TOWARDS EFFICIENT PEER-TO-PEER NETWORKS: A DISTRIBUTED CYCLE MINIMIZATION APPROACH

ZHU ZHENZHOU

NATIONAL UNIVERSITY OF SINGAPORE

2007

TOWARDS EFFICIENT PEER-TO-PEER NETWORKS: A DISTRIBUTED CYCLE MINIMIZATION APPROACH

ZHU ZHENZHOU

(B.Sc.(Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2007

ACKNOWLEDGEMENTS

I wish to express my deep gratitude to my Supervisor, Dr. Panagiotis KALNIS, for his invaluable guidance and constant encouragement, without which the work of this thesis would not have been possible. Moreover, I enjoy the discussions that we had for various research papers, which improves the quality of this work.

I would also like to thank my thesis examiners, Associate Professors TAN Kian Lee and Dr. Stéphane Bressan, for their comments during my Graduate Research Paper presentation.

I thank my friends in Database Research lab for both the discussions and entertainments we had, which makes my stay at the university a pleasant experience.

To my parents, I would like to express my thanks, for their love and their encouragement to help me grow in all these years of my life.

Contents

Summary

Peer-to-Peer (P2P) technology has attracted a lot of attention, which is due to the fact that it simplifies the implementation of large, ad-hoc distributed repositories of digital information. There are two major categories of P2P architectures: (i) Hash-based systems, which assign a unique key to each file and forward queries to specific nodes based on a hash function. (ii) Broadcast-based systems, which use message-flooding to propagate queries. Such systems have been successfully employed in practice to form world-wide ad-hoc networks, due to their simplicity and versatility. In this thesis, we focus on broadcast-based P2P architectures.

The motivation of this work is that most real-life networks exhibit power-law topology; there is a small number of peers with many neighbors, while most peers have fewer neighbors. This causes a lot of cycles in the network. Especially there are many small cycles around high-degree peers, which introduces a lot of duplicate messages to them. While such messages can be identified and ignored, they still consume a large proportion of the bandwidth and other resources, which is likely to affect many other nodes since high-degree peers are the hubs of the network.

To deal with this problem, we propose a Distributed Cycle Minimization Protocol (i.e., DCMP), which improves search efficiency and is resilient to failures at

the same time. DCMP is 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. Our methods are also applicable to two-layer networks based on super-peers, since the super-peer layers resemble Gnutella-style protocols.

We evaluate proposed algorithms by both simulation and real-world deployment in PlanetLab platform, which confirms the improvements we achieve: reducing more than 90% of the duplicate messages and the network works efficiently in realistic environment. We also compare with several existing techniques to show that DCMP outperforms them in various aspects, e.g., traffic reduction, user-perceived delay and control message overhead.

List of Figures

List of Tables

Chapter 1

Introduction

The emergence of Peer-to-Peer (P2P) file sharing applications such as Napster [1], Gnutella [2] [3], and Kazza [4], has attracted millions of users. The success of P2P technique is due to the fact that 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.

Different from a traditional server-client model (e.g., FTP), peer nodes simultaneously function as both "clients" and "servers" in a pure P2P network. Early P2P systems usually adopt a centralized approach. Manageable central servers maintain the indexing information for the files which are shared by users. Central servers handle queries by selecting the peers who share the required resources. Downloading of the files is carried among peer themselves. A typical example of this kind of systems is Napster. Centralized approach creates a single point of failure which makes the systems vulnerable to denial of service attacks.

A lot of research has been done to eliminate reliance on central servers and to

provide greater freedom for participating users to exchange information and services directly between each other. There are two major categories of decentralized P2P architectures:

- Hash-based systems (e.g., Chord [5], CAN [6], BATON [7], P-Grid [8],Viceroy [9]), which assign a unique key to each file and forward queries to specific nodes based on a hash function. Although they guarantee the location of content within a bounded number of hops, they require tight control of the data placement and the topology of the network. These approaches usually use different network topology (e.g.,ring structure in Chord and butterfly in Viceroy), Ref. [10] analyzes various their graph-theory properties in details.

- Broadcast-based systems (e.g., Gnutella [2] [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 employed in practice to form world-wide ad-hoc networks, due to their simplicity and versatility. Here we focus on broadcast-based P2P architectures in this thesis.

1.1 Fundamentals of Gnutella Protocol

As we focus on unstructured P2P systems in this thesis, we first introduce fundamentals of the popular Gnutella protocol. Basic activities of a peer in Gnutella networks are illustrated in Figure 1.1. Gnutella operates on a query flooding method. Basic type of messages in all versions of Gnutella protocols are listed as follows:

- ping: discover hosts in network (e.g., E 'ping' to connect to the network)

- pong: reply to ping (e.g., A 'pong' to offer E a connection)

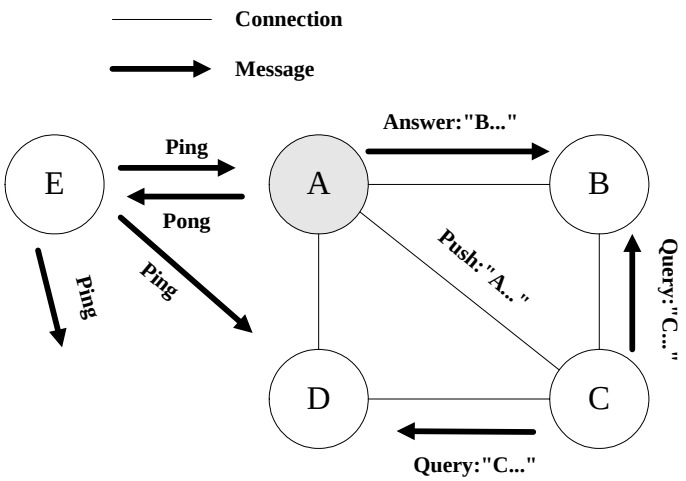


Figure 1.1 Example of message passing in a Gnutella network

- query: search for a file (e.g., C 'query' the network to search for content 'C')
- queryhit/answer: reply to query (e.g., A routes an 'answer' to B)
- push: download request if peer is behind a firewall (e.g., C 'push' A to start a download)

1.2 Motivation

Assume the network topology of Figure 1.2 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 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 received it by another path. If this is the case, it simply ignores the incoming message. We call this method Naïve

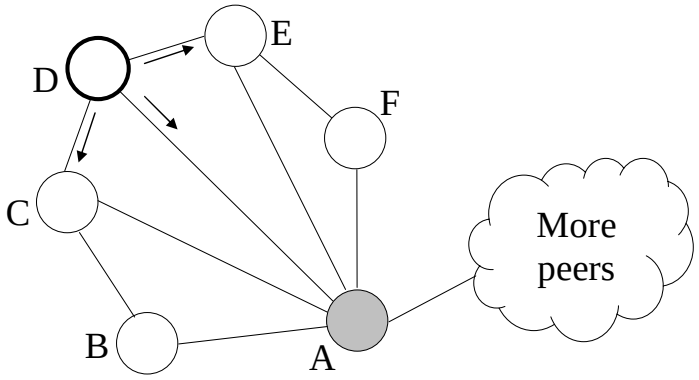


Figure 1.2 Example of duplicate messages in Gnutella networks

Duplicate Elimination (*NDE*).

The motivation of this project is that most real-life networks exhibit power-law topology [11]; 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.

To verify this claim, we simulated a 3000-node Gnutella-style power-law network and counted the number of duplicate versus the total messages. The results are shown in Figure 1.3. The x-axis represents individual nodes. 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. [12]. There are low-degree nodes (i.e., peers with few neighbors) which do not receive any duplicates because they do not participate in any cycle. We also conducted similar experiments with 300 nodes in PlanetLab [13] and similar patterns are observed. On the other

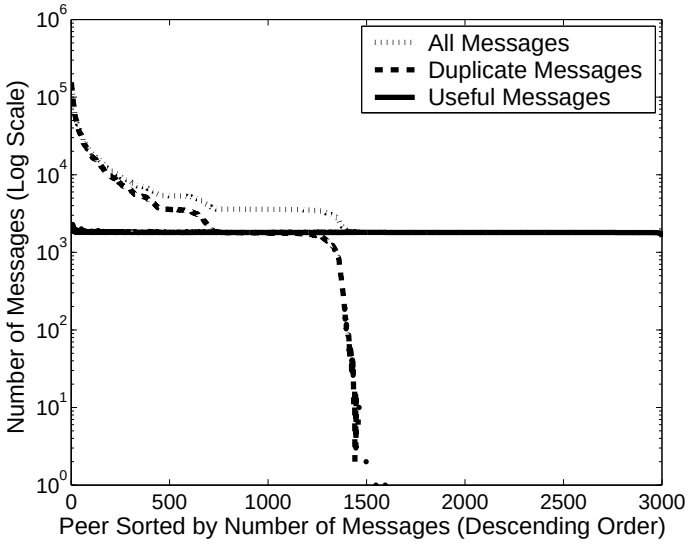


Figure 1.3 Total vs. duplicate messages

hand, our investigation revealed that the high-degree nodes (i.e., peers with many neighbors) receive most of the useless messages, since the probability of being involved in cycles is higher.

Duplicate messages consume bandwidth and system resources primarily from peers which are crucial for the connectivity of the network. Therefore, they affect severely the response time and scalability of P2P systems. In this thesis, we mainly describe the Duplicate Cycle Minimization Protocol (*DCMP*). Our protocol 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.

1.3 Our Contributions

The goal of this project is to explore various existing approaches in unstructured P2P networks and to find out efficient methods to improve the scalability of widely deployed P2P networks. The main characteristics of proposed protocol are:

- It reduces duplicate messages by as much as 90%.
- It requires few control messages, therefore the overhead is minimal.
- 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.
- 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 splits. As a result, the connectivity and average path length remain relatively unaffected.

We built a simulator to discover basic properties of DCMP network. We also implemented a prototype of our protocol and deployed it in PlanetLab using flat and super-peer network topologies. Our experiments indicate that DCMP achieves substantial reduction in 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 [4] or Gia [14]. Moreover, DCMP is orthogonal to the search algorithms. Our methods are also applicable to two-layer networks

based on super-peers (e.g., Kazaa [4]), since the super-peer layers employ Gnutella-style protocols.

1.4 Thesis Organization

The subsequent chapters of this thesis are organized as follows: Chapter 2 investigates and discusses the related work. Next, in Chapter 3 we describes DCMP in details. Chapter 4 and Chapter 5 include experimental results of our simulation and real world implementation, followed by the conclusions in Chapter 6.

Chapter 2

Related Work

Research in the P2P area was triggered by the apparent success of systems like Gnutella [2] [3] and Kazaa [4]. Gnutella is a pure P2P system which performs searching by Breadth-First-Search (BFS) 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. BFS, however, overloads the network with unnecessary messages; moreover, slow peers become bottlenecks. A lot of research has been done to overcome these problems. In general, existing approaches can be classified into three categories: *search-based, cache-based, and topology-based.*

2.1 Search-based Approach

There are two basic strategies which are used to search for objects in unstructured P2P networks: *blind search* and *informed search*:

- *Blind search* tries to propagate query to sufficient number of peers in order to satisfy the request. It does not use any hint of where the message should be

delivered to.

- *Informed search* makes use of the correlation between document locations and direction of connections. Thus query is forwarded to the neighbor/neoghbors who have high probability of returning queryhit according to different estimation techniques (e.g., degree, previous successful rate, indexing of documents).

2.1.1 Blind Search

Different techniques are proposed to compromise the QoS and traffic cost imposed to the system when conducting searches. They can be differentiated as:

Compromise Depth

We know the number of message used is exponentially increased as the TTL value increases. One direct method to reduce the large amount of traffic is to control how deep a query message can be forwarded.

Iterative Deepening [15] and *expending ring* [12] are two similar techniques which use multiple BFSs with successively larger depth if user-defined QoS is not satisfied (e.g., number of queryhit). Both algorithms achieve best results when a "small" flooding can get enough number of queryhit. In a different case, they produce even more workload and larger user-perceived delay than the standard flooding mechanism.

Compromise Breadth

We can also reduce the traffic cost by forwarding query messages to controlled number of neighboring peers.

Modified-BFS [16] is a variation of the flooding scheme, with peers randomly choosing only a ratio of their neighbors to forward the query to. This algorithm certainly reduces the average message production compared to the flooding method, but it still contacts a large number of peers. Assume the average degree of peers in the network is d and *Modified-BFS* forwards query to s ($s<d$) neighbors each round. Assuming the same TTL value t is used, this method only reduces the traffic cost from d^t to s^t.

Random walk [12], where a requesting node sends out k query messages to an equal number of randomly chosen neighbors. Each of these messages follows its own path, having intermediate nodes forward it to a randomly chosen neighbor at each step. These queries are also known as walkers. A walker terminates either with a success or a failure. The total number of messages can be controlled by fixed TTL value or periodically checking with source peer. Assuming TTL value t and w walkers are used, the number of messages needed is $t*w$ in the worst case, which does not depend on the underlying network. The most important advantage of this algorithm is the significant message reduction. As a tradeoff, the most serious disadvantage is its highly variable performance. Success rate and number of queryhit vary greatly depending on network topology and the random choices made.

Gkantsidis et al. [17] conduct further studies on random walk from another perspective. The authors make random walk use the same number of query messages as flooding approach does and count the number of answers for a query. The result shows that random walk can achieve improvement over flooding in the case of clustered overlay topologies and in the case of re-issuing the same request several times. Despite the delay problem, this study shows that both random walk and flooding

have similar network coverage if same amount of messages are used. In [18], same authors introduce a general search scheme of which flooding and random walk are special instances. This search scheme is actually the formalization of the work in the previous work [17]. In this scheme, a query is assigned with a budget value k by the peer who initializes the query and this budget is divided to k_1, k_2,...,k_d, where d is the degree of the source peer and sum of these k_i values is k. It assigns k_i values as the original TTL value for each query messages.

2.1.2 Informed Search

Informed search utilizes information about document locations. Thus searches can be forwarded to the peers with high probability of returning results.

Local Indices [15], where each node maintains an index over the data of all peers within r hops around itself, allowing each search to terminate after $d - r$ hops. This approach has high accuracy and hits since each contacted node indexes many peers. On the other hand, message production is comparable to the flooding scheme, although the processing time is much smaller because not every node processes the query. This scheme also requires a flood with $TTL = r$ whenever a node joins/leaves the network or updates its local repository, which introduces a large overhead in a dynamic environment.

Routing Indices [19], where documents are assumed to fall into a number of thematic categories. Each node knows an approximate number of documents from every category that can be retrieved through each outgoing link (i.e., not only from that neighbor but from all nodes accessible from it). Query is forwarded to the neighbor with the highest "goodness" value, which is calculated according to three different

ranking functions. This mechanism trades index overhead for increased accuracy. While a search is very bandwidth-efficient, RIs require flooding in order to be created and updated, which is similar to *Local Indices*. Thus it is also not suitable for highly dynamic networks. Moreover, the indices can be inaccurate due to thematic correlations, over-counts or under-counts in document partitioning and network cycles.

Intelligent-BFS [16] is a informed version of the *Modified-BFS* algorithm. Nodes store query-neighborID tuples for recently answered requests from (or through) their neighbors in order to rank them. First, a peer identifies all queries similar to the current one, according to a query similarity metric; it then chooses to forward to a set number of its neighbors that have returned the most results for these queries. If a hit occurs, the query takes the reverse path to the requester and updates local indices. This approach focuses more on object discovery than message reduction. At the cost of an increased message production compared to *Modified-BFS* [16], the algorithm increases the number of queryhits. It achieves very high accuracy, enables knowledge sharing and induces no overhead during node arrivals and departures. On the other hand, its message production is large and increases with time as knowledge is spread over the nodes. It also shows no easy adaptation to object deletion or peer departures because the algorithm does not utilize negative feedback from searches and forwarding is based on ranking. Finally, its accuracy depends on the assumption that nodes specialize in certain documents. Similar to *Intelligent-BFS*, *Directed BFS* is proposed in [15]. The authors suggest several heuristics to select the best neighbor to send the query, (e.g., select the neighbor that has returned the highest number of results for previous queries). This method is extended in Ref. [20, 21], where the network is reconfigured dynamically based on the query statistics.

Adaptive Probabilistic Search (APS) [22] has local index for each object a peer has requested per neighbor, which is associated with a probability value. Different from the above techniques, APS updates these probability values after *miss* or *hit* happens. This learning ability overcomes the disadvantage of explicitly updating of indexes in a dynamic environment, which induces zero overhead over the network at join/departure/update operations. The tradeoff is that the QoS is influenced by how fast the learning process can be carried on.

2.1.3 Discussion of Search-based Approaches

Blind search like Breadth First Search and Depth First Search has its own advantages. BFS aims at a small delay and large number of replies, while DFS (i.e., random walk is a kind of Depth First Search) aims at a small traffic cost. Compromising the pros and cons of BFS and DFS, a number of methods are proposed by controlling the TTL value used for the query message and the number of walkers. By maintaining indexing information of documents, informed search can answer queries more efficiently because they have mechanisms to forward queries to the peers which have high probability of providing answers. As a tradeoff, efficiently maintaining of the index information is an additional overhead.

2.2 Cache-based Approach

Caching techniques make use of the previous search results and thus reduces the traffic consumption for future searches. It can be generally divided into two groups: *peer-level caching* and *network-level caching*. Both approaches can achieve smaller user perceived delay and reduced traffic cost.

2.2.1 Peer-level Caching

In *Distributed Resource Location Protocol (DRLP)* [23], nodes with no information about the location of a file forward the query to each of their neighbors with a certain probability. If an object is found, the query takes the reverse path to the requester, storing the document location at those nodes. In subsequent requests, nodes with indexed location information directly contact the specific node. If that node does not currently obtain the document, it just initiates a new search as described before.

In *DiCAS* [24], the large amount of duplicated and unnecessary cache results among neighboring peers in the Uniform Indexing Caching (UIC)[1] mechanisms [25] [26] motivated the authors to propose a Distributed Caching and Adaptive Search (DiCAS) protocol. In DiCAS, each node randomly takes a initial value in a certain range [0,M-1] as a group ID when it participates. A peer caches the result of a query only if it satisfies the following condition:*Peer Group ID = hash (query) Mod M*. By doing these, the peers are divided into virtual multiple layers. Query flooding is restricted within the same layer to reduce the traffic cost.

2.2.2 Network-level Caching

In *Transparent Caching Scheme (TCS)* [27], caching is done at the gateway side. This scheme is proposed under the observation of existence of locality among the collective queries going through gateways. By caching at the gateway side, peers using the same gateway can share the queryhit stored at the gateways for a reduced delay and traffic. The drawback of this approach is similar to a centralized system: potential bottleneck and single point of failure at the gateway.

[1] *DRLP* [23] is also a UIC approach.

2.2.3 Discussion of Cache-based Approaches

Caching in P2P networks can save the overall traffic cost as well as reduce query response time. *Peer-level caching* retains the decentralized characteristic of P2P system and improves the efficiency of searches. The highly dynamic nature of P2P systems should be considered carefully in order to apply caching techniques effectively. Similar to traditional web caching techniques, *network-level caching* at gateway side in P2P system can further reduce the traffic cost, but also impose problems to gateways.

2.3 Topology-based Approach

Overlay topology is another important factor which can influence search efficiency by large volume. Hierarchical approach like Kazaa [4] and Limewire [28] adopts two-layer networks based on super-peers, which is widely deployed in current P2P networks. Non-hierarchical approach reorganizes the network topology according to different concerns without differentiating the layers in the network explicitly. We conduct a detailed study on these two directions in this section.

2.3.1 Hierarchical Approach

In hierarchical systems [4] [28], super-peer or ultra-peer works more like a server and leaf-peer works more like a client. Figure 2.1 represents a typical super-peer infrastructure. Leaf-peers only forward query to super-peers and super-peers flood query among its neighbors. Compared to original Gnutella networks, the traffic among leaf-peers is saved. Although the search cost is reduced, there are still redundant traffic when a leaf-peer submits query to multiple super-peers and super-peers flood query among themselves, which we can observed from Figure 2.1. This is due to the fact that in super-peer layer, super-peers keep many connections to other super-peers, which

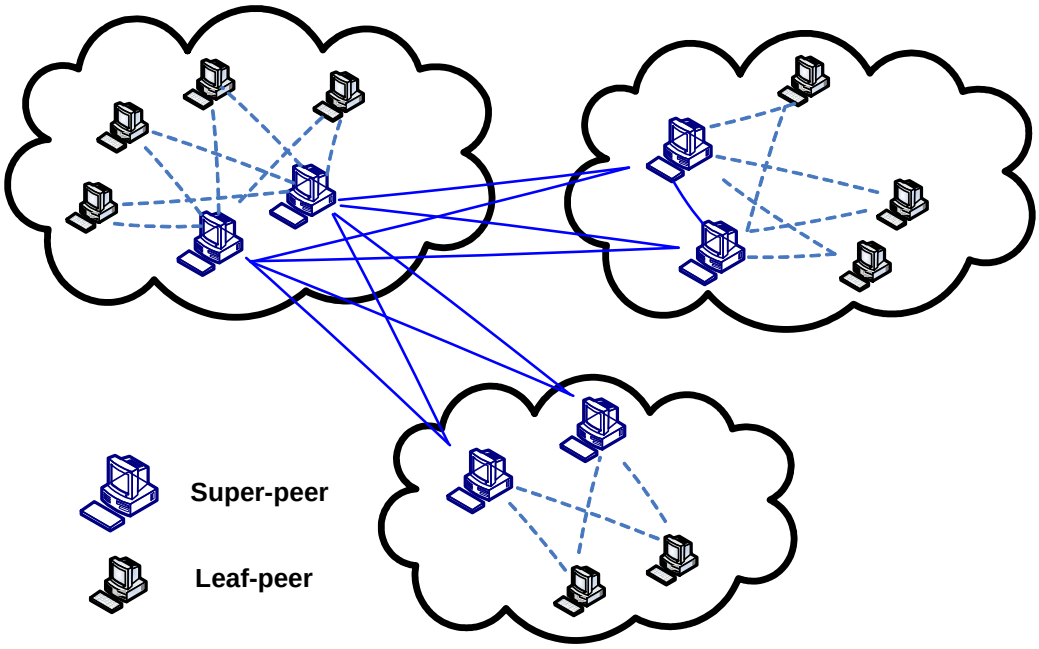


Figure 2.1 Hierarchical topology of unstructured P2P networks

introduces a lot of cycles.

In *SUPS* [29], authors describe a protocol which is inspired by the theory of random graphs. This method constructs a balanced and low-diameter super-peer topology and is designed to be robust to super-peer failures and inconsistent information. In *SUPS*, a super-peer i first estimates the network size N_i. Then, according to the random graph theory, it computes the minimum number of connections σ_i to other super-peers which should be maintained, $\sigma_i = \lceil \ln N_i \rceil + 1$. By keeping track of the other super-peers which has the lowest degree among all super-peers, a super-node tries to connect to low degree super-peers it knows whenever the current degree is smaller than the estimated σ_i. The advantages for this method is the almost equal sharing of workload among super-peers and the low cost since connection optimization happens infrequently. On the other hand, an accurate estimation of the σ_i for all

super-peers is not easy, which has negative impact on the optimality of the network structure. Besides, balanced structure may not be the best choice since there are still considerable heterogeneity among the super-peers.

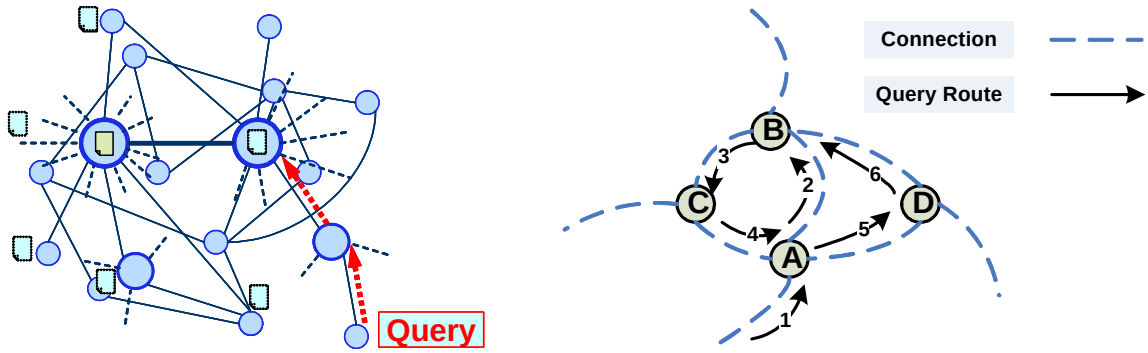
In *DLM* [30], based on a workload model, a super-peer estimates an optimal ratio between its neighboring leaf-peers and super-peers. Super-peers are promoted or demoted to keep the actual ratio of super-peers and leaf-peers around the optimal value.

Interest-based Locality [31] can be looked as a special case of building a layer above a Gnutella-style network. A peer first contacts the peers who have provided queryhits previously. In the case that these peers share similar interests, the documents can be found efficiently. If this process does not return enough results, a normal BFS is conducted.

Hierarchical approach strikes a balance between the efficiency of centralized search and the load-balancing and robustness to attacks provided by distributed search. In addition, super-peer architectures leverage on the pronounced heterogeneity of resources across peers, by assigning each peer a task that is in agreement with the amount of resources it holds.

2.3.2 Non-hierarchical Approach

Unlike hierarchical systems, non-hierarchical systems also make use of the large heterogeneity of different nodes. The main difference is that all peers are equal in the search phase in a non-hierarchical approach. Now we discuss techniques proposed in [14] [28] [32] [33], which are more relevant to our approach.



(a) Example of a Gia network (b) Example of duplicate messages in Gia

Figure 2.2 Gia Examples

GIA [14] improves the scalability of Gnutella by using a combination of topology adaptation, flow control and one-hop replication. The basic network structure of Gia is shown in Figure 2.2.a. 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 searching by biased random walks. Specifically, each peer forwards the query to the neighbor with the highest capacity. However, the possibility of duplicates still exists. Consider for instance the network of Fig. 2.2.b, 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.

Limewire [28], a Kazaa clone, maintains a table where it stores the IDs of dupli-

cate messages and the directions (i.e., neighbor peers) from where they arrive. Once a message is identified as duplicate, it is discarded. Further forwarding of messages avoids the directions from where duplicates have arrived. Keeping information for every instance of each duplicate message may require a lot of memory, especially in high-degree peers. Limewire also implements a simplified version which disables those connections from where "a lot" of duplicates are arriving. In practice, however, 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.

In order to reduce the topology mismatch between the P2P overlay network and the physical underlying network, both ACE [32] and LTM [33] use network delay as a metric to reduce the traffic crossing different Autonomous Systems, thus improve the efficiency of P2P overlay topology. Now we describe the two mechanisms in details.

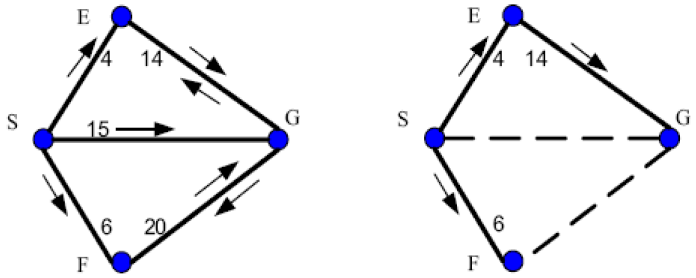


Figure 2.3 Building local multicast tree in ACE

In *ACE* [32], a node probes the delay to its direct neighboring nodes periodically and maintains a cost table. Two neighboring nodes exchange their cost tables so that a node can obtain the cost between any pair of its logical neighbors. With this information, a minimum spanning tree among each node and its immediate logical neighbors is built using a standard algorithm like PRIM which has a computation complexity of $O(m^2)$, where m is the number of logical neighbors of the source node.

In Figure 2.3, node S builds a local multicast tree using PRIM algorithm. Peers continue to probe and measure the delay to neighboring peers up to h hops away, thus, new multicast tree can be built dynamically in this process.

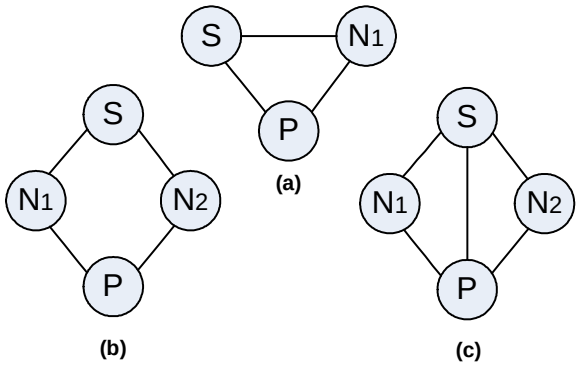


Figure 2.4 Eliminating cycles in LTM

In *LTM* [33] network delay is also used as the cost metric to improve the network structure. In this framework, each peer floods a specially designed message called $TTL2-detector$ periodically, which is flooded within two hop distance around the source peer. The purpose of this message is to collect network delay information between the peers in a cycle, thus eliminate the cycles by cutting the connection with the largest delay. As an example, in Figure 2.4.a, when peer P receives a detector message with TTL=1, it can calculate the cost of link SP from Source Timestamp and the time P receives the message from S. When P receives a detector message with TTL=0, it can calculate the cost of link SN_1 and N_1P from the Timestamps contained in the message. Here if we assume link SP has the maximum delay, then this virtual connection SP will be disabled. Two more examples are shown in Figure 2.4.b and c respectively. We do not describe them in details since they are similar to the case in Fig. 2.4.a. Moreover, this technique is similar to ACE [32] in the sense

that peers also try to probe the delay between a peer and its neighbors' neighbors to form new structure dynamically in the purpose of maintaining connections with small delay.

2.3.3 Discussion of Topology-based Approaches

As we have explained in the introduction, the large amount of duplicate traffic which occurs at high-degree peers makes them the potential bottleneck of the network. Ref. [32] and [33] closely relate to the proposed protocol in this thesis. Thus, we discuss them in the following section.

Limitation of Closely Related Approaches

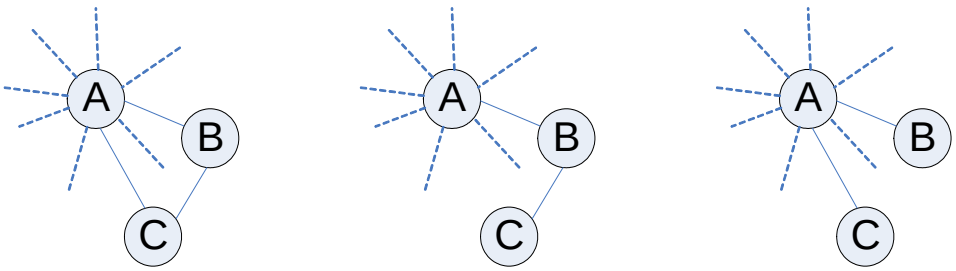
A. Decision on cutting position

Both of *ACE* and *LTM* use network delay when eliminating redundant connections. In particular, *LTM* cuts a cycle at the position which has the largest measured delay. We discuss the problems in the following paragraphs:

- clock synchronization: all peers need to be synchronized accurately because the algorithm measures the delay by inserting timestamps in $TTL2-detector$ message, which is nontrivial in a large distributed network [34].

- inaccuracy of estimated delay: since all peers flood $TTL2-detector$ messages, measured delay values of the same link will hardly be the same. This is due to the jitter both in network and application layers. This inaccuracy can result different decisions on the cut positions. Thus, network can potentially split into fragments. Take the graph in Figure 2.4.b for example, peer N_2 is isolated if the following happens: N_1 and S are the sources of the detector messages,

measured delay of SN_2 by node N_2 is the largest in cycle N_1PN_2S and measure delay of PN_2 by node P is the largest in cycle SN_1PN_2. Neither of the papers has considered about this problem.



(a) Original network topology

(b) Network topology after using delay to decide cut position

(c) Network topology after using node degree to decide cut position

Figure 2.5 Example of using delay and degree to decide cut position

- the effectiveness of cutting a cycle by eliminating the connection which has the largest delay in the cycle is questionable. Now we consider the examples in Figure 2.5. Figure 2.5.b is the structure of using delay as a cost metric to cut the cycle under the assumption that connection AC has the largest delay D_{AC} in cycle ACB. We can simply apply triangular inequality here that $D_{CB}+D_{BA}$ is larger than D_{AC}, which holds because the delay can be estimated by the physical distance. In Figure 2.5.c, connection BC is cut because peer A has the highest degree. Having the case that peer C initializes a query, we consider the structure in Figure 2.5.c is better than Figure 2.5.b because peer C can relay on peer A to contact many peers in the network which increases the probability of returning results. If we follow the network structure in Figure 2.5.b, a constant delay to all the potential results is added if node B does not have the

requested data, which is true in most of the cases. This example suggests that in a power-law network like Gnutella, forwarding queries to high-degree peers can be more efficient.

B. Overhead

In LTM, every peer floods detector messages periodically, which creates a considerable overhead to all the peers during their life time. This is due to fact that it adopts an "eager" approach of disabling unnecessary connections.

Cheaper ways of handling this problem can follow a "lazy" approach. One possibility is to only react to the problematic cycles, which is the basic idea of our proposed approach. By detecting the duplicate queries caused by small cycles in the network, we can eliminate them according to some defined metrics. Thus, the overhead will only be proportional to the number of small cycles in the network, which reduces the overhead significantly. Moreover, there will be additional benefits since we can gather and keep useful information in the cycle elimination process. For example, the information is used for distributed network maintenance in our protocol.

Chapter 3

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.

3.1 Notations

- 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 virtual 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: $(GUID(msg), B \rightarrow A)$

3.2 Simplistic Cycle Elimination (SCE)

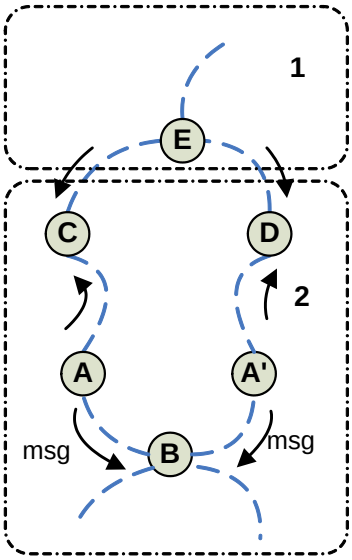


Figure 3.1 Simplistic Cycle Elimination

To motivate our approach, here we describe a straight-forward method for eliminating cycles and explain its drawbacks. Consider Fig. 3.1 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.

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.

3.3 DCMP: Distributed Cycle Minimization Protocol

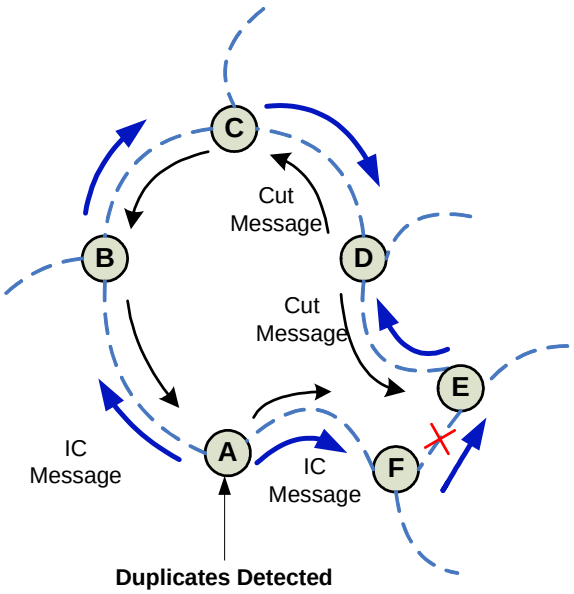


Figure 3.2 Distributed Cycle Minimization Protocol

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. 3.2 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.

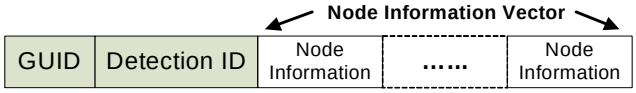


Figure 3.3 Structure of the IC message

Fig. 3.3 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

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

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

Figure 3.4 Algorithm for handling the IC message

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.

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. 3.4.

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 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 Chapter 4 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

[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

Figure 3.5 Algorithm for handling the CM message

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. 3.5.

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 and (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.

3.4 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:

$$\begin{cases} |p| = |p'| = \lceil |S_N|/2 \rceil \text{, if } |S_N| \text{ is odd} \\ |p| = |p'| - 1 = |S_N|/2, \text{ if } |S_N| \text{ is even} \end{cases} \tag{3.1}$$

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

$$\mathcal{P}(N) = \sum_{i=1}^{n} w_i \cdot f_i^N \tag{3.2}$$

where w_i is the predefined weight of ith factor that contributes to the equation, f_i^N is the value of ith factor at node N, n is the number of factors. Currently we have adopted three factors in our protocol. They are the bandwidth, CPU processing

power and degree of the node (i.e., maximum number of simultaneous connections). However, it is extensible for more factors.

It is obvious why the bandwidth and CPU power characterize how powerful a peer is. Larger the bandwidth is, faster the messages will be dispatched to neighboring nodes. CPU power is also an important factor because table lookup is an essential operation, where searching for matching objects and searching for message ID perform frequently at the peer. 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. [35] 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

Precondition: Node N receives two IC messages icm and icm' which satisfy the conditions:

$$GUID(icm) = GUID(icm') \textbf{ and } DetectionID(icm) = DetectionID(icm')$$

1. Calculate the *power* P_i of each peer in NIVs using **Definition** 3.2
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** 3.1
5. Generate *Cut* message(s) accordingly

Figure 3.6 Algorithm for selecting the GatePeer in a cycle

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. 3.6.

3.5 Disseminating GatePeer Information

GatePeers assist to recover from node failures and are used as entrance points in a dynamic network (refer to Section 3.7.1); 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*. 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 5.1.3 for details).

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

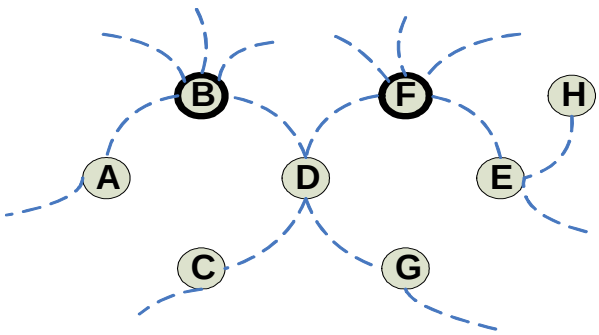


Figure 3.7 Example of transitive peer D

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. 3.7 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. 3.7 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, 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.

3.6 Concurrent Cycle Elimination

In Section 3.2 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 3.8 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 that in the cycle $ABGFEA$ is F. The connection opposite to C is $\overline{AE}$, whereas the

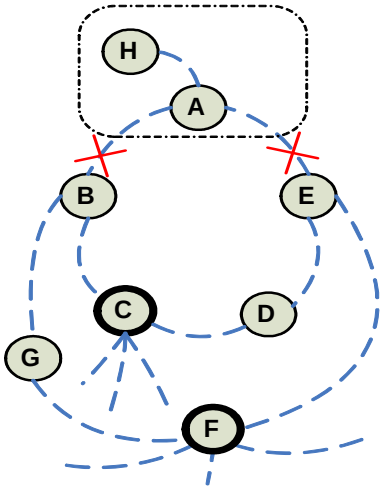


Figure 3.8 Network splits because of concurrent cycle elimination

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

[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.

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

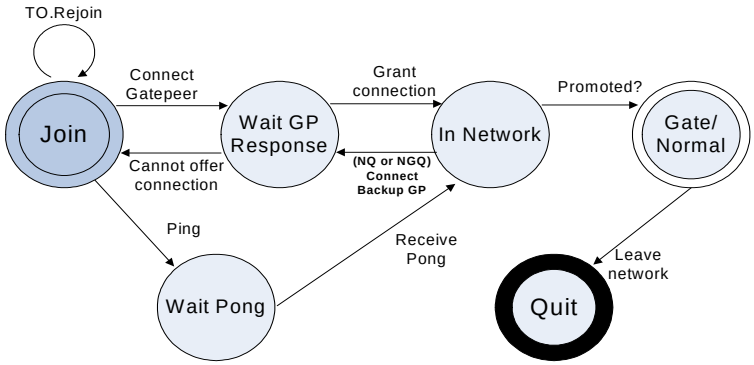


Figure 3.9 State diagram to handle node quit/join. *GP* is GatePeer. *NGQ* means Neighbor GatePeer Quit. *NQ* means departing node is in the primary direction of a peer. *TO*: timeout period.

practice this happens rarely. Moreover, even if a new cycle is generated, it will be identified and eliminated soon after.

3.7 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. 3.9.

3.7.1 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, it uses 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.

3.7.2 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 (i.e., 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 two to five backup GatePeers were usually selected depending on the degree of G and 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 failure), then the affected peers simply re-join the network using the peer arrival procedure described above.

3.7.3 Departure of Normal Peer

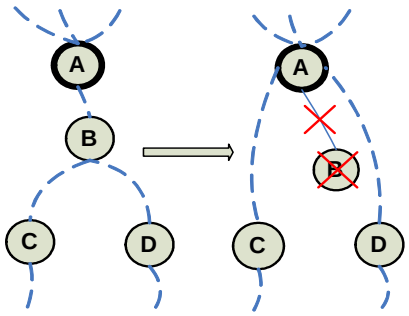


Figure 3.10 Failure of normal peer B

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. 3.10. 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.

Chapter 4

Evaluation by Simulation

We developed an event-driven simulator to test the various parameters of our protocol in a controlled environment. Our simulator is accurate down to the message transmission layer and takes into account the network delays; however, we do not simulate the TCP/IP layer. The simulator is written in C++ and was executed on a Linux machine with six 3.0GHz CPUs and 18GB of RAM (3GB for each processor). We used power-law topology with average degree 3.4, whereas the network size varied from 500 to 3000 peers (results based on 3000 by default in this chapter). 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.

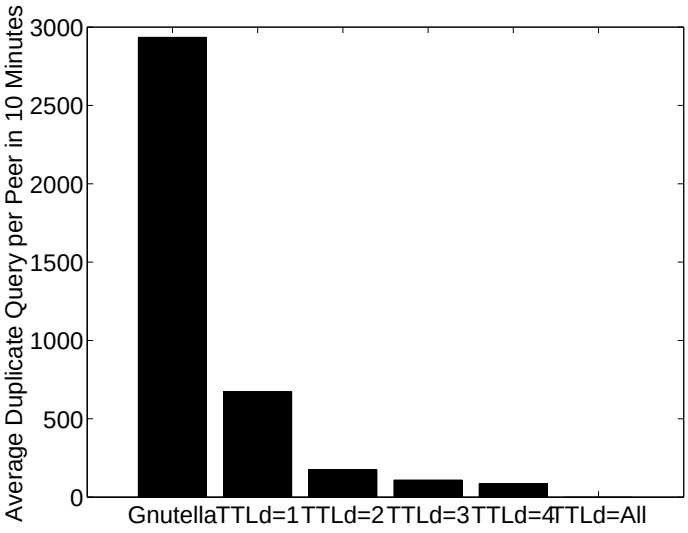


Figure 4.1 Duplicate messages vs. TTL_d values

4.1 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. 4.1, 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 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 con-

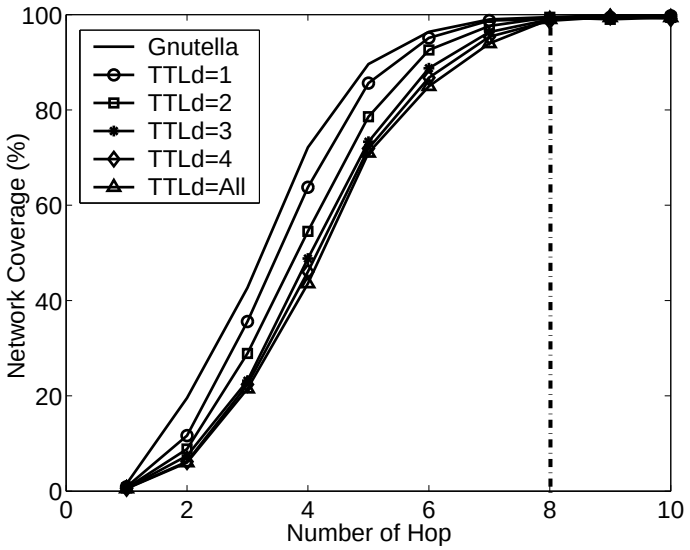


Figure 4.2 Connectivity vs. TTL_d values

nectivity. 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. 4.2 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 the two diagrams from Fig. 4.1 and Fig. 4.2, 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. 4.3, 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

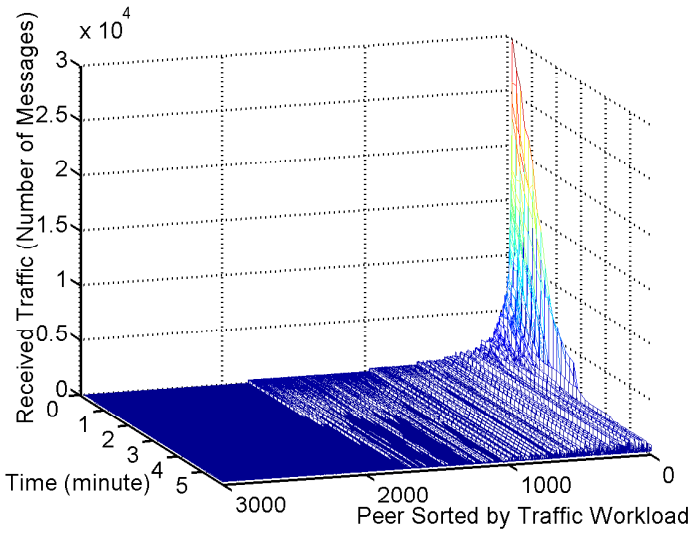
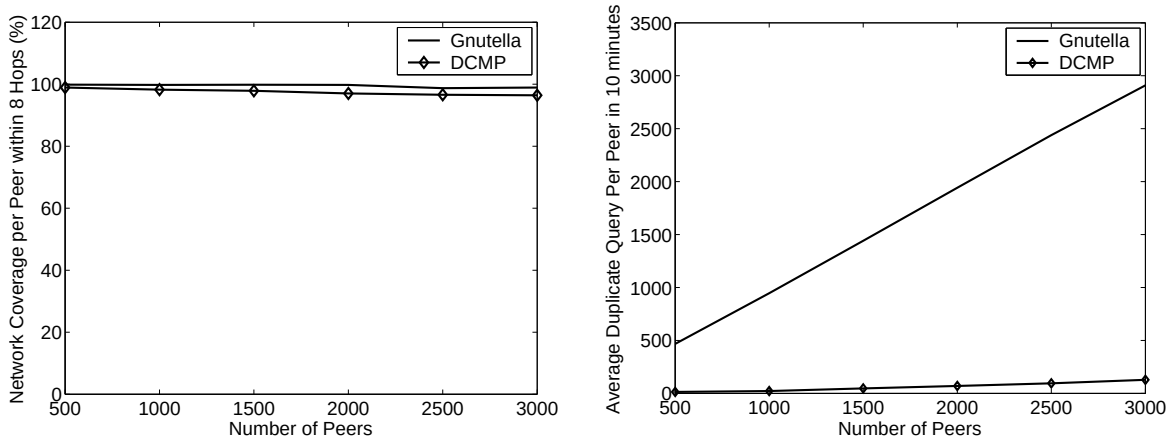


Figure 4.3 Received traffic

remains almost unaffected.

4.2 Influence of Network Size

In this experiment, we vary the number of peers in the network. Fig. 4.4.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. In Fig. 4.4.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) are small.



(a) Connectivity (b) Duplicate query

Figure 4.4 Scalability for Gnutella and DCMP (static networks)

4.3 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. 4.5, 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.

4.4 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.,

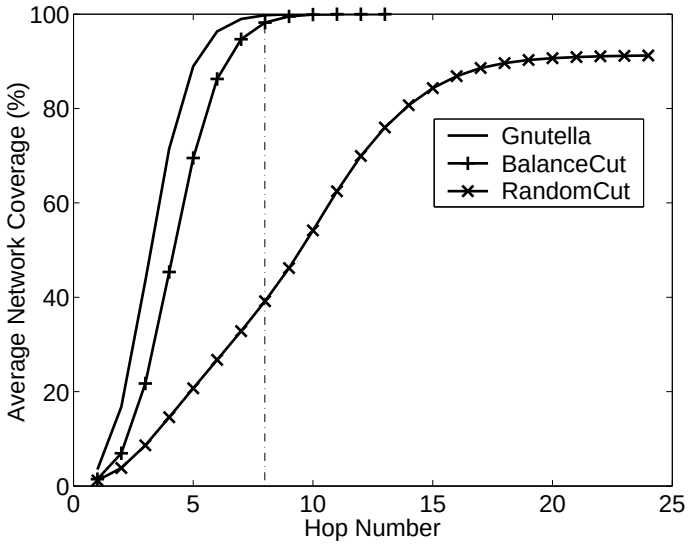


Figure 4.5 Balance vs. Random cut

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. 4.6 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 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

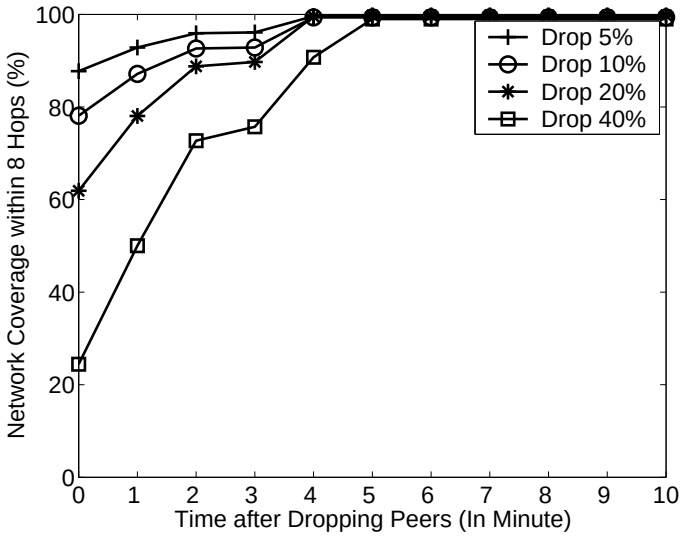


Figure 4.6 Random failure analysis

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. 4.6). 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; however, it is outside the scope of this paper.

4.5 Comparison with other Approaches

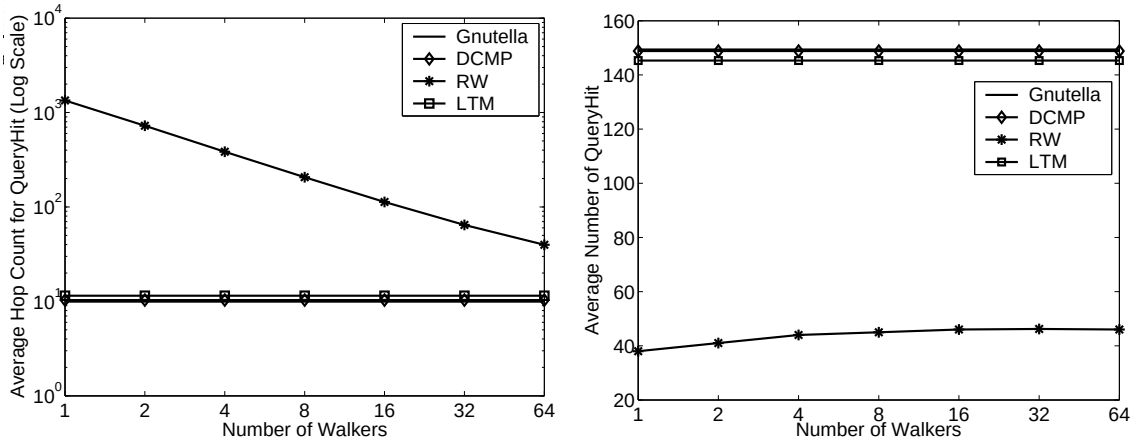
Ref. [12] uses *Random Walk* (RW) for searching in unstructured P2P networks. The algorithm initiates k random walkers. In order to reduce the system load, the walkers contact the initiator node periodically, to check whether the search should stop. Despite the overhead of contacting the query initiator, 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. [17] 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 [33] 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.

4.5.1 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 Ref. [17]). For LTM, we followed the optimal frequency of broadcasting detection messages suggested by Ref. [33]. DCMP, RW and LTM transmit fewer messages for each query compared to Gnutella, since many duplicates are avoided. Delay is measured as the number of hops from the moment a query is sent until each answer arrives to the querying peer. The average delay is shown in Fig. 4.7.a. We observe that the delay of RW is about four times larger than the other techniques, even when many walkers are used. Increasing the number of walkers reduces the delay, which is expected since



(a) Path lengths of QueryHit (b) Number of QueryHit

Figure 4.7 Comparison of Random Walks, Gnutella, DCMP and LTM (dynamic network)

RW tends to flood the neighbors. In our experiments, there are around 150 replicas of each object in the network. Fig. 4.7.b shows that Gnutella, DCMP and LTM can find almost all of them, but RW discovers less than 33% of the copies. Also, the performance of DCMP is slightly better than LTM.

Fig. 4.8 shows the number of duplicates for each technique. For all the cases, RW generates more duplicate messages than DCMP, if both methods transmit the same number of messages. Compared to LTM, our protocol is slightly better. However, our protocol generates much less overhead than LTM during peer joins, as we will explain in the next experiment. Besides, we present the data for four techniques in table 4.1.

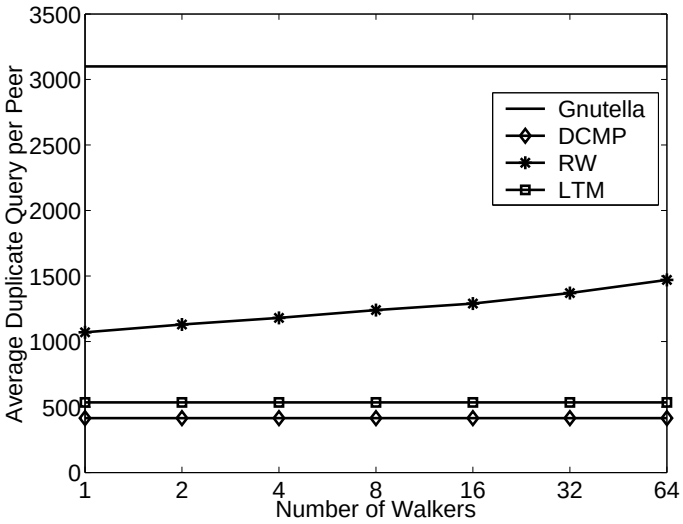


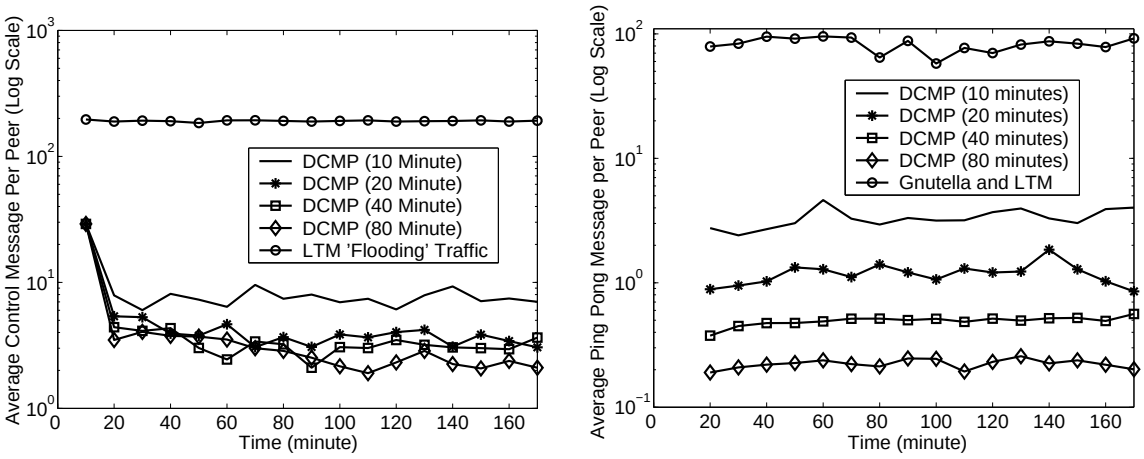
Figure 4.8 Comparison of Random Walks, Gnutella, DCMP and LTM (dynamic network)

	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

Table 4.1 Comparison of Random Walks, DCMP, Gnutella and LTM

4.5.2 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 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) Overhead analysis

(b) Ping and Pong Messages

Figure 4.9 Overhead analysis and effect of session time on Control and Ping/Pong Messages

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 at least once.

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. 4.9.a, LTM's overhead is one to two orders of magnitude larger than that of DCMP. 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. 4.9.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. [36]), the joining overhead for DCMP is reasonably small.

Chapter 5

Prototype Evaluation on PlanetLab

We implemented the DCMP protocol in a prototype and deployed it in PlanetLab [13]; 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.

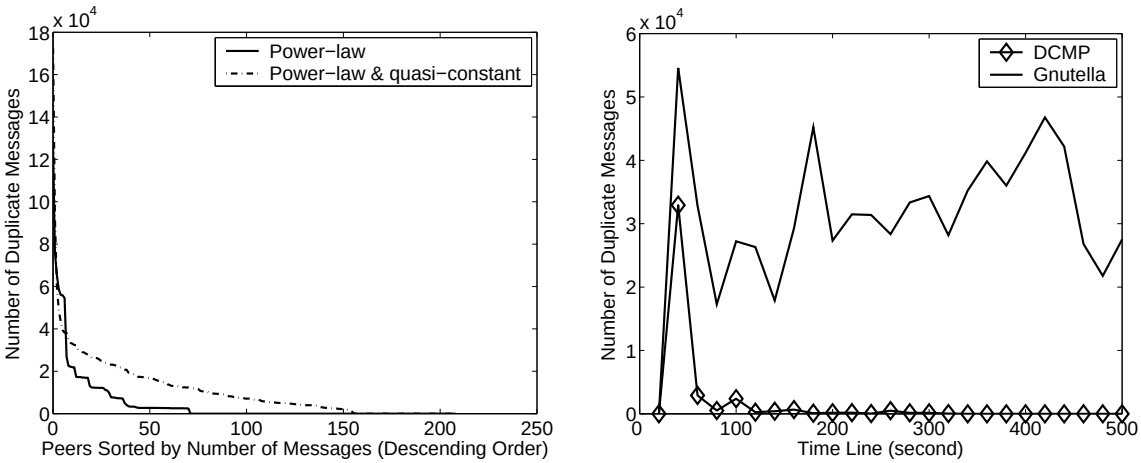
We generated two network topologies which appear in real-life P2P networks [11], in order to test DCMP: (i) Power-law topology, with average degree 3.4. We used the PLOD [37] method to construct the network. This topology reflects the original Gnutella network (i.e., protocol v0.4). (ii) Two-layer network, 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 [38] 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) Duplicate distribution in Gnutella and Super-peer architectures

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

Figure 5.1 Duplicate distribution and time-line of duplicates

5.1 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.

5.1.1 Duplicates Analysis

In Fig. 5.1.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[6]. Both topologies are prone to a large number of duplicates; however the two-layer network suffers most.

[6]Recall from Fig. 1.2.b that duplicates account for more than 50% of the total messages.

In two-layer architectures, about 10% of the nodes are super-peers [38] having a large number of neighbors which are also super-peers. Therefore, cycles are formed with high probability and they introduce numerous duplicate messages.

Fig. 5.1.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 exact number of cycles; in practice, it takes no more than a few minutes. Similar results were obtained for super-peer architectures.

5.1.2 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 which receives a message, uses the message's TTL to calculate the distance (in hops) to the origin. Obviously, if a duplicate

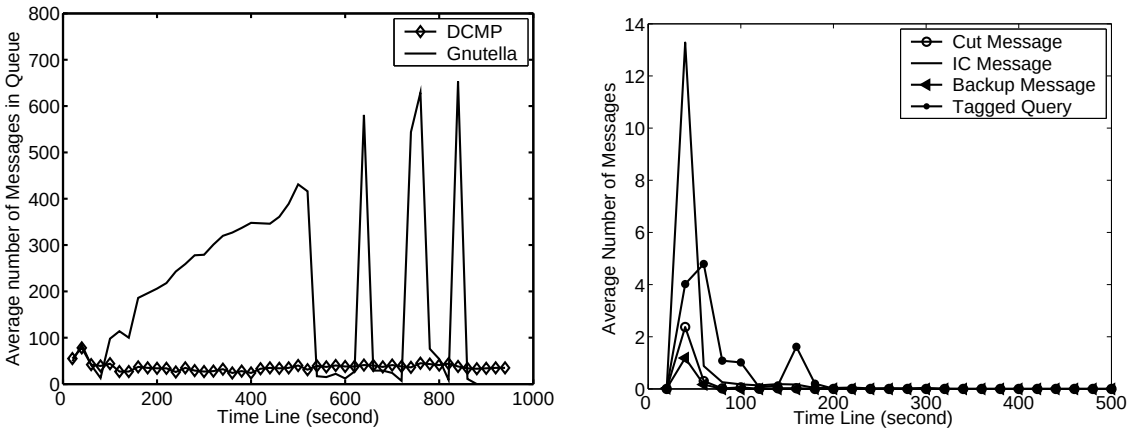
	Average Hops
DCMP	2.8
Gnutella	3.9

Table 5.1 Average number of hops in static networks

arrives, it is ignored and the distance is not computed.

The average number of hops is shown in Table 5.1. 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.

To verify this behavior, in Fig. 5.2.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.



(a) Average message queue size vs. elapsed time

(b) Overhead of control messages (DCMP)

Figure 5.2 Average queue size and overhead analysis

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.

5.1.3 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 the set of 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. 5.2.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 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.

5.2 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 Poisson distribution with mean equal to 90 min [39]. 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 [35] states that the lifespan of super-peer architectures follows the Pareto distribution. This implies that our GatePeers should have a lifespan of several

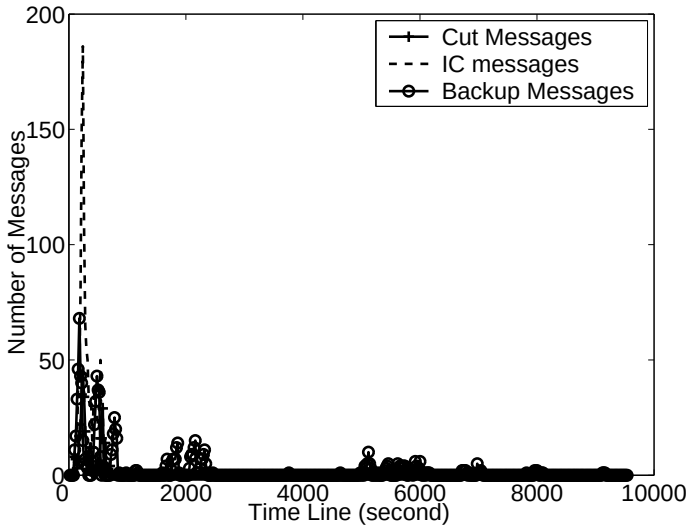


Figure 5.3 Overhead due to control messages in dynamic DCMP systems

days [36]. 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 Poisson distribution, which causes GatePeers to fail faster and allows us to investigate the behavior of DCMP under such failures. We stretch that the Poisson distribution represents the worst case for our protocol. In practice, we expect less GatePeer failures, hence better overall performance.

5.2.1 Overhead Analysis

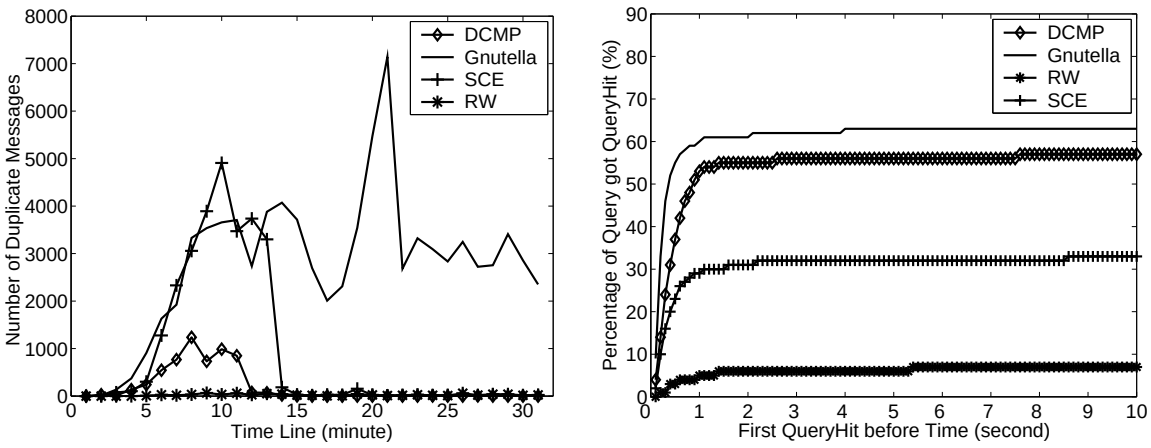
In Fig. 5.3 we present the overhead due to control messages in the dynamic environment. We do not show the tagged messages since they follow largely the IC messages. Compared to the static case (i.e., Fig. 5.2.b), more control messages are required since new cycles are introduced. For example, except 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 rejoining 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. From the graph, we can clearly 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.

5.2.2 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. 5.4.a 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. 5.4.b. 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



(a) Duplicate Queries (b) Delay of first answer

Figure 5.4 Analysis of DCMP Control Message, Duplicate Queries and Real Time delay for Gnutella, DCMP, SCE, RW

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 many factors (e.g., 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 5.2 we show the average number of hits per query. Clearly DCMP provides the best results.

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

Table 5.2 Average number of queryhit in dynamic networks

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. 5.5 and Fig. 5.6. As expected, DCMP generates much fewer duplicates than Gnutella. Moreover, since the network traffic has increased, the overhead of duplicates becomes more obvious. As a result, DCMP easily outperforms Gnutella in terms of delay.

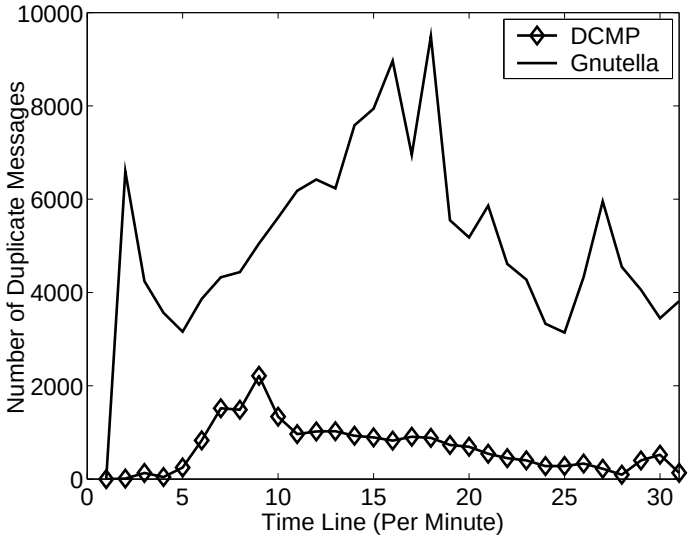


Figure 5.5 Duplicates in Dynamic heavy loaded networks

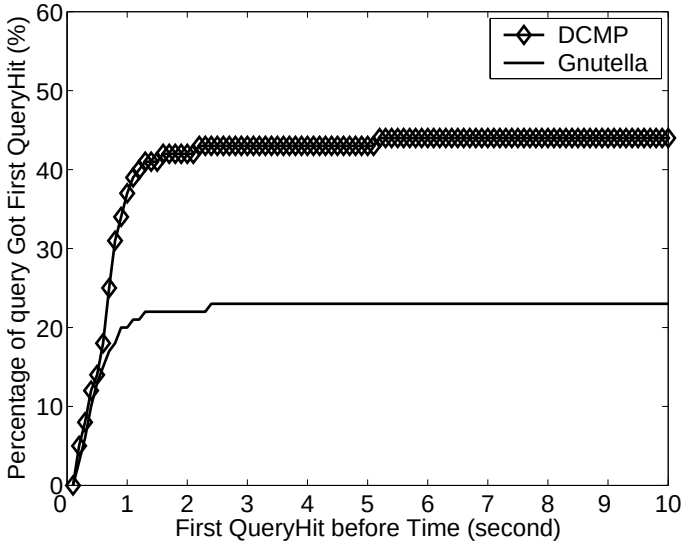


Figure 5.6 Delay in Dynamic heavy loaded networks

Chapter 6

Conclusions

6.1 Contributions

In this thesis we proposed DCMP, a dynamic, fully decentralized protocol which reduces significantly the duplicate messages by eliminating unnecessary cycles in unstructured P2P networks. DCMP preserves the low diameter of Gnutella-like networks while eliminating most of the duplicate messages. 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 proposed protocol is 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 built a simulator and investigated the basic characteristics of DCMP networks. We also deployed a prototype on PlanetLab and verified that our techniques are

applicable to realistic environments. Here the contributions of this thesis are listed as follows:

- We explore the existing techniques and discuss them by classifying them into three categorizes: search-based, cache-based and topology-based.
- DCMP reduces duplicate messages by as much as 90%.
- DCMP requires few control messages, therefore the overhead is minimal.
- 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.
- 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 splits. As a result, the connectivity and average path length remain relatively unaffected.

6.2 Future Works

In the future the following directions are worth looking into:

1. We would like to consider peer session time in selecting the GatePeers since longer lifetime is beneficial for a more stable network structure in an environment which majority of the peers are volatile.
2. We would like to make DCMP more robust to well orchestrated attacks. As existing investigation [36] has show that Gnutella-like networks are sensitive to attacks. To deal with this, one possibility is to conduct distributed monitoring

among the peers. There are some audit algorithms proposed in Ref. [40] to improve the robustness of the network. We can investigate the possibility of adapting them into our protocol.

3. We also plan to investigate the possibility of employing DCMP outside the P2P area; for instance in sensor networks. Since traffic reduction is a promising direction for extending the lifetime of small devices powered by batteries.

Bibliography

[1] Napster, http://www.napster.com/.

[2] Gnutella, http://www.gnutella.com/.

[3] gdf, http://groups.yahoo.com/group/the_gdf/.

[4] Kazza, http://www.kazaa.com/.

[5] 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 **11,** 17–32 (2003).

[6] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A Scalable Content-Addressable Network," In *Proc. of ACM SIGCOMM*, pp. 161–172 (2001).

[7] H. V. Jagadish, B. C. Ooi, and Q. H. Vu, "BATON: A Balanced Tree Structure for Peer-to-Peer Networks," In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, (2005).

[8] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-Grid: a Self-organizing Structured P2P System," SIGMOD Rec. **32,** 29–33 (2003).

[9] D. MALKHI, M. NAOR, and D. RATAJCZAK, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," In *Proc. of ACM symposium on Principles Of Distributed Computing*, (ACM Press, 2002).

[10] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh, "Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience," In *Proc. of ACM SIGCOMM*, pp. 395–406 (2003).

[11] M. Ripeanu, A. Iamnitchi, and I. T. Foster, "Mapping the Gnutella Network," IEEE Internet Computing **6,** 50–57 (2002).

[12] 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)*, pp. 84–95 (2002).

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

[14] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-Like P2P Systems Scalable," In *Proc. of ACM SIGCOMM*, pp. 407–418 (2003).

[15] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," In *Proc. of IEEE ICDCS*, pp. 5–14 (2002).

[16] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," In *Proc. of Int. Conf. on Information and Knowledge Management*, pp. 300–307 (ACM Press, 2002).

[17] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," In *Proc. of IEEE INFOCOM*, **1,** 120–130 (2004).

[18] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid Search Schemes for Ustructured Peer-to-Peer Networks," In *Proc. of IEEE INFOCOM*, **3,** 1526–1537 (2005).

[19] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems," In *Proc. of IEEE ICDCS*, p. 23 (2002).

[20] S. Bakiras, P. Kalnis, T. Loukopoulos, and W. S. Ng, "A General Framework for Searching in Distributed Data Repositories," In *Proc. of IEEE IPDPS*, pp. 34–41 (2003).

[21] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K.-L. Tan, "An Adaptive Peer-to-Peer Network for Distributed Caching of OLAP Results," In *Proc. of ACM SIGMOD*, pp. 25–36 (2002).

[22] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," In *P2P '03: Proc. of Int. Conf. on Peer-to-Peer Computing*, p. 102 (IEEE Computer Society, 2003).

[23] D. A. Menasce and L. Kanchanapalli, "Probabilistic Scalable P2P Resource Location Services," SIGMETRICS Perform. Eval. Rev. 30 (2002).

[24] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed Caching and Adaptive Search in Multilayer P2P Networks," In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, (2004).

[25] K. Sripanidkulchai, "The Popularity of Gnutella Queries and its Implications on Scalability,", http://www.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html, 2001.

[26] E. P. Markatos, "Tracing A Large-Scale Peer to Peer System: An Hour in the Life of Gnutella," In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, (2002).

[27] S. Patro and Y. C. Hu, "Transparent Query Caching in Peer-to-Peer Overlay Networks," In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, (2003).

[28] Limewire, http://www.limewire.com/.

[29] Y. J. Pyun and D. Reeves, "Constructing A Balanced, (log(N)/1oglog(N))-Diameter Super-Peer Topology for Scalable P2P Systems," In *Proceedings. Proceedings. Fourth International Conference on Peer-to-Peer Computing*, pp. 210 – 218 (2004).

[30] Z. Zhuang, L. Xiao, and Y. Liu, "Dynamic Layer Management in Superpeer Architectures," IEEE Trans. Parallel Distrib. Syst. **16,** 1078–1091 (2005).

[31] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," In *Proc. of IEEE INFOCOM*, pp. 2166–2176 (2003).

[32] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, "A Distributed Approach to Solving Overlay Mismatching Problem," In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pp. 132–139 (2004).

[33] X. Liu, Y. Liu, L. Xiao, L. M. Ni, and X. Zhang, "Location Awareness in Unstructured Peer-to-Peer Systems," IEEE Trans. Parallel Distrib. Syst. **16,** 163–174 (2005).

[34] S. Muir, "The Seven Deadly Sins of Distributed Systems," In *WORLDS 04: First Workshop on Real, Large Distributed Systems*, (2004).

[35] F. Bustamante and Y. Qiao, "Friendships that Last: Peer Lifespan and Its Role in P2P Protocols," In *International Workshop on Web Content Caching and Distribution*, pp. 233–246 (2004).

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

[37] C. R. Palmer and J. G. Steffan, "Generating Network Topologies that Obey Power Laws," In *Proc. of GLOBECOM*, pp. 434–438 (2000).

[38] Statistics, http://www.limewire.com/english/content/uastats.shtml.

[39] N. Christin, A. S. Weigend, and J. Chuang, "Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks," In *Proc. of ACM Conf. on Electronic Commerce (EC)*, pp. 68–77 (2005).

[40] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses," In *Proc. of IEEE INFOCOM*, **1,** 120–130 (2006).