed when there is an update.

If G quits/fails, its neighbors attempt to repair the network. The backup GatePeers of G connect to each other. The rest of G's neighbors attempt to connect to some backup GatePeers randomly. Therefore, only a small number of peers (i.e., the direct neighbors of G) are affected and the network topology does not change significantly. If for some reason this process is not successful (e.g., none of the backup GatePeers can accept more connections, because of simultaneous GatePeer failures), then the affected peers simply re-join the network using the peer arrival procedure described above.

C. Departure of Normal Peer

If a normal peer quits/fails we must also ensure that the network remains connected. In contrast to GatePeer failures, this case affects only neighbors whose primary direction includes the quitting node. To explain this, consider the network of Fig. 9(b). Peer A is a GatePeer and it is also the *referred* GatePeer of both C and D. Assume that B fails (B is a normal peer) and note that the primary direction of C and D is $C \rightarrow B$ and $D \rightarrow B$, respectively. Recall that the primary direction indicates the preferred path towards the rest of the network. Therefore, B's failure is likely to affect the connectivity for the subgraphs under C and D. In our protocol, the affected peers attempt to connect to their referred GatePeer; hence, C and D will connect to A.

V. Evaluation by Simulation

We developed an event-driven simulator, which is accurate down to the message transmission layer and takes into account the processing latency and the network delay. We simulate the latency caused by network congestion at the overlay layer; however, we do not simulate the TCP/IP layer. The simulator is written in C++ and was executed on a Linux machine (with 3.0GHz CPU and 3GB of RAM). We used power-law topology with average degree 3.4, whereas the network size varied from 500 to 10K peers (results based on 3000 nodes by default in this section). The bandwidth of each peer ranged from 56Kbps (i.e., modem) to 45Mbps (i.e., T3 connection), following also power-law distribution. The TTL for the messages was set to 8 (except for the random walk algorithm). Peers initiated queries with uniform distribution and mean query frequency 3.6 queries/peer/hour. Each experiment was executed with six different seeds and the results show the average of all runs.

A. Topology and Workload Analysis

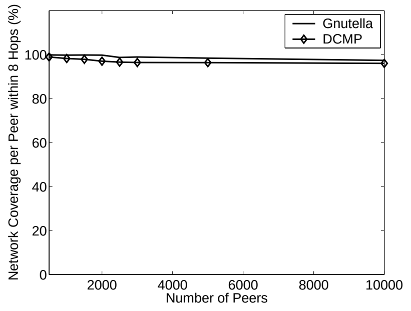
In the first set of experiments, we generate a power-law network with 3000 peers and count the number of duplicate messages before DCMP starts to eliminate cycles. Then we allow DCMP to reach a stable state and count the number of duplicate messages again. In Fig. 10(a), we show the number of duplicate messages after eliminating cycles by using different TTL_d values. Recall that TTL_d guides the process of eliminating the cycles which are shorter than certain lengths. Therefore, cycles that have more that $2 \cdot TTL_d$ edges are largely maintained. $TTL_d = All$ will eliminate all cycles causing the network to degenerate to a tree. From the graph we observe that the number of duplicate messages is reduced considerably for $TTL_d = 2$ (i.e., more than 90% of the duplicate messages are eliminated). Further increasing TTL_d does not result to significant improvement.

However, there is a tradeoff between the number of cycles and the network connectivity. If we eliminate too many cycles, the average distance (in hops) between any pair of nodes will increase and so will the average delay. Moreover, the system's resilience to node failures will suffer. In Fig. 10(b) we present the average connectivity of the network for varying TTL_d. For instance, if $TTL_d = 1$, a message can reach almost 65% of the peers in the network within 4 hops, on average; however, if $TTL_d = All$ (i.e., tree topology), messages can reach only 43% of the peers. From these two diagrams of Fig. 10, we conclude that $TTL_d = 2$ provides a good tradeoff between the number of duplicates and connectivity; therefore, we use this value for the following experiments.

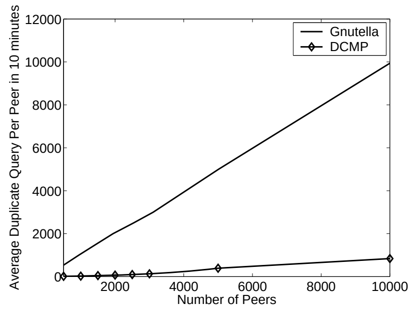
Recall that duplicate messages affect mostly the high-degree peers. This is obvious in Fig. 10(c), where peers are sorted according to their workload. As time passes, DCMP eliminates a large number of small cycles around high-degree peers, reducing significantly their workload. On the other hand, the workload for the rest of the peers remains almost unaffected.

B. *Influence of Network Size*

In this experiment, we vary the number of peers in the network. Fig. 11(a) shows the network coverage. The graph reveals that DCMP preserves short routing paths as the network size increases. DCMP eliminates only the small cycles around GatePeers, achieving almost as good coverage as Gnutella[6]. In Fig. 11(b) we present the average number of duplicates for various network sizes. Observe that for DCMP the number of duplicates increases very slowly, since the number of cycles with length larger than $2 \cdot TTL_d$ (i.e., the ones that introduce duplicates in DCMP) is small.



(a) Connectivity



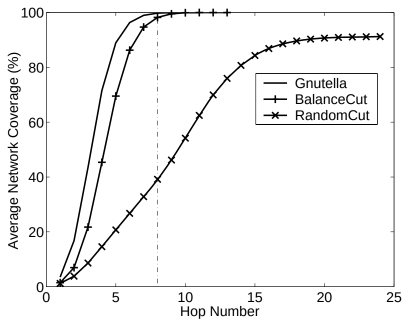
(b) Duplicate query

Fig. 11. Scalability for Gnutella and DCMP (static networks)

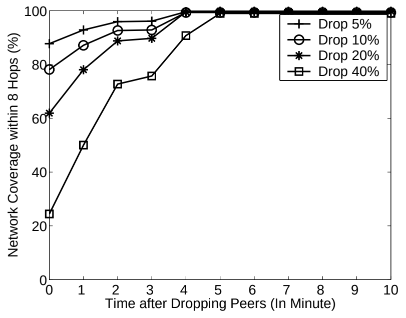
[6]The default $TTL = 8$. For other TTL values the graph follows the difference between the Gnutella and $TTL_d = 2$ lines of Fig. 10(b).

C. *Symmetric Cut vs. Random Cut*

Here we investigate the effectiveness of the symmetric cut heuristic employed by DCMP. We compare our method against cutting the cycle at a random position. The results are shown in Fig. 12(a), where we draw the network coverage for varying number of hops. By cutting cycles symmetrically to GatePeers, DCMP manages to follow closely the good coverage of Gnutella. The random heuristic, on the other hand, creates long chains of peers and network fragments, since all peers in a cycle may decide to break the cycle concurrently. Therefore, the coverage drops significantly; for instance, less than 40% of the peers are reachable within 8 hops.



(a) Balance vs. random cut

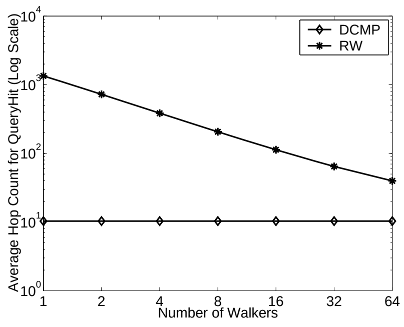


(b) Random failure analysis

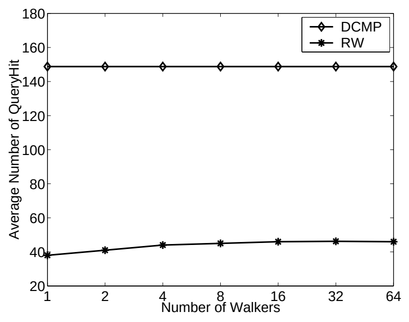
Fig. 12. Comparison of symmetric/balance cut and random cut and failure analysis

D. *Failure and Attack Analysis*

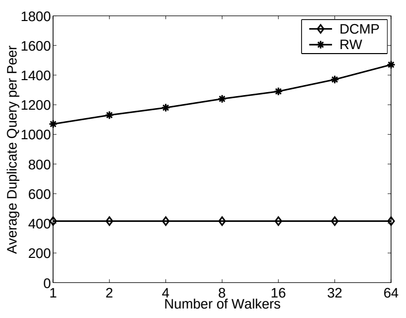
In peer-to-peer systems, peers are usually unstable and the network is very dynamic. One important requirement of the system is to be resilient to failures. To test the robustness of DCMP, we force 5-40% of all peers to fail simultaneously. All peers (i.e., both normal peers and GatePeers) have the same probability to fail. We calculate the network coverage immediately after dropping these peers and once a minute in the following 10 minutes. The failure can be detected either when a peer sends a message or when the "KeepAlive" timer of the TCP layer expires (in our simulation, the timer expires in 4 min). By utilizing the backup and referred GatePeer information, the network fragments can connect to each other efficiently even when 40% of the peers fail at the same time. Fig. 12(b) shows that the network coverage restores to almost 100% after 5 minutes. Interestingly, if there were more messages to be sent via the area where some GatePeers fail, the failures would be detected and repaired faster. The graph depicts the worst case, where many peers rely on the



(a) Path lengths of QueryHit



(b) Number of QueryHit



(c) Number of duplicate query

Fig. 13. Comparison of Random Walks and DCMP (dynamic network)

TABLE I
COMPARISON OF RANDOM WALKS, DCMP, GNUTELLA AND LTM

	Hop count for QueryHit	QueryHit Number	Duplicate Query
Gnutella	9.96	149.4	3100
DCMP	10.33	148.8	415
RW (8 Walkers)	206.67	45.1	1240
LTM	11.49	145.3	535

TCP layer for failure detection. During the experiment there were cases where all the backup GatePeers of a normal peer failed simultaneously. In these cases, the peer had to re-join the network.

A drawback of our protocol is that, compared to Gnutella, it is more vulnerable to well-orchestrated attacks. To verify this, we sorted all peers according to their power and failed simultaneously the top 1%. The coverage of the network dropped to around 20% and the system needed around 5 minutes to recover (very similar to Fig. 12(b)). Gnutella, on the other hand, is less affected because many nodes remain connected via longer paths. The protection of high-degree GatePeers against malicious attacks is an important issue of our future work, but it is outside the scope of this paper. Notice, however, that one could implement various methods on top of DCMP for, e.g., detecting malicious peers [18] or defending against DoS attacks [19]. This is possible, since these protocols work independently of each other.

E. Comparison with other Approaches

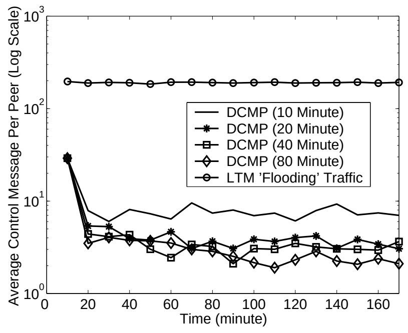
Ref. [6] uses *Random Walks* (RW) for searching in unstructured P2P networks. The algorithm initiates k random walkers. In order to reduce the system load, the walkers contact the initiator node periodically, to check whether the search should stop. Despite the overhead of contacting the query initiator, this approach reduces the total number of messages compared to flooding, and reduces the duplicate messages as a consequence. The tradeoffs are increased user-perceived delay and fewer answers, since RW favors searches for popular objects but exhibits poor performance for rare ones. Nevertheless, Gkantsidis et al. [20] observed that if RW is forced to transmit the same number of messages as flooding approaches, it achieves almost the same network coverage (the delay problem remains). Obviously, RW does not alter the network's structure. Nevertheless, we study it here, since it has the potential to minimize the duplicate messages.

LTM [16] is a different approach that periodically broadcasts detection messages to discover and cut the connections which have the maximum delay. In LTM, the following two steps are performed at each peer: (i) Forward a detection message: if a detection message (received or self-created with initial $TTL = 2$) has not expired, the peer inserts a new timestamp and broadcasts the message to the neighbor peers. (ii) Cut a connection: upon receiving two detection messages with the same $GUID$, the peer drops the link with the largest delay among all traversed links, using the timestamps to calculate delays.

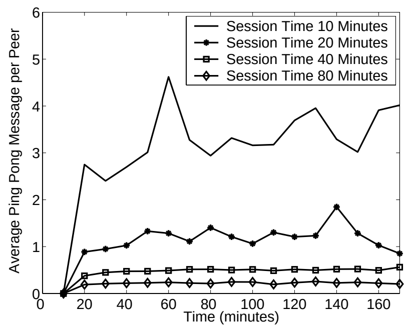
1) QoS and Duplicate Reduction Analysis: In our experiments, we varied the number of walkers from 1 to 64 and forced RW to transmit the same number of messages as DCMP (similar to Ref. [20]). For LTM, we followed the optimal frequency of broadcasting detection messages suggested by Ref. [16]. In Fig. 13 we compare RW and DCMP. First, the average delay is shown in Fig. 13(a) (delay is measured as the number of hops from the moment a query is sent until each answer arrives to the querying peer). We observe that the delay of RW is about four times larger than DCMP, even when many walkers are used. Increasing the number of walkers reduces the delay, which is expected since RW tends to flood the neighbors. In our experiments, there are around 150 replicas of each object in the network. Fig. 13(b) shows that DCMP can find almost all of them, but RW discovers less than 33% of the copies. Finally, Fig. 13(c) shows the number of duplicates. For all the cases, RW generates more duplicate messages than DCMP, if both methods transmit the same number of messages.

Table I summarizes the results for the four techniques. DCMP, RW and LTM transmit fewer messages for each query compared to Gnutella, since many duplicates are avoided. DCMP incurs lower delay, returns more results, and decreases the number of duplicate messages by 22%, compared to LTM. Furthermore, DCMP generates much less overhead than LTM, as we will explain in the next experiment.

2) Overhead Analysis and Effect of Peer Session Time: In order to reduce useless traffic, both DCMP and LTM transmit special messages to construct and maintain the desired network topology; however, the resulting overhead is different. To investigate this, we conducted the following experiment: We generated a power-law network with 3000 on-line peers and placed 3000 additional peers in a waiting list. When the session time of an on-line peer P had expired, P would fail and it would be placed at the back of the waiting list. At the same moment a random peer from the waiting list would join the network at a random location; therefore, the number of the on-line peers was remaining constant. The peer session time followed the exponential distribution; we varied the mean between 10 and 80 minutes. We run the simulation long enough for each of the original 3000 online peers to have the chance to quit and re-enter.



(a) Overhead analysis



(b) Ping and pong messages

Fig. 14. Overhead analysis and effect of session time on control and ping/pong messages

Compared to LTM, DCMP has much smaller overhead (i.e., control messages), which is due to the fact that LTM adopts an "eager" approach (i.e., broadcasts control messages periodically), while DCMP adopts a "lazy" one. As shown in Fig. 14(a), LTM's overhead is one to two orders of magnitude greater than that of DCMP (notice that we counted all tagged messages in DCMP as separate messages). In the same graph we analyze the effect of peer session time. We observe that the overhead increases when the network becomes more dynamic. This is caused by the unstable GatePeers, which tend to create more cycles. Fig. 14(b) confirms this phenomenon. When peers join and quit/fail with increasingly higher frequency, the GatePeer information used to maintain the network connectivity is outdated faster. As a consequence, joining peers rely more on the Gnutella-style ping/pong protocol. However, by joining at a random position, the probability of introducing a cycle (thus the overhead for cycle elimination) increases. Nevertheless, if the mean session time is more that 10 minutes (this number is consistent with most of the observations in the literature, e.g., Ref. [21]), the joining overhead for DCMP is reasonably small.

VI. Prototype Evaluation on PlanetLab

We implemented the DCMP protocol in a prototype and deployed it on PlanetLab [7]; our prototype implements all the features except the downloading of files after they are located. There are 665 nodes which are distributed over 315 locations in PlanetLab, at the time of writing this paper. Unfortunately, some nodes are problematic, so our experiments use up to 400 nodes scattered worldwide. This number may be considered small for a P2P network. However, we believe it is important to show accurate measurements (especially response time) from a real system.

We generated two network topologies which appear in real-life P2P networks [5], in order to test DCMP: (i) Power-law topology, with average degree 3.4. We used the PLOD [22] method to construct the network. This topology reflects the original Gnutella network (i.e., protocol v0.4). (ii) Two-layer network, with power-law distribution at the super-peer layer and quasi-constant distribution at the leaf layer. This topology corresponds to the latest version of the Gnutella protocol (i.e., v0.6). We used statistics from Limewire [23] to generate a realistic network.

In our experiments we use a small set of 5 nodes as the network seed. The IP addresses of the seed nodes are known to all peers. The seeds are used as entry points to propagate ping messages in order to assist other nodes to join. We also use a coordinator peer which transmits configuration parameters to other nodes, starts or stops the experiment and gathers statistics from all nodes. The seeds and the coordinator are used to assist the experimental setup; otherwise they are not required by our protocol.

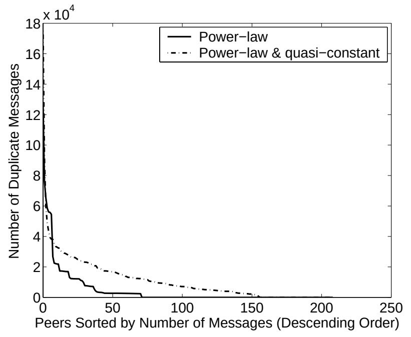
We compare DCMP with the Gnutella protocol which also represents the upper layer of super-peer P2P networks (e.g., Kazaa). We also evaluate our protocol against the Simplistic Cycle Elimination (SCE) technique (similar to the approach suggested by Limewire). Finally we compare DCMP with Random Walks. We use the following metrics: (i) Number of duplicate messages. This metric indicates how much unnecessary traffic is eliminated. (ii) Delay (or response time). It is the delay from the moment a query is initiated by a peer until the moment the first result reaches the peer. In our setup each query can be answered by 5% of the nodes (answers are uniformly distributed). Although this is not an accurate representation of files in a real P2P system, it is adequate for our experiments, since we are interested in the network structure instead of the search algorithm. (iii) DCMP Overhead: These are the control messages (IC, CM, message tagging), which are essential in our protocol.

In the following, we present the results of our experiments. In order to understand the behavior of DCMP, first we consider a static snapshot of the network (i.e., peers do not enter/leave). Next, we deploy a realistic dynamic network and measure the actual delay perceived by the users.

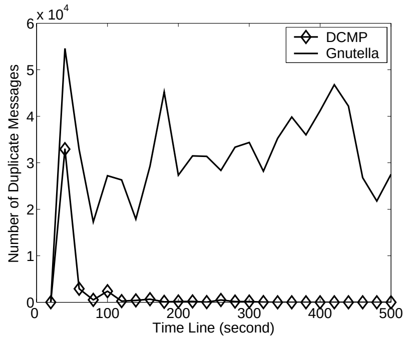
A. Static Peers

For the static snapshot, first we allow all peers to enter the network. Then, the coordinator peer broadcasts the command to start the experiment. From that point on, peers send messages to each other as usual, but no peer can enter/leave the network.

1) Duplicates Analysis: In Fig. 15(a) we analyze the duplicates' distribution in two topologies: Power-law (i.e., Gnutella) and Power-law Quasi-constant (i.e., super-peer architectures). The x-axis represents individual nodes appearing in descending workload order; therefore, $x = 0$ corresponds to the node which receives the most duplicates[7]. Both topologies are prone to a large number of duplicates; however, the two-layer network suffers most. In two-layer architectures, about 10% of the nodes are super-peers [23] having a large number of neighbors which are also super-peers. Therefore, cycles are formed with high probability and they introduce numerous duplicate messages.



(a) Duplicate distribution in Gnutella and super-peer architectures



(b) Number of duplicates vs. elapsed time for DCMP and Gnutella

Fig. 15. Duplicate distribution and time-line of duplicates

Fig. 15(b) shows the number of duplicates for both DCMP and Gnutella. The x-axis represents the elapsed time since the beginning of the experiment. Nodes record the number of duplicates they receive in 20 seconds intervals. The y-axis represents the sum of duplicates in all nodes. Initially, both systems experience a large number of duplicates. As time progresses, DCMP eliminates the cycles, therefore duplicates are reduced. Gnutella, on the other hand, generates continuously numerous duplicates. Note that in DCMP the number of duplicates drops significantly after about 20 seconds and almost all duplicates are eliminated after 100 seconds. The actual time for eliminating these cycles is affected by the size of the network and the exact number of cycles; in practice, it takes no more than a few minutes. Similar results were obtained for super-peer architectures.

2) Delay Analysis: DCMP eliminates cycles by disabling the connections symmetrical to GatePeers, in order to keep the network diameter small. Here, we investigate how DCMP affects the average number of hops; the actual delay is measured in the next section.

In our experiments, each peer generates traffic by initiating 10 query messages; the mean time between queries is 30 seconds. Incoming messages are placed in a queue until it is processed. Every peer has a maximum queue size; if the queue is full, incoming messages are discarded. A peer which receives a message, uses the message's TTL to calculate the distance (in hops) to the origin. Obviously, if a duplicate arrives, it is ignored and the distance is not computed.

The average number of hops is shown in Table II. Contrary to our intuition, the average number of hops for DCMP is smaller than Gnutella, although the network contains fewer connections. To understand this, assume there is a path from peer A to B consisting of several hops, and there is a shorter path which goes through another peer C. Let A send a message msg and let C be overloaded. When msg reaches C it will be delayed. In the meanwhile, msg reaches B, and B calculates its distance from A. Eventually msg will be propagated by C towards B, where it will be rejected as duplicate. Therefore, the longer path is observed.

TABLE II
AVERAGE NUMBER OF HOPS IN STATIC NETWORKS

	Average Hops
DCMP	2.8
Gnutella	3.9

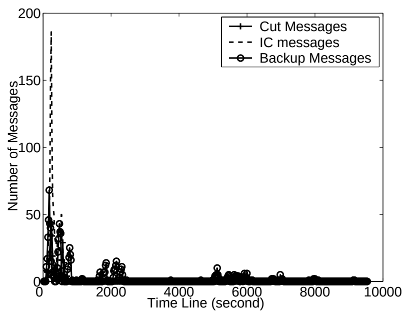
To verify this behavior, in Fig. 16(a) we show the average queue size in the peers versus the elapsed time. Larger queue size indicates that there will be longer delays before a message can be propagated. Gnutella experiences a much larger queue size on average compared to DCMP. Although the collected data are noisy, the pattern is still apparent. The instability is mainly caused by the large number of duplicates flooding the network. As we already discussed, most duplicates will arrive at the powerful peers, which will be overloaded. Since the shorter paths are congested, messages follow longer paths, thus increasing the average number of hops. In DCMP, on the other hand, most duplicates are eliminated (especially for high-degree peers); therefore, queues are smaller allowing messages to travel through the shortest path.

For demonstration purpose we also tested a lightly loaded environment by changing the mean time between queries to 200 seconds. In this case, the average hop number of Gnutella was marginally better than DCMP. Note that such a low query frequency is unlikely to be observed in practice. This is because in an existing P2P system the previous discussion would concern the super-peer layer, where each super-peer handles all the queries of its children.

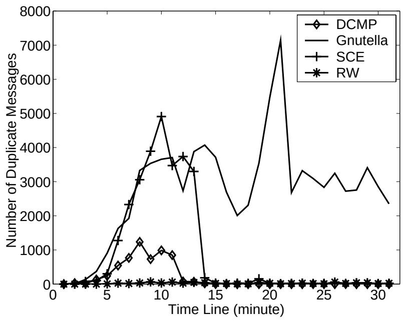
3) Overhead Analysis: DCMP introduces overhead in the form of control messages. There are two main types of such messages: the IC and the CM message. Also, GatePeers use *Backup Messages* to broadcast their backup GatePeers. Additionally, GatePeers and transitive peers perform message tagging periodically. While in this case DCMP does not transmit a new message but only appends a few bytes of information in existing messages, for simplicity we consider the entire tagged message as overhead.

Using the settings of the previous experiment, we counted the overhead due to control messages. The results are presented in Fig. 16(b), where the x-axis corresponds to the elapsed time. Initially, most of the overhead is IC messages. These are generated when a peer detects a duplicate. Therefore, numerous IC messages indicate the existence of many cycles in the network. Observe that

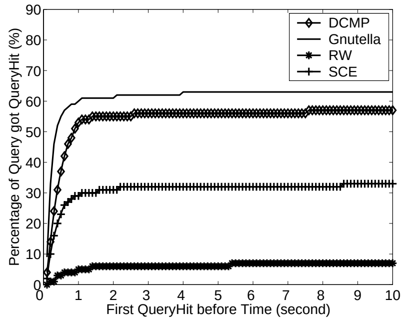
[7]Recall from Fig. 1(b) that duplicates account for more than 50% of the total messages.



(a) Control messages of DCMP

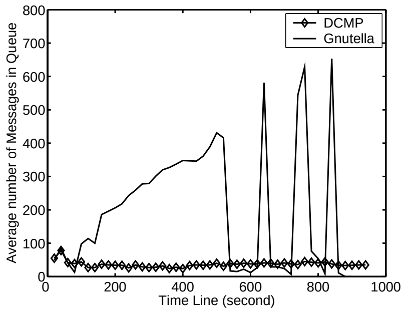


(b) Duplicate queries

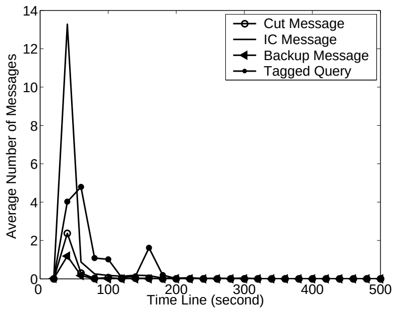


(c) Delay of first answer

Fig. 17. Analysis of DCMP control messages, duplicate queries and real-time delay for Gnutella, DCMP, SCE, RW



(a) Average message queue size vs. elapsed time



(b) Overhead of control messages (DCMP)

Fig. 16. Average queue size and overhead analysis

there are also many tagged queries, since GatePeers tag the query messages very frequently when the cycles are just cut. After a while, when many cycles have been eliminated, the number of IC and tagged messages drops significantly. Moreover, the overhead due to CM and backup messages is minimal. Initially, the total overhead is around 20 messages per peer. This number accounts for 10-20% of the total network traffic. This overhead becomes very insignificant when most of the cycles are eliminated; in practice, this is achieved after a couple of minutes. Then the overhead corresponds to 1-2% of the total traffic. The overhead is acceptable, considering the large number of duplicates which are avoided.

B. Dynamic Peers

For the next set of experiments, we deployed a dynamic P2P system on PlanetLab. Initially, the seed peers join the network and the coordinator starts the experiment; then, other nodes can join or fail/quit. The lifespan of the nodes follows the exponential distribution with mean equal to 90 min [24]. First, we consider a lightly loaded system, where peers initiate queries every 100 to 200 sec with uniform distribution; we examine heavier loads in the next section.

Previous work [17] states that the lifespan of super-peer architectures follows the Pareto distribution. This implies that our GatePeers should have a lifespan of several days [21]. Due to the instability of some PlanetLab nodes, however, we were not able to sustain the experimental environment for so long. Therefore, we chose the exponential distribution, which causes GatePeers to fail faster and allows us to investigate the behavior of DCMP under such failures. We stretch that the exponential distribution represents the worst case for our protocol. In practice, we expect less GatePeer failures, hence better overall performance.

In Fig. 17(a) we present the overhead due to control messages in the dynamic environment. We do not show the tagged messages since they follow largely the IC messages. Compared to the static case (i.e., Fig. 16(b)), more control messages are required since new cycles are introduced. For example, except from the initial period, we observe two peaks at around 2000 and 6000 sec. During these periods, it happened that some GatePeers and all their backup GatePeers failed. Therefore, many peers needed to connect to alternative GatePeers, possibly by re-joining the network. In such a process, it is possible to introduce new cycles (e.g., large cycles may become shorter and detectable). The total overhead accounts for 125 messages in these two periods. Observe that the total overhead of DCMP is 2-3 orders of magnitude less that the number of duplicates it avoids during the whole run (see next experiment); therefore, the overall traffic reduction is significant.

1) Comparison with Other Techniques: We also implemented two more techniques which may potentially reduce the duplicate messages in Gnutella-like networks: (i) The Simplistic Cycle Elimination (SCE) technique (similar to LimeWire), and (ii) Random Walks (RW), with $TTL = 50$. In Fig. 17(b) we show that all three methods (i.e., RW, SCE, DCMP) can reduce the number of duplicates compared to Gnutella. RW appears to be the most efficient one, especially at the initial period where there are a lot of cycles. This is because RW only forwards the query to one connection at each time and the overall messages are reduced, resulting in fewer duplicates. Observe that DCMP is the second best.

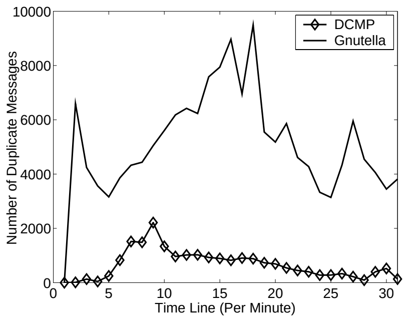
Note that the lower number of duplicates is only an indication that the load of the network is reduced, and should not affect the user's experience. To evaluate this, we measure the delay from the moment a peer initiates a query, until it receives the first query hit (i.e., answer to the query). The results are shown in Fig. 17(c). The x-axis corresponds to the delay since the initiation of the query. The y-axis represents the cumulative percentage of queries which received hits. For example, in DCMP, $x = 1$ corresponds to $y = 51\%$ meaning that 51% of the queries received at least one answer within one second. Queries expire after 5 min; any results arriving after the timeout period, are discarded. Gnutella performs better among the four methods, whereas DCMP follows closely. The reason for the slightly larger delay is twofold: first, the initial overhead of the cycle elimination messages affects DCMP; second, as DCMP disables some connections, the number of answer messages routed through high-degree peers increases, resulting to longer delays. For the SCE technique, note that only 30% of the queries received at least one hit before expiring. This is because peers disable connections based only on local information; thus, the network may break into fragments. Also note that RW can greatly reduce the system's workload by sending one copy of the query each time, but it explores only a small part of the network. The low coverage influences the ability to return answers. In the experiment, only about 7% of the queries receive some answer before the timeout. Increasing the TTL value can increase the coverage, but the delay will increase as well. Besides, there will be more duplicates since RW cannot avoid cycles.

RW and SCE reduce the number of duplicates at the expense of response time. To further investigate the quality of the search operation, we counted the total number of query hits before the query timeout. Recall that every query can be satisfied by 5% of the peers. Since peers enter and quit the network continuously, there are around 320-350 of them on-line concurrently at any given time. Therefore, in the best case each query should return around 15 results. Of course, this is impossible in practice due to small TTL, nodes failing while processing a query, delays longer than the timeout, etc. Still, a larger number of hits indicates better quality of service. In Table III we show the average number of hits per query; DCMP provides the best results.

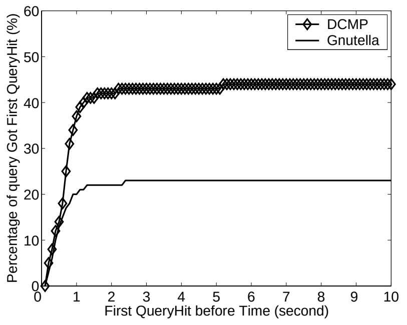
TABLE III
AVERAGE NUMBER OF QUERYHIT IN DYNAMIC NETWORKS

	Average Number of Query Hits
DCMP	7.7
Gnutella	6.9
RW	1.2
SCE	6.1

For the previous experiments the query frequency initiation was set to one query per 100-200 sec; this corresponds to a very lightly loaded network. In our final experiment, we investigate the effect of increasing the frequency to one query per 50 sec. Again, we count the number of duplicates and measure the delay until the first query hit. The results are presented in Fig. 18. DCMP generates much fewer duplicates than Gnutella. Moreover, since the network traffic has increased, the overhead of duplicates becomes more obvious and DCMP outperforms Gnutella in terms of delay.



(a) Duplicates in dynamic heavy loaded networks



(b) Delay in dynamic heavy loaded networks

Fig. 18. Duplicate queries and real-time delay for heavy loaded networks

VII. Conclusions

In this paper we presented DCMP, a protocol for distributed cycle minimization in broadcast-based P2P systems. DCMP preserves the low diameter of Gnutella-like networks while eliminating most of the duplicate messages. Moreover, the overhead due to control messages is minimal. This results in reduced response time, which in turn increases the scalability of the system. Our protocol is suitable for dynamic networks, since it handles peer joins/departures efficiently and is resilient to failures. DCMP is also designed to be as simple as possible and is independent of the search algorithm. Therefore, it can be implemented on top of popular P2P systems such as Gnutella, Kazaa or Gia with minimal effort. We used a simulator and a prototype implementation on PlanetLab to verify that our techniques are applicable to realistic environments. The initial results are very promising. In the future we plan to further improve our protocol by considering other factors, such as maintaining statistics of peers for a more stable and robust network. We also plan to investigate the possibility of employing DCMP outside the P2P area; for instance, in sensor networks.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proc. of ACM SIGCOMM*, 2001, pp. 161–172.
[2] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.
[3] Gnutella, http://www.gnutella.com/ and http://groups.yahoo.com/group/the_gdf/.
[4] Kazza, http://www.kazaa.com/.
[5] M. Ripeanu, A. Iamnitchi, and I. T. Foster, "Mapping the Gnutella Network," *IEEE Internet Computing*, vol. 6, no. 1, pp. 50–57, 2002.
[6] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," in *Proc. of Int. Conf. on Supercomputing (ICS)*, 2002, pp. 84–95.
[7] PlanetLab, http://www.planet-lab.org/.
[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-Like P2P Systems Scalable," in *Proc. of ACM SIGCOMM*, 2003, pp. 407–418.

[9] B. Yang and H. Garcia-Molina, “Designing a Super-Peer Network,” in *Proc. of IEEE ICDE*, 2003, pp. 49–60.
[10] ——, “Improving Search in Peer-to-Peer Networks,” in *Proc. of IEEE ICDCS*, 2002, pp. 5–14.
[11] S. Bakiras, P. Kalnis, T. Loukopoulos, and W. S. Ng, “A General Framework for Searching in Distributed Data Repositories,” in *Proc. of IEEE IPDPS*, 2003, pp. 34–41.
[12] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K.-L. Tan, “An Adaptive P2P Network for Distributed Caching of OLAP Results,” in *Proc. of ACM SIGMOD*, 2002, pp. 25–36.
[13] K. Sripanidkulchai, B. Maggs, and H. Zhang, “Efficient Content Location Using Interest-Based Locality in P2P Systems,” in *Proc. of IEEE INFOCOM*, 2003, pp. 2166–2176.
[14] Limewire, http://www.limewire.com/.
[15] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, “A Distributed Approach to Solving Overlay Mismatching Problem,” in *Proc. of IEEE ICDCS*, 2004, pp. 132–139.
[16] X. Liu, Y. Liu, L. Xiao, L. M. Ni, and X. Zhang, “Location Awareness in Unstructured Peer-to-Peer Systems,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 163–174, 2005.
[17] F. Bustamante and Y. Qiao, “Friendships that Last: Peer Lifespan and Its Role in P2P Protocols,” in *Proc. of Int. W/shop on Web Content Caching and Distribution*, 2004, pp. 233–246.
[18] A. Singh, T.-W. J. Ngan, P. Druschel, and D. S. Wallach, “Eclipse Attacks on Overlay Networks: Threats and Defenses,” in *Proc. of IEEE INFOCOM*, 2006, pp. 120–130.
[19] N. Daswani and H. Garcia-Molina, “Query-flood DoS Attacks in Gnutella,” in *Proc. of ACM CCS*, 2002, pp. 181–192.
[20] C. Gkantsidis, M. Mihail, and A. Saberi, “Random Walks in Peer-to-Peer Networks,” in *Proc. of IEEE INFOCOM*, vol. 1, 2004, pp. 120–130.
[21] S. Saroiu, P. Gummadi, and S. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems,” in *Proc. of Multimedia Computing and Networking*, 2002, pp. 156–170.
[22] C. R. Palmer and J. G. Steffan, “Generating Network Topologies that Obey Power Laws,” in *Proc. of IEEE GLOBECOM*, 2000, pp. 434–438.
[23] Statistics, Gnutella-like Networks, http://www.limewire.com/english/content/uastats.shtml.
[24] N. Christin, A. S. Weigend, and J. Chuang, “Content Availability, Pollution and Poisoning in File Sharing P2P Networks,” in *Proc. of ACM Electronic Commerce*, 2005, pp. 68–77.

PLACE
PHOTO
HERE

Zenzhou Zhu received the BS and MS degrees in Computer Science from the National University of Singapore in 2005 and 2007, respectively. His research interests are in the areas of peer-to-peer networks, grid computing, and distributed database systems.

PLACE
PHOTO
HERE

Panos Kalnis received his Diploma in Computer Engineering from the Computer Engineering and Informatics Department at the University of Patras, Greece, and his PhD from the Computer Science Department at the Hong Kong University of Science and Technology. In the past he was involved in the designing and testing of VLSI chips at the Computer Technology Institute, Greece. He also worked in several companies on database designing, e-commerce projects and web applications. Currently, he is an Assistant Professor in the Computer Science Department at the National University of Singapore. His research interests include peer-to-peer systems, mobile computing, OLAP, data warehouses, spatial databases, and anonymity.

PLACE
PHOTO
HERE

Spiridon Bakiras received his BS degree (1993) in Electrical and Computer Engineering from the National Technical University of Athens, his MS degree (1994) in Telematics from the University of Surrey, and his PhD degree (2000) in Electrical Engineering from the University of Southern California. Currently, he is an Assistant Professor in the Department of Mathematics and Computer Science at John Jay College, CUNY. Before that, he held teaching and research positions at the University of Hong Kong and the Hong Kong University of Science and Technology. His research interests include high-speed networks, peer-to-peer systems, mobile computing, and spatial databases. He is a member of the ACM and the IEEE.