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 (for example, Gnutella) and two-layer superpeer implementations (for example, 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. Although 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 the Distributed Cycle Minimization Protocol (DCMP), a dynamic fully decentralized protocol that significantly reduces 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 significantly improves 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.

1 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: 1) Hash-based systems (for example, CAN [1] and 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. 2) Broadcast-based systems (for example, 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 worldwide 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 superpeers (for example, Kazaa [4]), since the superpeer layers resemble Gnutella-style protocols.

Assume the network topology in Fig. 1a 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 seven or eight) 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 a power-law topology [5]; there is a small number of peers with many neighbors (A in our example), whereas 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 this claim, we deployed a 3,000-node Gnutella-style power-law network and counted the number of duplicates, useful messages, and total messages (see Section 5 for details). The data are sorted by total messages first and then by useful messages, as shown in Fig. 1b. Nodes appear in descending workload order; therefore, $x = 0$ corresponds to the node that receives the most messages. It is clear from the graph that a large

• 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.

Manuscript received 6 Feb. 2007; revised 10 May 2007; accepted 29 May 2007; published online 3 July 2007.
Recommended for acceptance by C. Shahabi.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0042-0207.
Digital Object Identifier no. 10.1109/TPDS.2007.70732.

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.

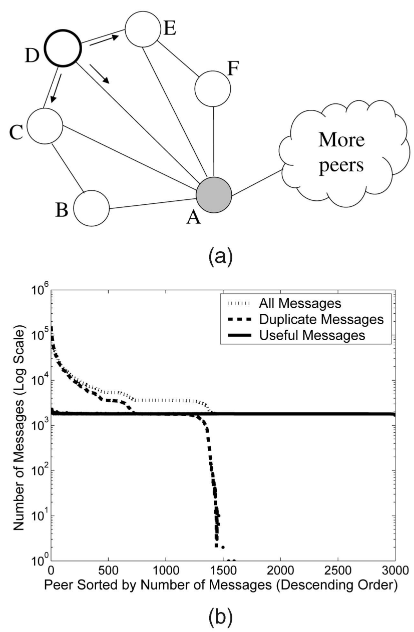


Fig. 1. Cycles introduce numerous duplicate messages. (a) Example network. (b) Total versus duplicate messages.

proportion of the transmitted messages are duplicates that will be ignored; similar results appear in [6]. Observe that several low-degree nodes (that is, 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 (that is, 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 severely affect 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 that 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 a GatePeer via a distributed process, the system is resilient to failures.

The main characteristics of DCMP are the following:

1. It reduces duplicate messages by as much as 90 percent.
2. It requires few control messages; therefore, the overhead is minimal.
3. 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.
4. There is a trade-off 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 superpeer network topologies. We also implemented a prototype, which was deployed on PlanetLab [7]. Our experiments indicate that DCMP achieves a 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 2 presents the related work. Next, in Section 3, we describe the main aspects of DCMP, whereas in Section 4, we discuss how DCMP deals with dynamic networks. In Sections 5 and 6, we present the experimental results from our simulation and the PlanetLab implementation, respectively. Finally, Section 7 concludes the paper and discusses directions for future work.

2 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 that 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. Reference [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: 1) Iterative Deepening, where multiple BFTs are initiated with successively larger depths, 2) 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, and 3) 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 [11] and [12], where the network is reconfigured dynamically based on the query statistics. A similar technique, called Interest-based Locality [13], directly contacts 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.

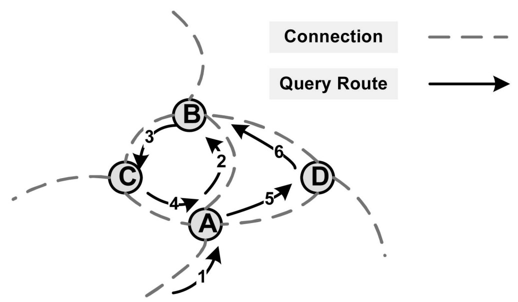


Fig. 2. Example of duplicate messages in Gia.

Limewire [14] maintains a table where it stores the IDs of duplicate messages and the directions (that is, neighbor peers) from where they arrive. Once a message is identified as a 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 that disables those connections from where "a lot" of duplicates are arriving. In practice, it is difficult to unambiguously define 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 1-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 (RWs), 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 in Fig. 2, where the order of the peers based on capacity is A, B, C, and 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 that 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 that cause duplicates.

In order to reduce the unnecessary traffic (that is, 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 that 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 that are physically far away (for example, 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 (for example, CAN [1] and 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.

3 Protocol Design

In this section, we describe our protocol in detail 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 (that is, 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 traveled 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: $(GUID(msg), B \rightarrow A)$.

3.1 Simplistic Cycle Elimination (SCE)

To motivate our approach, here, we describe a straightforward method for eliminating cycles and explain its drawbacks. Consider Fig. 3a and let peer B receive the same message msg from A and A'. B identifies msg as a 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.

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.

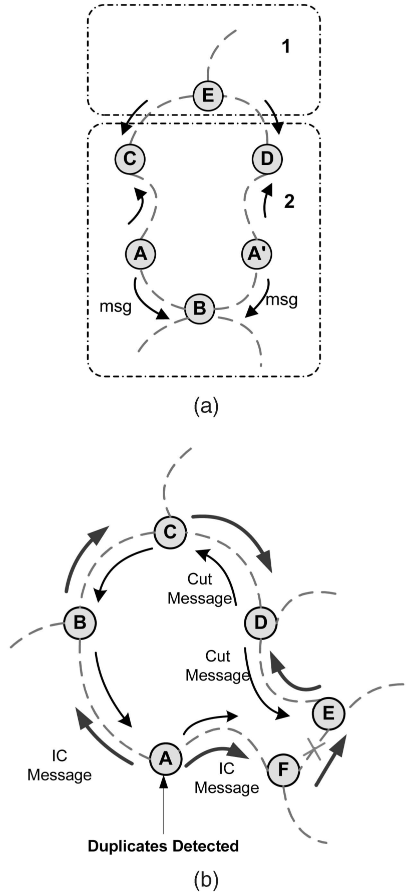


Fig. 3. Cycle elimination methods. (a) SCE. (b) DCMP.

3.2 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 that receives a duplicate. Fig. 3b provides an example. Assume that peer A receives a message msg from $B \rightarrow A$, and, soon after, it receives the same message[2] (that is, same $GUID$) from $F \rightarrow A$. Peer A identifies msg as a 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 that 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

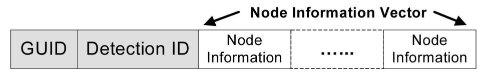


Fig. 4. Structure of the IC message.

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.

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 that propagated the IC message. This includes the bandwidth of each peer, the processing power, the IP address, and the topology information about the peer's degree and its neighbors. In our example, the NIV of icm initially contains information only about peer A.

Peer A sends one copy of icm toward $A \rightarrow B$ and another toward $A \rightarrow F$. Each peer that receives icm appends its own information to the NIV field and then treats icm similar 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' farther 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 toward $D \rightarrow D'$. 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 (for example, node failure), it is possible that no peer receives a duplicate icm. In this case, icm is simply propagated toward the origin of msg. We could avoid both cases by using a more complicated protocol. However, TTL is between three and seven 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 trade-off

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 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 CM.

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 5, 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 that 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 CM toward $D \rightarrow C$ and $D \rightarrow E$, respectively. These are the reverse directions from where icm arrived previously. Similarly, CMs received by any peer are propagated toward the reverse path of the corresponding IC. Eventually, the CM will reach either E or F, and one of these peers will cut the connection, thus eliminating the cycle. The algorithm for handing CMs is presented in Fig. 6.

Observe that D could initiate only one copy of the CM to traverse the cycle. The reason for sending two copies is threefold: 1) 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 CMs, thus complicating the protocol. 2) The delay until cutting the cycle is minimized since the average number of hops for CMs is reduced. 3) The total number of transmitted messages is the same since the CM carries useful information for all the peers and must traverse the entire cycle, as we will discuss in the next section.

3.3 Deciding the Cutting Position

Here, we explain how we choose the connection to disable in order to cut a cycle. This decision is made at the peer that receives two copies of the same IC message (that is, 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 that arrive at the coordinator have gathered this information.

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.

The following definitions are necessary:

Definition 1 (opposite edge). *Let S_N be the set of nodes that 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*

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

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

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

where $\mathcal{B}$ is the bandwidth, $\mathcal{C}$ is the CPU processing power, $\mathcal{D}$ is the peer's degree (that is, the 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 that accepts many neighbors is beneficial for low network diameter. There are several other factors that can influence the characteristics of the network. For example, Bustamante and Qiao [17] suggest that the distribution of the lifespan of peers follows the Pareto distribution and propose 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 the GatePeer.*

The heuristic we use in our protocol is to cut cycles by disabling the connection that 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 that 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 that 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 CMs. 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.

3.4 Disseminating GatePeer Information

GatePeers assist to recover from node failures and are used as entrance points in a dynamic network (refer to Section 4.1); 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*. Although 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 trade-off between cost and efficiency: for a period of 1 minute after a new GatePeer is elected, a message is tagged every 5 seconds; after that, the tagging frequency is lowered to 1 message every 10 minutes. Note that the exact values are not crucial, and the overhead of tagging is small (refer to Section 6.1.3 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. 8a, where B and 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 that is not a GatePeer or a transitive peer is called a *normal* peer. Normal peers may receive tagged messages from multiple GatePeers (or transitive peers), but all come from the same direction. In the example in Fig. 8a, peer H receives tagged messages from D and F, but all arrive through $E \rightarrow H$. We 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 and 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 (that is, $N \rightarrow M$) is the* primary direction *of N.*

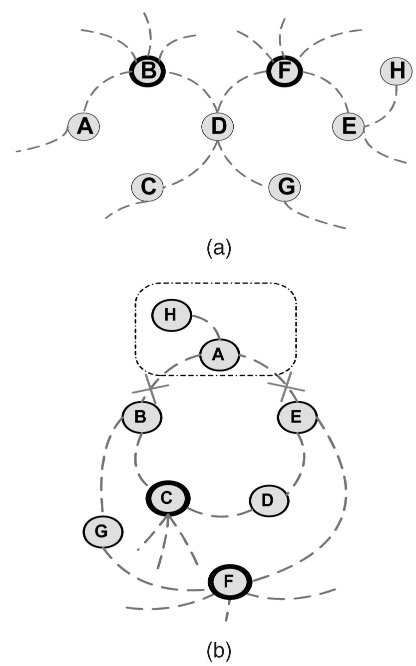


Fig. 8. Transitive peers and network splits. (a) Example of transitive peer D. (b) Example of a network split in DCMP.

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.

3.5 Concurrent Cycle Elimination

In Section 3.1, 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. 8b, 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),

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

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

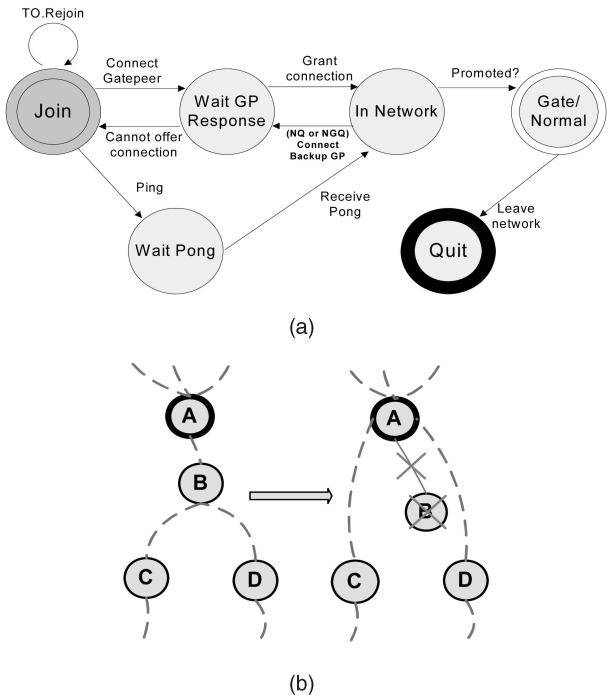


Fig. 9. Join and quit in dynamic networks. (a) State diagram to handle node quit/join. GP is GatePeer. NGQ means Neighbor GatePeer Quit. NQ means that the departing node is in the primary direction of a peer. TO: time-out period. (b) Failure of normal peer B.

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 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. 9a.

4.1 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 does it use the ping/pong protocol.

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.

4.2 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 (that is, only 1 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 that 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 (that is, the direct neighbors of G) are affected, and the network topology does not change significantly. If, for some reason, this process is not successful (for example, none of the backup GatePeers can accept more connections because of simultaneous GatePeer failures), then the affected peers simply rejoin the network using the peer arrival procedure described above.

4.3 Departure of a 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 in Fig. 9b. 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 toward 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.

5 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

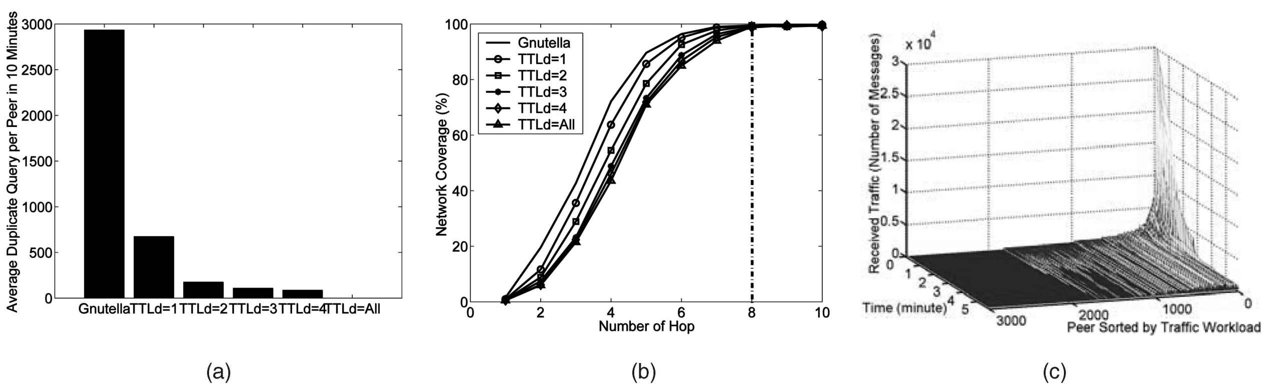


Fig. 10. Analysis of traffic workload and effect of using different TTL_d (static networks). (a) Duplicate messages. (b) Connectivity. (c) Received traffic (DCMP).

layer. The simulator is written in C++ and was executed on a Linux machine (with a 3.0-GHz CPU and 3 Gbytes of RAM). We used a power-law topology with an average degree of 3.4, whereas the network size varied from 500 to 10,000 peers (results are based on 3,000 nodes by default in this section). The bandwidth of each peer ranged from 56 Kbps (that is, modem) to 45 Mbps (that is, T3 connection), following also a power-law distribution. The TTL for the messages was set to eight (except for the RW algorithm). Peers initiated queries with a uniform distribution and a mean query frequency of 3.6 queries per peer per hour. Each experiment was executed with six different seeds, and the results show the average of all runs.

5.1 Topology and Workload Analysis

In the first set of experiments, we generate a power-law network with 3,000 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. 10a, 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 that 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 $TTL_d = 2$ (that is, more than 90 percent of the duplicate messages are eliminated). Further increasing TTL_d does not result in a significant improvement.

However, there is a trade-off 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. 10b, we present the average connectivity of the network for varying TTL_d. For instance, if $TTL_d = 1$, a message can reach almost 65 percent of the peers in the network within 4 hops on the average; however, if $TTL_d = All$ (that is, tree topology), messages can reach only 43 percent of the peers. From the two diagrams in Fig. 10, we conclude that $TTL_d = 2$ provides a good trade-off between the number of duplicates and connectivity; therefore, we use this value for the following experiments.

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

5.2 Influence of Network Size

In this experiment, we vary the number of peers in the network. Fig. 11a 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 coverage almost as good as that by Gnutella.[6] In Fig. 11b, 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 a length larger than $2 \cdot TTL_d$ (that is, the ones that introduce duplicates in DCMP) is small.

5.3 Symmetric Cut versus 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. 12a, where we draw the network coverage for varying numbers of hops. By cutting cycles symmetrically to GatePeers, DCMP manages to closely follow 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 percent of the peers are reachable within 8 hops.

5.4 Failure and Attack Analysis

In P2P 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,

6. The default $TTL = 8$. For other TTL values, the graph follows the difference between the Gnutella and $TTL_d = 2$ lines in Fig. 10b.

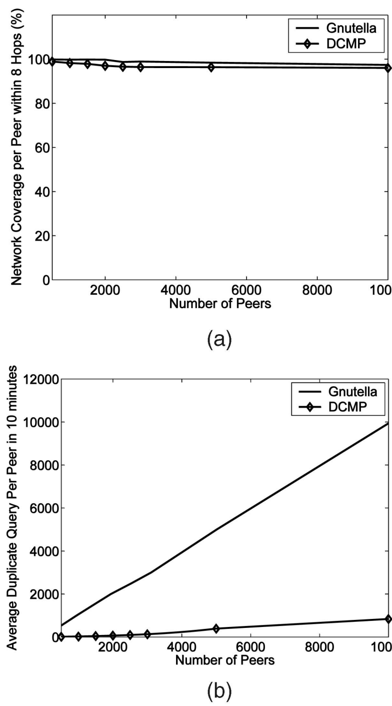


Fig. 11. Scalability for Gnutella and DCMP (static networks). (a) Connectivity. (b) Duplicate query.

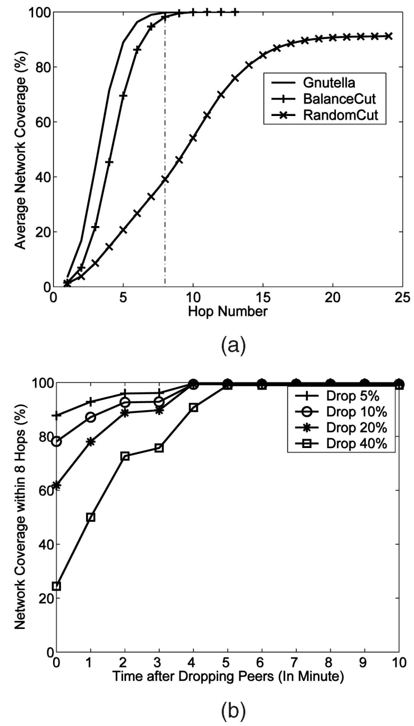


Fig. 12. Comparison of symmetric/balance cut and random cut and failure analysis. (a) Balance versus random cut. (b) Random failure analysis.

we force 5-40 percent of all peers to fail simultaneously. All peers (that is, 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 minutes). By utilizing the backup and referred GatePeer information, the network fragments can connect to each other efficiently even when 40 percent of the peers fail at the same time. Fig. 12b shows that the network coverage restores to almost 100 percent 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 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 rejoin 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 percent. The coverage of the network dropped to around 20 percent, and the system needed around 5 minutes to recover (very similar to Fig. 12b). 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, for example, detecting malicious peers [18] or defending against DoS attacks [19]. This is possible, since these protocols work independently of each other.

5.5 Comparison with Other Approaches

Lv et al. [6] use RWs 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 trade-offs 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 that have the maximum delay. In LTM, the following two steps are performed at each peer: 1) Forward a detection message. If a detection message (received or self-created with initial $TTL = 2$) has not expired, the peer inserts a new time stamp and broadcasts the message to the neighbor peers. 2) 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 time stamps to calculate delays.

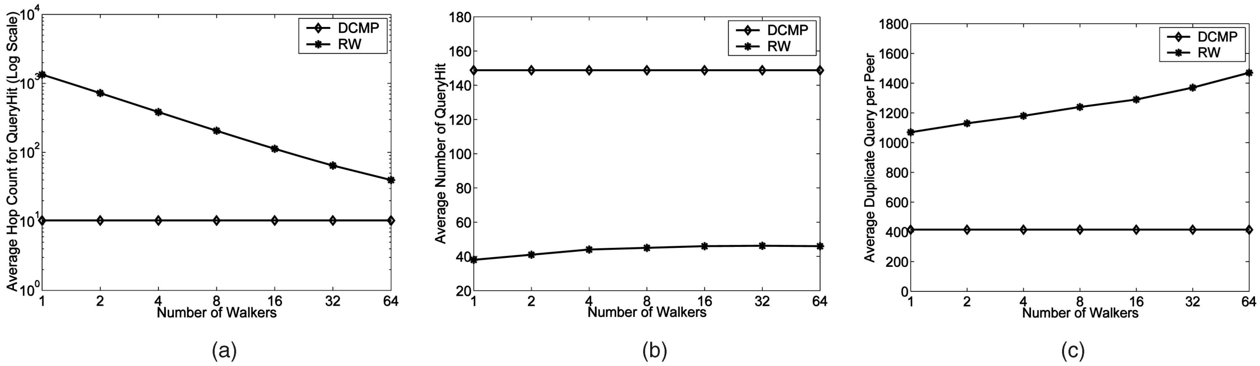


Fig. 13. Comparison of RWs and DCMP (dynamic network). (a) Path lengths of a query hit. (b) Number of query hits. (c) Number of duplicate queries.

5.5.1 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 [20]). For LTM, we followed the optimal frequency of broadcasting detection messages suggested in [16]. In Fig. 13, we compare RW and DCMP. First, the average delay is shown in Fig. 13a (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. 13b shows that DCMP can find almost all of them, but RW discovers less than 33 percent of the copies. Finally, Fig. 13c 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 1 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 a lower delay, returns more results, and decreases the number of duplicate messages by 22 percent, compared to LTM. Furthermore, DCMP generates much less overhead than LTM, as we will explain in the next experiment.

5.5.2 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 3,000 online peers and placed 3,000 additional peers in a waiting list. When the session time of an online 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 online 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 3,000 online peers to have had the chance to quit and reenter.

Compared to LTM, DCMP has much smaller overhead (that is, control messages), which is due to the fact that LTM adopts an "eager" approach (that is, broadcasts control messages periodically), whereas DCMP adopts a "lazy" one. As shown in Fig. 14a, LTM's overhead is one to two orders of magnitude greater than that of DCMP (notice that

TABLE 1
Comparison of RWs, 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

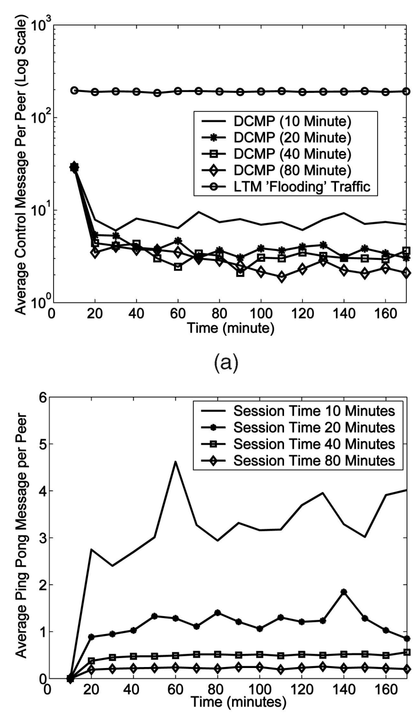


Fig. 14. Overhead analysis and effect of session time on control and ping/pong messages. (a) Overhead analysis. (b) Ping and pong messages.

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. 14b 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 than 10 minutes (this number is consistent with most of the observations in the literature, for example, [21]), the joining overhead for DCMP is reasonably small.

6 Prototype Evaluation on PlanetLab

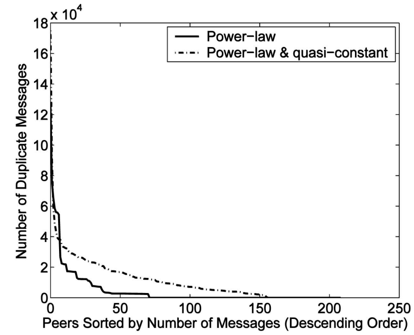
We implemented DCMP in a prototype and deployed it on PlanetLab [7]; our prototype implements all the features except the downloading of files after they are located. There are 665 nodes that are distributed over 315 locations in PlanetLab at the time of writing this paper. Unfortunately, some nodes 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 that appear in real-life P2P networks [5] in order to test DCMP: 1) One is a power-law topology with an average degree of 3.4. We used the PLOD [22] method to construct the network. This topology reflects the original Gnutella network (that is, protocol v0.4). 2) The other is a two-layer network with power-law distribution at the superpeer layer and quasi-constant distribution at the leaf layer. This topology corresponds to the latest version of the Gnutella protocol (that is, v0.6). We used statistics from Limewire [23] to generate a realistic network.

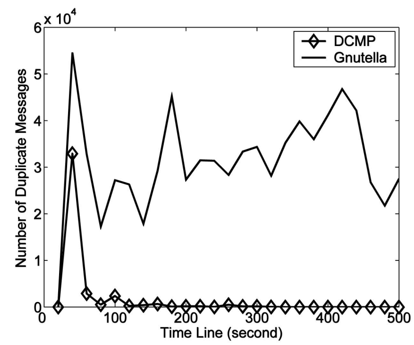
In our experiments, we use a small set of five nodes as the network seed. The IP addresses of the seed nodes are known to all peers. The seeds are used as entry points to propagate ping messages in order to assist other nodes to join. We also use a coordinator peer that transmits configuration parameters 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 superpeer P2P networks (for example, Kazaa). We also evaluate our protocol against the SCE technique (similar to the approach suggested by Limewire). Finally, we compare DCMP with RWs. We use the following metrics:

1. *Number of duplicate messages.* This metric indicates how much unnecessary traffic is eliminated.
2. *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 percent 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.
3. *DCMP overhead.* These are the control messages (IC, CM, and message tagging), which are essential in our protocol.



(a)



(b)

Fig. 15. Duplicate distribution and timeline of duplicates. (a) Duplicate distribution in Gnutella and super-peer architectures. (b) Number of duplicates versus elapsed time for DCMP and Gnutella.

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 (that is, peers do not enter/leave). Next, we deploy a realistic dynamic network and measure the actual delay perceived by the users.

6.1 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.

6.1.1 Duplicates Analysis

In Fig. 15a, we analyze the duplicates' distribution in two topologies: power law (that is, Gnutella) and power law quasiconstant (that is, superpeer architectures). The x-axis represents individual nodes appearing in descending workload order; therefore, $x = 0$ corresponds to the node that receives the most duplicates.[7] Both topologies are prone to a large number of duplicates; however, the two-layer network

7. Recall from Fig. 1b that duplicates account for more than 50 percent of the total messages.

TABLE 2
Average Number of Hops in Static Networks

	Average Hops
DCMP	2.8
Gnutella	3.9

suffers most. In two-layer architectures, about 10 percent of the nodes are superpeers [23] having a large number of neighbors that are also superpeers. Therefore, cycles are formed with high probability, and they introduce numerous duplicate messages.

Fig. 15b 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 second 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, continuously generates 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 superpeer architectures.

6.1.2 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 that 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.

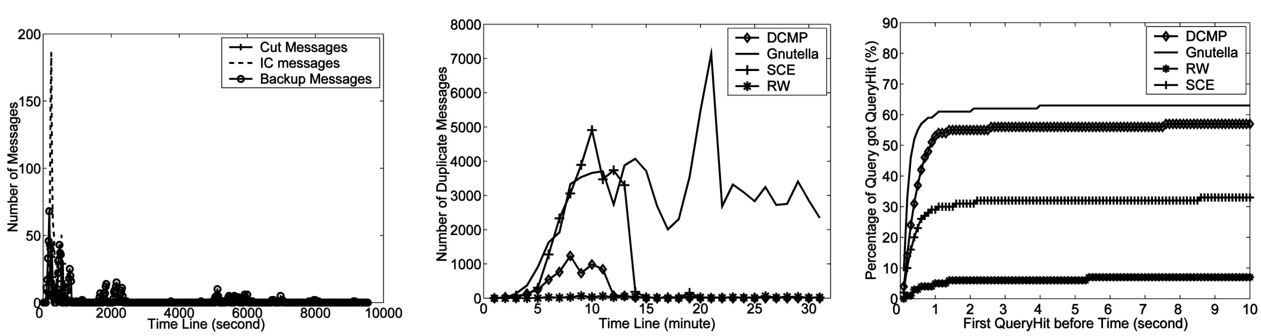
The average number of hops is shown in Table 2. Contrary to our intuition, the average number of hops for DCMP is smaller than that for Gnutella, although the network contains fewer connections. To understand this, assume that there is a path from peer A to B consisting of several hops, and there is a shorter path that 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 toward B, where it will be rejected as a duplicate. Therefore, the longer path is observed.

To verify this behavior, in Fig. 16a, we show the average queue size in the peers versus the elapsed time. A larger queue size indicates that there will be longer delays before a message can be propagated. Gnutella experiences a much larger queue size on the 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 purposes, 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 superpeer layer, where each superpeer handles all the queries of its children.

6.1.3 Overhead Analysis

DCMP introduces overhead in the form of control messages. There are two main types of such messages: the IC message and the CM. Also, GatePeers use *Backup Messages* to broadcast their backup GatePeers. Additionally, GatePeers and transitive peers perform message tagging periodically. Although, 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.



(a) (b) (c)

Fig. 16. Average queue size and overhead analysis. (a) Average message queue size versus elapsed time. (b) Overhead of control messages (DCMP).

Using the settings of the previous experiment, we counted the overhead due to control messages. The results are presented in Fig. 16b, 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 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 CMs and backup messages is minimal. Initially, the total overhead is around 20 messages per peer. This number accounts for 10-20 percent 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 percent of the total traffic. The overhead is acceptable, considering the large number of duplicates that are avoided.

6.2 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 a mean equal to 90 minutes [24]. First, we consider a lightly loaded system, where peers initiate queries every 100 to 200 seconds with a uniform distribution; we examine heavier loads in the next section.

Previous work [17] states that the lifespan of superpeer 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 and, hence, better overall performance.

In Fig. 17a, 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 (that is, Fig. 16b), more control messages are required since new cycles are introduced. For example, except for the initial period, we observe two peaks at around 2,000 and 6,000 seconds. 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 rejoining the network. In such a process, it is possible to introduce new cycles (for example, 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 the next experiment); therefore, the overall traffic reduction is significant.

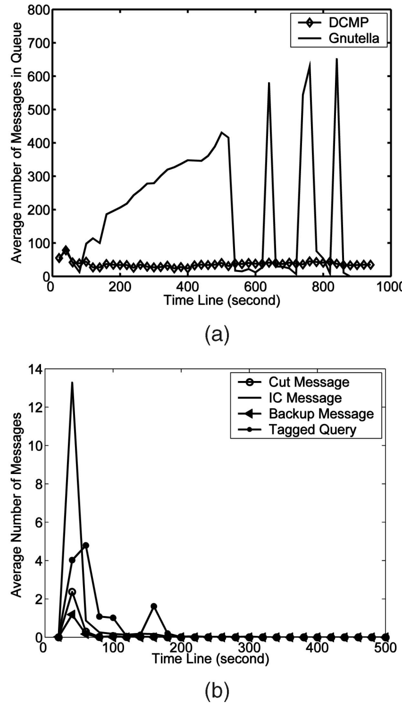


Fig. 17. Analysis of DCMP control messages, duplicate queries, and real-time delay for Gnutella, DCMP, SCE, RW. (a) Control messages of DCMP. (b) Duplicate queries. (c) Delay of the first answer.

6.2.1 *Comparison with Other Techniques*

We also implemented two more techniques that may potentially reduce the duplicate messages in Gnutella-like networks: 1) The SCE technique (similar to Limewire) and 2) RWs, with $TTL = 50$. In Fig. 17b, we show that all three methods (that is, RW, SCE, and 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 (that is, answer to the query). The results are shown in Fig. 17c. The x-axis corresponds to the delay since the initiation of the query. The y-axis represents the cumulative percentage of queries that received hits. For example, in DCMP, $x = 1$ corresponds to $y = 51$ percent, meaning that 51 percent of the queries received at least one answer within 1 second. Queries expire after 5 minutes; any results arriving after the time-out period are discarded. Gnutella performs best 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, and second, as DCMP disables some connections, the number of answer messages routed through high-degree peers increases, resulting in longer delays. For the SCE technique, note that only 30 percent of the queries received at least one hit before expiring. This is because

TABLE 3
Average Number of Query Hits in Dynamic Networks

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

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

Fig. 18. Duplicate queries and real-time delay for heavily loaded networks. (a) Duplicates in dynamic heavily loaded networks. (b) Delay in dynamic heavily loaded networks.

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 percent of the queries receive some answer before the time out. 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 the response time. To further investigate the quality of the search operation, we counted the total number of query hits before the query timed out. Recall that every query can be satisfied by 5 percent of the peers. Since peers enter and quit the network continuously, there are around 320-350 of them online 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 time out, etc. Still, a larger number of hits indicate better quality of service. In Table 3, we show the average number of hits per query; DCMP provides the best results.

For the previous experiments, the query frequency initiation was set to 1 query per 100-200 seconds; this corresponds to a very lightly loaded network. In our final experiment, we investigate the effect of increasing the frequency to 1 query per 50 seconds. 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.

7 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 the 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," *Proc. ACM SIGCOMM '01,* pp. 161-172, 2001.

[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 Trans. Networking,* vol. 11, no. 1, pp. 17-32, 2003.

[3] Gnutella, http://www.gnutella.com/ and http://groups.yahoo.com/group/the_gdf/, 2007.

[4] Kazaa, http://www.kazaa.com/, 2007.

[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," *Proc. ACM Int'l Conf. Supercomputing (ICS '02),* pp. 84-95, 2002.

[7] PlanetLab, http://www.planet-lab.org/, 2007.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-Like P2P Systems Scalable," *Proc. ACM SIGCOMM '03,* pp. 407-418, 2003.

[9] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," *Proc. 19th Int'l Conf. Data Eng. (ICDE '03),* pp. 49-60, 2003.

[10] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS '05),* pp. 5-14, 2002.

[11] S. Bakiras, P. Kalnis, T. Loukopoulos, and W.S. Ng, "A General Framework for Searching in Distributed Data Repositories," *Proc. 17th Int'l Parallel and Distributed Processing Symp. (IPDPS '03),* pp. 34-41, 2003.

[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," *Proc. ACM SIGMOD '02,* pp. 25-36, 2002.

[13] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in P2P Systems," *Proc. IEEE INFOCOM '03,* pp. 2166-2176, 2003.

[14] Limewire, http://www.limewire.com/, 2007.

[15] Y. Liu, Z. Zhuang, L. Xiao, and L.M. Ni, "A Distributed Approach to Solving Overlay Mismatching Problem," *Proc. 24th Int'l Conf. Distributed Computing Systems (ICDCS '04),* pp. 132-139, 2004.

[16] X. Liu, Y. Liu, L. Xiao, L.M. Ni, and X. Zhang, "Location Awareness in Unstructured Peer-to-Peer Systems," *IEEE Trans. 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," *Proc. Ninth Int'l Workshop Web Content Caching and Distribution (IWCW '04),* pp. 233-246, 2004.

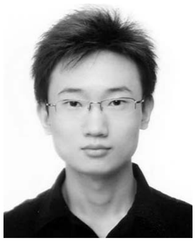
[18] A. Singh, T.-W.J. Ngan, P. Druschel, and D.S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses," *Proc. IEEE INFOCOM '06,* pp. 120-130, 2006.

[19] N. Daswani and H. Garcia-Molina, "Query-Flood DoS Attacks in Gnutella," *Proc. Ninth ACM Conf. Computer and Comm. Security (CCS '02),* pp. 181-192, 2002.

[20] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," *Proc. IEEE INFOCOM '04,* vol. 1, pp. 120-130, 2004.

[21] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking Conf. (MMCN '02),* pp. 156-170, 2002.

[22] C.R. Palmer and J.G. Steffan, "Generating Network Topologies That Obey Power Laws," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '00),* pp. 434-438, 2000.
[23] Statistics, Gnutella-Like Networks, http://www.limewire.com/english/content/uastats.shtml, 2007.
[24] N. Christin, A.S. Weigend, and J. Chuang, "Content Availability, Pollution and Poisoning in File Sharing P2P Networks," *Proc. Sixth ACM Conf. Electronic Commerce (EC '05),* pp. 68-77, 2005.



Zhenzhou 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.

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

Spiridon Bakiras received the BS degree in electrical and computer engineering from the National Technical University of Athens in 1993, the MS degree in telematics from the University of Surrey in 1994, and the PhD degree in electrical engineering from the University of Southern California in 2000. Currently, he is an assistant professor in the Department of Mathematics and Computer Science, John Jay College, City University of New York (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.