EFFICIENT MUTUAL EXCLUSION IN PEER-TO-PEER SYSTEMS

BY

MOOSA MUHAMMAD

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

Traditional peer-to-peer (p2p) applications such as Kazaa and Gnutella have been primarily used for sharing read-only files (such as mpegs, jpegs, and mp3s) over the Internet. Such applications did not demand mechanisms to protect concurrent access and update to shared data, and therefore mutual exclusion has not yet been thoroughly studied in the p2p community. Due to the recent surge in the area of Grid computing, there is an urgency to find efficient ways of sharing resources. In the future, application developers should be able to write Grid and p2p applications without worrying about handling the low-level details of protecting consistent and concurrent access to shared resources.

This thesis proposes two novel protocols for achieving mutual exclusion efficiently in dynamic p2p systems. The protocols are layered atop a distributed hash table (DHT), making them scalable and fault-tolerant. The burden of controlling access to the critical section is also evenly distributed among all the nodes in the network, making the protocols more distributed and easily adaptable to growing networks. Since the protocols are designed independent of any specific DHT implementation, they can be incorporated with any generic p2p DHT, depending on the application requirements.

We present experiments comparing our implementations with existing mutual exclusion algorithms. Since the size of the sample network being considered in these experiments is in the 1000's, it presents a realistic study and the results derived from it can be directly applied to “real world” p2p networks. The significant reduction in overall message overhead and better load-balancing mechanisms makes the proposed protocols very attractive in being used for current and future p2p and Grid applications.

To my loving parents and brother

Acknowledgements

I am really grateful to my advisor Professor Indranil Gupta for giving me an opportunity to work under him. He guided me throughout my Master's career with his wisdom and helpful attitude. I could not have wished for a better advisor and mentor. Adeep Cheema for his continued help in the design phase of this project. I would also like to thank Jalal Al-Muhtadi for his guidance and advice throughout my college career. Finally, my family has been a source of constant support and has always been there when I needed them the most.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The problem of mutual exclusion can be described as a collection of asynchronous processes, each alternately executing a critical and a non-critical section that must be synchronized so that no two processes ever execute their critical sections concurrently. It was first described and solved by Dijkstra in [5]. Distributed mutual exclusion introduces some new requirements which can be summarized as follows:

- Safety – At most one process may execute in critical section at any time
- Liveness – Every request for a critical section is eventually granted
- Ordering (desirable) – Requests are granted in First-In-First-Out (FIFO) order

Even though mutual exclusion is a classical, well studied problem in distributed systems and several viable solutions have been proposed, it yet remains to be completely explored in the peer-to-peer (p2p) domain. Directly adapting the mutual exclusion algorithms from distributed computing literature is not possible due to the differences in the underlying system models, one of which is the absence of any centralized index server to keep track of membership and to ensure consistency. Classical decentralized algorithms use several rounds of all-to-all communication which is unscalable. Mutual exclusion algorithms that currently exist do not have scalability and efficiency, which makes their applicability limited. This gives rise to new challenges that need to be tackled in order for this field to become successful in the future.

Today p2p and the Grid are in the same developmental stage, as traditional distributed systems were about a decade ago. Concepts like scalability and fault-tolerance need to be reworked for this new generation of distributed environment. One of the fundamental obstacles to overcome is to provide a mechanism to share resources transparently and efficiently across a large number of independent hosts. Resources can be either computational resources or data, and access to them should be controlled in a completely decentralized manner, even in the presence of high churn.

A few important characteristics that mutual exclusion protocols for p2p systems should demonstrate (in addition to the previously defined distributed mutual exclusion requirements) are:

- Scalability – Since 1000's of nodes can be actively participating in a p2p system at any given point in time, the protocol should be scalable with system size.
- Fault tolerant – Failure of nodes should be gracefully handled and should not pose a large background overhead. Also, node failures (of either replicas or other nodes) should not break the correctness of the protocol.
- Churn resistant – The dynamic nature of p2p systems should be taken into account when designing protocols. Typical p2p systems experience high churn rate, i.e., nodes join and leave the network at a high rate.

1.1. Context

Since each resource can have multiple replicas, the problem in question becomes even more challenging. Access to that resource is controlled by a set of replicas. In order to access it, majority of the replicas must reach a consensus. The concept of a quorum set is defined to be a group of nodes such that the intersection of any two quorum sets must not be empty. A token ring approach to mutual exclusion is one where all the nodes are arranged in a ring formation and a token is constantly circulated. When a node acquires the token from its neighbor it checks to see if it is attempting to enter a critical region. If so, it enters the region, does all the work it needs to and leaves the region. Token is then passed to the next node in the ring.

1.2. Proposed Solutions

The mutual exclusion protocols presented in this thesis combine token and quorum based approaches, to provide efficient and reliable access to shared resources in dynamic p2p systems. The protocols can be incorporated with any generic p2p DHT and are scalable with system size even under high contention.

The algorithms are distributed such that every node has to do roughly the same amount of work to acquire a resource. Scalability is achieved by a route-based quorum scheme that reduces the burden of mutual exclusion on the replicas. The two protocols differ in the way they conform to the End-to-End argument [34], from the computer networks domain. This dictates the exact placement of the protocol functionality and also its internal state, within the p2p network.

The protocols are route-based in the sense that a quorum set is constructed for every replica, based on the route traversed by a request to reach that replica node. The replica's quorum set thus comprises of every intermediate node lying on the DHT-based route between the requester and the replica itself. This translates to O(log N) quorum set nodes per replica and O(log N) messages, where N is the total number of nodes present in the p2p network, using an underlying DHT routing scheme like Chord [35] or Pastry [32]. Additional nodes, based on heuristics such as including the leaf or neighborhood sets of the replicas, may be added to the replica's quorum set. Increasing the size of a quorum set with nodes likely to be en-route to the replica also reduces the load on the replicas. Moreover, doing so increases the probability of encountering an arbitrator node (i.e., a node that is in a voted state) when another node tries to route to the replica for obtaining mutual exclusion. The use of these quorum sets allow the protocols to scale well with system size since a large number of messages will be intercepted before they reach the replicas.

1.3. Contributions

The main contribution of this study is the design and detailed analysis of two novel protocols for achieving mutual exclusion in dynamic p2p systems. This goal is accomplished while maintaining a low message overhead and at the same time reducing the burden on the replicas of controlling access to the critical section, by distributing the load evenly among the quorum set nodes. This ultimately results in highly scalable, robust to contention and fault tolerant protocols. Another important contribution is its ability to be incorporated with any generic p2p DHT [35] [32] [41] [14], depending on the application requirements.

1.4. Outline

The following chapter describes the system model which will serve as the basis for all the protocols that are presented in this thesis. Chapter 3 sheds light into the previous and current research being done in the area of mutual exclusion and p2p systems. Chapter 4 provides a detailed description of the Sigma [18] [19] and the two proposed protocols. Chapter 5 provides a brief discussion of the correctness of the protocols and proposes ways of handling failures in them. Chapter 6 dives into analyzing the experimental results gathered from simulating the protocols on a "virtual" network. Finally Chapter 7 concludes the work and presents some research ideas that can be explored in the near future.

CHAPTER 2

System Model

The protocols presented in this paper are based on the following system model representing a dynamic p2p DHT:

- The basic entities in the system are called nodes (or peers).
- Each virtual resource (e.g., a file or a computational resource) corresponds to a set of nodes (i.e., replicas) that are responsible for granting access to that resource.
- It is possible for a node to access a resource if and only if each responsible replica grants access to that node.
- These are connected over a p2p DHT that allows any node to route a message to any other node.
- The replicas for a resource are always available, but their internal states may be randomly reset due to a crash-recovery failure. They rejoin the network with the same nodeId. The protocols must account for this when discussing safety and liveness guarantees.
- The number of clients is unpredictable and can be very large. Clients are not malicious.
- There may be high churn in the system – nodes may enter and leave the system anytime.

- Clients and replicas communicate via messages across unreliable channels. Messages can be replicated, lost, but never forged.

CHAPTER 3

Related Work

3.1. Mutual Exclusion

Distributed mutual exclusion protocols tend to fall into two categories as detailed in survey papers [29] and [39]. These include token based protocols [28] and quorum based protocols [31] [36] [20], which intersect at completely centralized exclusion. This study proposes a hybrid between these two sets of protocols, with a competitive level of performance and adherence to general guidelines established for such protocols.

The basic safety requirement guaranteed by any mutual exclusion protocol is that at most one node should be able to access a resource ay any given point of time, from the set of nodes contending for that resource. Starvation and deadlocks are possible in contention scenarios and every protocol should take these into account. Mutual exclusion algorithms are distributed when each node shares an equal responsibility in controlling access to the critical section and must do the same amount of work to achieve exclusive access. The liveness requirement states that every request made for mutual exclusion must eventually be granted. Performance metrics for these protocols include bandwidth (in terms of number of messages), load, synchronization delay, etc. Robustness – handling of failures – is an increasingly important issue especially concerning p2p systems. Additional desirable properties in the dynamic DHT domain are resistance to churn and resistance to variance in latency.

Previous work by researchers in the mutual exclusion domain have set certain performance thresholds and presented a few common tradeoffs. Maekawa's algorithm [20] improves upon Ricart and Agrwala's [31] completely distributed approach and Suzuki's algorithm for instance, by grouping nodes into overlapping sets and reducing the number of messages from O(N) to O(√N). Certain assumptions on topology can be used to reduce this to O(log N) messages [28]. [25] shows how quorum based protocols can be analyzed for a random distribution of nodes. [18] presents an initial attempt at an algorithm for achieving mutual exclusion in dynamic p2p systems, but lacks a detailed analysis of their work.

3.2. Peer-to-Peer Systems

Several applications have been built and deployed on p2p DHTs, such as distributed file systems and various resource sharing overlays such as POST [23], PAST [33], SCRIVENER [26], and SQUIRREL [13]. In the physics and medical communities for instance, large pools of networked computing resources are needed to solve computationally intensive tasks. Some currently active projects within these domains include the Grid Physics Network [11] and the Human Proteome Folding Project [38]. This gives rise to the challenge of coming up with efficient resource management and sharing mechanisms that scale appropriately. Some previous protocols developed for this purpose include the Grid [6] and SHARP [9]. The Grid is an infrastructure that allows clients to tap into a pool of resources, making it possible for them to perform computationally intensive tasks in a distributed fashion. It focuses on flexible, secure, and coordinated resource sharing among a dynamic group of nodes. Resource sharing is currently handled by placing certain constraints on the particular resource being shared [7]. An application-independent technique is desired to achieve seamless sharing of resources among various clients. SHARP, on the other hand, provides a framework for secure distributed resource management in an Internet-scale computing infrastructure, by means of issuing and granting tickets and leases.

Previous research conducted in efficiently routing messages to the nodes holding a particular resource include, the Chord and the Pastry protocols. Chord is a structured p2p DHT. By choosing its neighbors intelligently, it lowers the latency and message cost of routing (i.e., lookups and inserts) to just O(log N). It applies a consistent hash to peers' addresses and filenames (corresponding to files that are stored at each peer), to organize them into a logical ring. Since files are just a specific type of resource, a similar routing technique can be used to achieve mutual exclusion among the various resources being shared. Pastry is a p2p location and routing substrate that is similar to Chord, but arranges nodes based on their geographical locality.

PAST is a large-scale, p2p archival storage utility that provides scalability, availability, security and cooperative resource sharing. Files in PAST are immutable and can be shared at the discretion of their owner. Ivy [24] on the other hand is a multi-user read/write p2p file system that provides useful integrity properties without requiring users to fully trust either the underlying p2p storage system or the other users of the file system.

Prior work in distributed searching includes Gnutella [37], which is a protocol for distributed search in decentralized p2p architecture. It routes different messages (i.e., Ping, Pong, Query, QueryHit, or Push) within the overlay graph. A search is initiated by sending out a TTL restricted Query message; servents respond with a QueryHit message, if they are storing the requested data. By extending this protocol to keep track of all the nodes that a particular Query message has visited (before reaching its destination), we can build a quorum set out of such nodes. Kelips [12] is another p2p DHT that makes a tradeoff by consuming greater amount of memory and constant background communication in order to reduce file lookup times and increase resistance to failures and churn.

3.3. Pastry Routing Scheme

Each node in the Pastry network has a unique, 128-bit nodeId, and the set of existing nodeIds is uniformly distributed. Given a message and a key, Pastry reliably routes the message to the Pastry node with the nodeId that is numerically closet to the key, among all live Pastry nodes. To accomplish this, each node stores a routing table and a leaf set. The routing table is organized into ($\log_2^b$ N) rows with 2^b-1 entries in each row, where 2^b is the base of the nodeId's sequence of digits. Each of the 2^b-1 entries in row n of a routing table refer to a node whose nodeId matches the present node's nodeId in the first n digits, but whose n+1th digit has one of the 2^b-1 possible values other than the n+1th digit in the present node's nodeId. The leaf set maintains IP addresses of L/2 numerically larger and smaller nodeIds, where a typical value of L is 16. In each routing step, the current node forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current nodeId. If no such node is found in the routing table, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the current nodeId. Such a node must exist in the leaf set unless the nodeId of the current node or its immediate neighbor is numerically closest to the key, or L/2 adjacent nodes in the leaf set have failed concurrently.

Figure 3.3 shows the steps that a query takes, while being routed through the Pastry routing substrate. In the example below, node with id 65A1FC is trying to locate data using the key D46A1C.

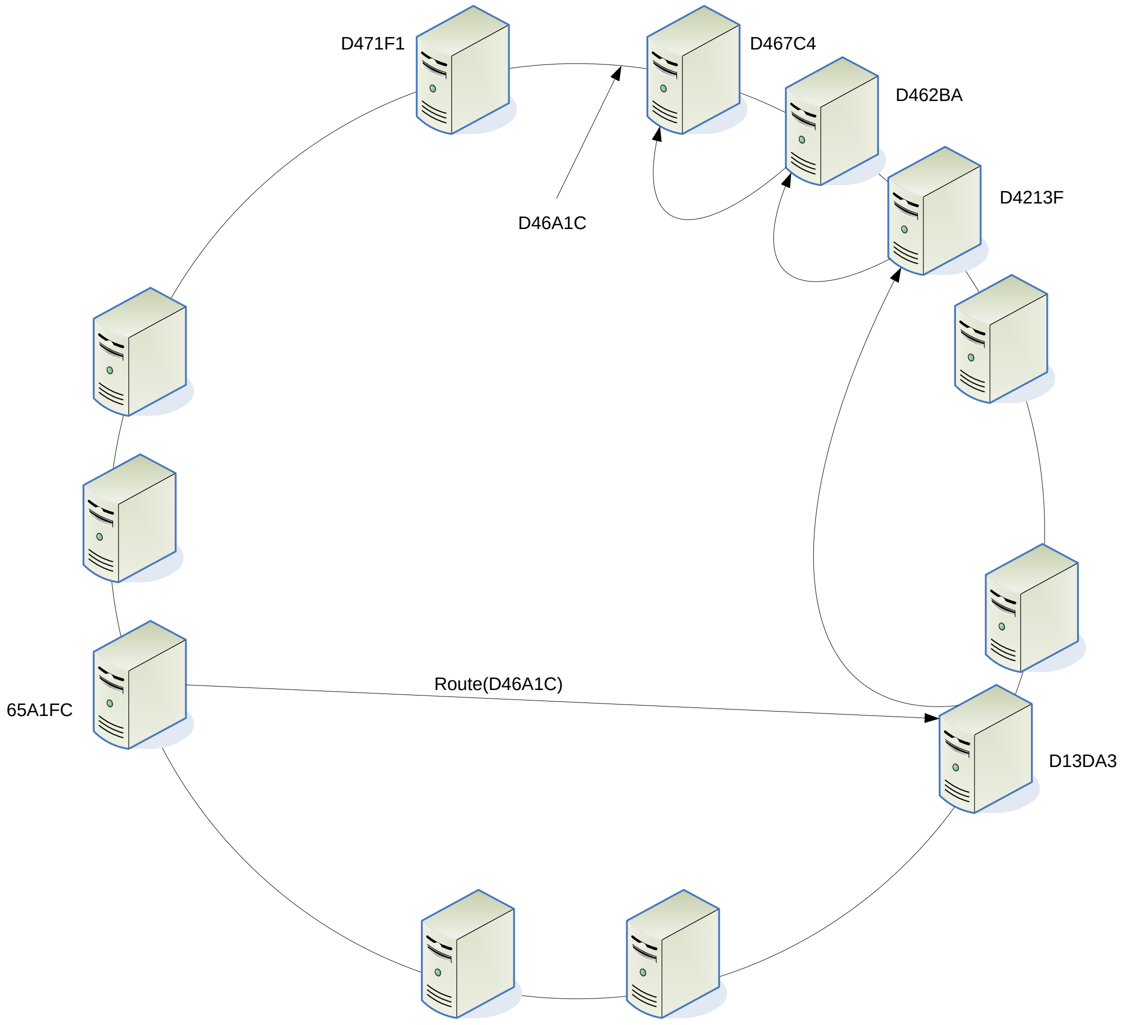


Figure 3.3: An example lookup for data with key D46A1C from node 65A1FC, using Pastry

CHAPTER 4

Protocols

4.1. Sigma Protocol

We discuss the only existing protocol for mutual exclusion, called Sigma [18]. Sigma is a protocol for providing mutual exclusion in dynamic p2p systems. It is implemented inside a p2p DHT and adopts queuing and cooperation between clients and replicas, in order to enforce a quorum consensus scheme. It utilizes the fact that nodes in the DHT collectively form a logical space that does not have holes, institute a set of *logical replicas* upon which a quorum consensus protocol grants access to the critical section. It deals with failures by two techniques: informed back-off and lease.

The protocol starts off by clients sending out REQUEST messages to all the responsible replicas for the resource that they need to access to. This message is handled by the OnRequest at the replica, which either grants its vote outright or queues the request up (depending upon if it has already voted or not). In either case, a RESPONSE message is sent containing the current owner's nodeId. As the RESPONSE messages arrive at the client, the following three outcomes are possible: (1) The client is the winner and therefore can access the critical section (2) Someone else wins, in which case the client just waits for its turn to access the resource (3) Nobody receives enough votes, in which case all the clients send out a YIELD message to each of the acquired replicas. Figure 4.1 below helps to graphically demonstrate the internals of this protocol.

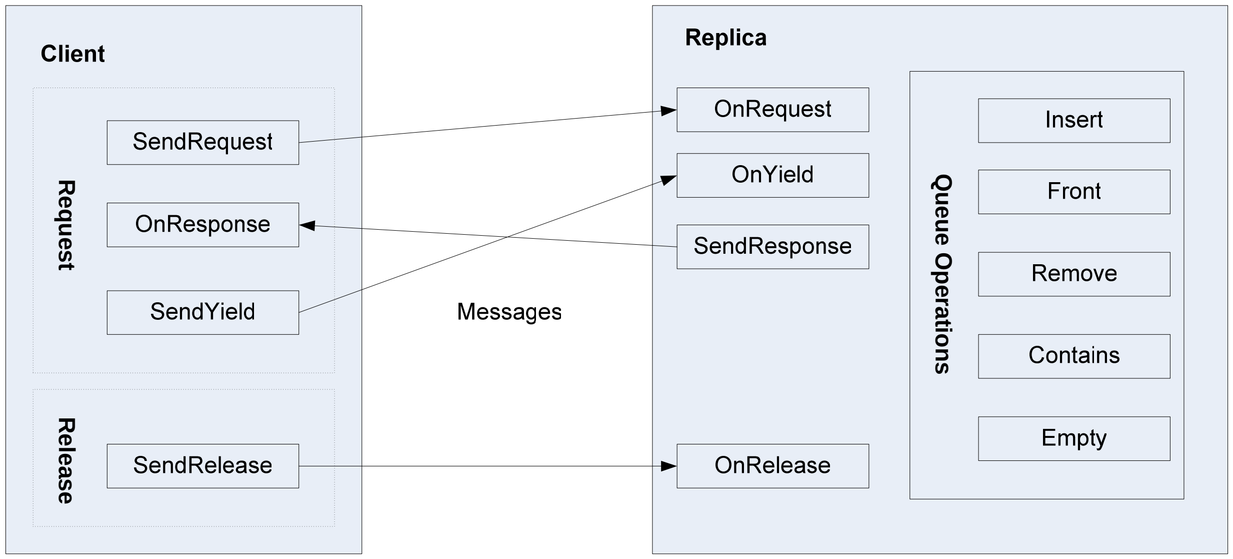


Figure 4.1: Architecture of Sigma Protocol

4.2. Proposed Protocols

The Sigma protocol presents a good start to thinking about mutual exclusion protocols for p2p systems, but it does not fully utilize the decentralized nature of this domain. It relies on the replica to maintain the queue of requests for the resource, leading to a less fault tolerant system due to a central point of failure and increased load on a few nodes. Along with addressing these problems, the following proposed solutions also achieve better performance and load balancing characteristics, which play an integral part in p2p systems.

The two protocols below differ in terms of how well they conform to the End-to-End argument. Comparing both protocols using this as one of the metrics helps to decide about the best placement for a mutual exclusion mechanism, in a p2p system.

4.2.1. End-to-End Mutual Exclusion Protocol

In order to achieve better load-balancing characteristics, this protocol maintains the queue of future requests at the node that is currently in the critical section, instead of at the replica (as was the case in the Sigma protocol). It defines the quorum set for gaining mutually exclusive access to a responsible replica R to be every node in the path from the requesting node to R. Since all the nodes within the quorum set maintain information regarding the current owner of the replica, requests for a resource that is already being held by another node can be satisfied by any of them, by means of an *ENQUEUE* message being sent to the current owner. When the node is done using a resource and leaves the critical section, it sends the queue of requests (i.e., token) to the node who requested the resource the earliest. This can easily be determined from the request queue, which is kept sorted by timestamps of request messages received.

The primary goals of this protocol are: (1) To achieve mutual exclusion with a low message overhead (i.e., lower bandwidth consumption) (2) Reduce the burden of the replicas for controlling access to the critical section.

The internal state information maintained by each node is as follows:

- A set of request queues (one queue for each resource currently being accessed by this node). Each entry of the request queue consists of the requester's nodeId and timestamp of that request. Each queue is kept sorted by Lamport timestamps [17], in order to maintain fairness and avoid starvation.
- A replica list maintaining (Replica Id, Owner Id) pairs to keep track of which replica is held by (i.e., voted for) which node, for the active quorum sets that this node is part of. *Next* and *Previous* node pointers are also maintained for each replica list entry to aid during node failures.
- List of resources this node currently has access to.
- Set of all nodes this replica has voted for. The set can contain more than one node since a node can be a replica for more than one resource.

The description of the end-to-end mutual exclusion protocol is as follows:

- When a node wants mutually exclusive access to a particular resource, it sends a *REQUEST* message, with the resource id of the needed resource, to all the replicas of that resource, using the underlying DHT routing mechanism. The set of responsible replicas for a given resource can be found using the Peer-to-Peer Replica Location Service (P-RLS) [2].
- Intermediate nodes lookup their replica list for the intended resource id. If found, the *REQUEST* message is stopped being forwarded and instead an *ENQUEUE* message is sent directly to the node currently accessing the

resource in context. Otherwise, the *REQUEST* message continues to be routed.

- Upon receiving the *REQUEST* message at the target node, that replica will check if it has voted already or not. A *RESPONSE* message is routed directly to the original sender of the *REQUEST* message. The *RESPONSE* message contains the id and timestamp of the replica's owner (i.e., the node this replica has voted for). If the replica has not voted before, this information would be that of the requesting node.
- Whenever a requesting node receives a *RESPONSE* message, it checks to determine if it has received a majority of replica votes. If so, it sends an *IAMWINNER* message to all the replicas, declaring itself as the new owner of the resource. *Next* and *Previous* node pointers (part of the replica list entry) are updated as this message is routed to all the replicas. If nobody has accumulated enough votes in a round, the requesters send out a *YIELD* (i.e., *RELEASE + REQUEST)* message to each of the replicas that voted for it.
- When an *ENQUEUE* messages reaches its destination (i.e., reaches the owner of the resource that is being requested), the owner adds the requester's id to its request queue.
- In order to release a resource, a *RELEASE* message is routed beginning at the current owner of the resource and is targeted for all the responsible replicas for that resource. The intermediate nodes that this message traverses through depend on the *Next* pointers of each node's replica list entry. Clean-up occurs

as this message is being routed to the replicas. Once the message reaches its target replica, owner information of that replica is reset.

- When the owner of the resource is done using that resource and leaves the critical section, it sends a *TOKEN* message to the node next in line, by removing the head of the queue. The *TOKEN* message that is sent contains the remainder of the request queue.

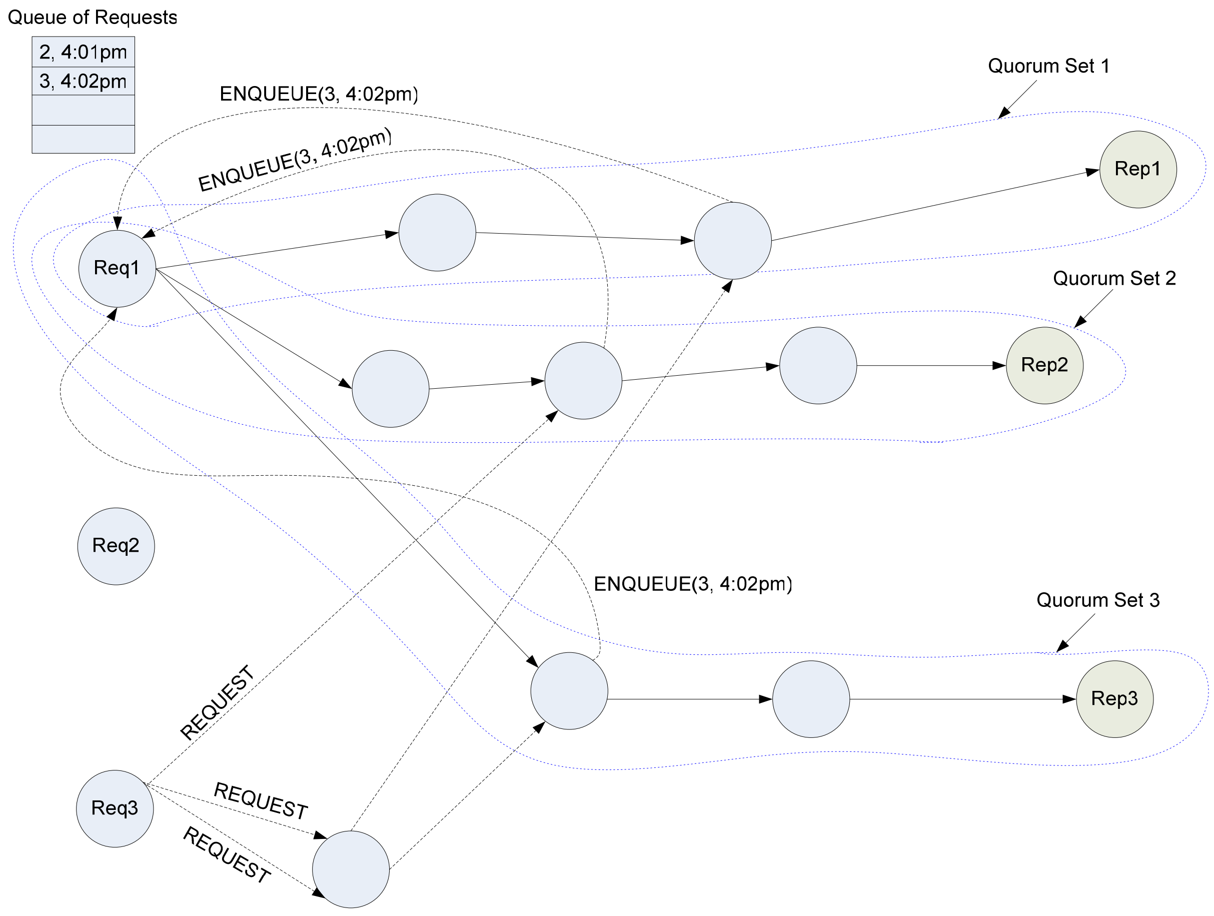


Figure 4.2.1: This is a snapshot of the E2E Protocol, where Req1 is currently in the critical section and Req2's request to use the same resource is queued up at Req1. Now Req3 sends out a request for the same resource. Its request is intercepted by quorum set nodes (before reaching the replica), and therefore ENQUEUE messages are sent to Req1 and it adds Req3 to its queue. The three quorum sets that exist in this scenario are marked above

4.2.2. Non End-to-End Mutual Exclusion Protocol

The fundamental idea behind this protocol is to modify the previous protocol, and instead maintain a partial queue of requests at all the nodes in the quorum set rather than a complete queue at only the accessing node. Information maintained in these partial queues can be consolidated when the current node exits the critical section, to determine the next node in line to use this resource.

The message overhead is low, so is the burden of achieving mutual exclusion on the replicas (just like the previous protocol). Furthermore, this protocol is more distributed and fault-tolerant, since the queue of requests previously being stored at one node is now split among the quorum set. The quorum set contains every node in the path from a requester X to the set of replicas R. This results in O(log N) nodes in the quorum set and O(log N) message overhead, where N is the total number of nodes present in the system.

The internal state information maintained by each node is as follows:

- A replica list maintaining (Replica Id, Node Id, Replica Request Queue, Next, Previous) entries. Used to keep track of which replicas are held and subsequent requests for them, for the active quorum sets that this node is part of.
- List of resources this node currently has access to.

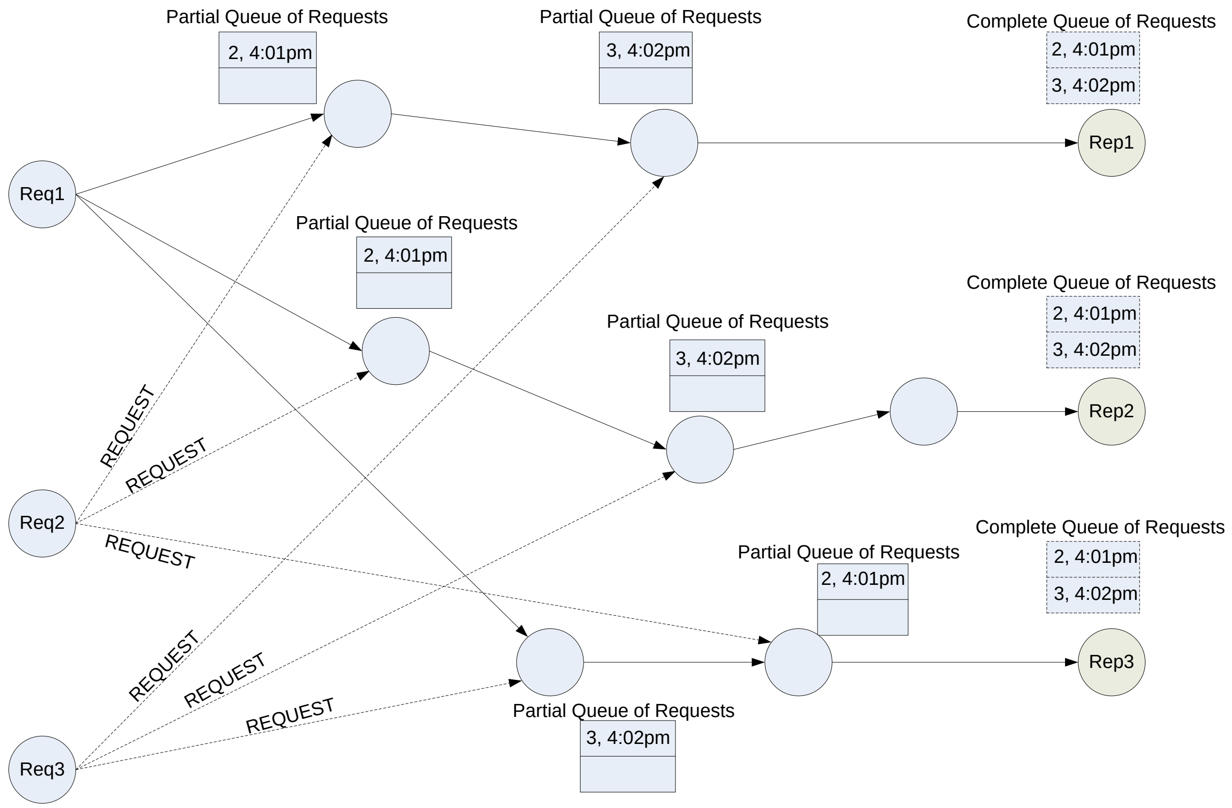
- Set of all nodes this replica has voted for. The set can contain more than one node since a node can be a replica for more than one resource.

The description of the non end-to-end mutual exclusion protocol is as follows:

- When a node wants mutually exclusive access to a particular resource, it sends a *REQUEST* message, with the resource id of the needed resource, to all the replicas of that resource, using the underlying DHT routing mechanism.
- Intermediate nodes lookup their replica list for the intended replica id. If found then the request is queued locally on the node and a *GRANTREQ* message is sent to the requester, containing the current holder of the replica. Otherwise, their replica list is updated and the *REQUEST* message continues to be routed.
- Upon receiving the *REQUEST* message at the target node (i.e., a replica), the replica will check if it has already granted access to the required resource or not. If it has, the request is queued on the replica itself. In either case, a *GRANTREQ* message is routed directly to the requester, specifying the holder of the replica (which will be the requester if the replica has not voted before).
- In order to release a resource, *RELEASE* messages are routed to all the responsible replicas for that resource. All quorum set members update their replica list by deleting the respective entry from it and forward the message together with a list of requests, sorted by timestamp, seen so far by any of the nodes prior to them in the sequence from the holder to the replica. When this

message reaches its target replica, it contains every request seen so far for that replica, and is merged with the local queue. The next owner of this replica is then determined from this newly assembled queue and *GRANTREQ* message is sent directly to that node.

- When all *GRANTREQ* messages reach their destination, the requester checks if it has attained majority of the replica votes, to enter the critical section. If no requester receives majority of the votes, all of them propagate a *YIELD* message to the replicas. Otherwise only one of the requesters will receive the majority votes and can therefore access the resource.
- The *YIELD* operation reflects the collaborative nature of this protocol and is used to reshuffle the queue. The fact that nobody wins indicates that contention has occurred. The *YIELD* message allows all requesters to try to acquire the resource again after a random time period. Typically, this self-stabilization process will quickly settle, as verified in [19].



Partial Queue of Requests
2, 4:01pm
Partial Queue of Requests
3, 4:02pm
Complete Queue of Requests
2, 4:01pm
3, 4:02pm
Rep1
Req1
Partial Queue of Requests
2, 4:01pm
Complete Queue of Requests
2, 4:01pm
3, 4:02pm
Partial Queue of Requests
3, 4:02pm
Rep2
REQUEST
REQUEST
Req2
REQUEST
Complete Queue of Requests
Partial Queue of Requests
2, 4:01pm
2, 4:01pm
3, 4:02pm
REQUEST
REQUEST
REQUEST
Rep3
Partial Queue of Requests
3, 4:02pm
Req3

Figure 4.2.2: In the snapshot of the Non E2E Protocol above, Req1 is currently in the critical section. Req2 and Req3 have issued *REQUEST* messages to also try to gain access to the same resource. Their requests are queued by the intermediate nodes of the respective quorum sets. When Req1 is done using the resource and leaves the critical section, the partial queues will be merged (within each quorum set) to form the complete queue of requests at the replicas

CHAPTER 5

Handling Failures and Discussion of Correctness

5.1. Failure Handling

The protocols described above work well under failure-free environments. This section describes ways of detecting and recovering from failures. The types of failures that are being considered in the analysis below include: (1) Failure of node currently present in the critical section (2) Failure of node(s) maintaining the queue of requests (3) Failure of intermediate nodes of the quorum set. In all the protocols, lost messages are detected using a “ping/ack” scheme, with a constant overhead. High churn, which is common in a typical p2p system, can also be categorized under node failures and handled by similar mechanisms.

5.1.1. End-to-End Mutual Exclusion Protocol

Failure of the node that is currently in the critical section and is therefore also holding the token (i.e., maintaining the queue of requests), can have a devastating effect on the entire system. A very clever scheme has been proposed to handle this kind of failure. Since the request queue maintains information for nodes that are waiting to use a particular resource, why not replicate and maintain the same queue of requests at the first n waiting requesters (based on timestamps of the requests submitted)? These n+1 nodes (including the node currently accessing the resource) can run a membership protocol within themselves, to propagate updates and also to determine if a particular node has failed. Whenever the request queue is updated, this change is propagated to the other n nodes of its membership. These updates are ordered by sequence numbers so that, in case of failure, we can use the most current copy of the request queue. “Ping” messages are sent periodically from n waiting nodes to the node holding the token, at a low frequency, for example 1 in every 10 minutes. This means that the failure of that node can be detected in just 10/n minutes. When the node holding the token fails, among the waiting nodes, the node with the highest sequence number will have the most updated copy of the queue. This node routes a message to all the replicas notifying itself as the owner of the resource. This failure detection/correction mechanism is tolerant to the failure of n waiting nodes, where n is a parameter that can be set in order to come up with the ideal tradeoff between extra bandwidth consumption and fault-tolerance, for a particular application criteria.

Failure of intermediate nodes within a quorum set is detected using the *Next* and *Previous* pointers maintained by each node, as part of its replica list entry. In case of failure, queries are re-routed by the underlying Pastry layer using an alternate set of nodes.

5.1.2. Non End-to-End Mutual Exclusion Protocol

Since the request queue is being maintained by all the intermediate nodes of the quorum set, therefore the queue of requests would not be lost by a single node failure. In case a node in the path fails, its predecessor routes a new request message to the replica, creating a new quorum path, and its successor directs a cleanup message to clear the old quorum nodes from the failed node to the replica. Since each node maintains *Next* and *Previous* node pointers, detecting predecessor/successor failures is achievable.

During a *RELEASE* message sequence, it can happen that the replica fails after receiving the partially assembled queue from its predecessor. The failure of the replica can be detected by the "ping/ack" mechanism between the replica and its predecessors. In such a case, an alive predecessor would act as the temporary token holder, where the new queue would be rebuilt.

5.2. Discussion of Correctness

Correctness of p2p mutual exclusion algorithms can be guaranteed if the safety and liveness requirements of distributed mutual exclusion (as presented in Chapter 1) are upheld. In case of failures, only probabilistic correctness can be guaranteed by the proposed protocols. Fairness is maintained if requests to access a particular resource are granted in FIFO ordering.

5.2.1. End-to-End Mutual Exclusion Protocol

Without replica failures the end-to-end protocol satisfies mutual exclusion since no two nodes can ever be in a critical section at the same time. Assuming the contrary, i.e., two nodes are simultaneously in the critical section for a resource then both nodes must have received a majority of the votes from the set of replicas that control the resource. This is not possible given that each vote is sent explicitly as a *RESPONSE* message either directly from the replica or as a *TOKEN* message from the node that previously held the resource.

In case of a replica failing and randomly resetting its state, it may happen that subsequent requests reach the replica resulting in a grant. Considering a resource with M replicas needed for mutual exclusion, the probability of such a series of events breaking mutual exclusion is given by: $P_f = 1 - (1-P_{rr})^M$, where P_{rr} is the probability of a single replica failing and randomly resetting its state. The above equation holds as long as Pastry routing works correctly.

Deadlock is avoided by using timestamps along with introducing release timeouts for nodes which receive fewer grant messages than needed. Starvation is avoided by ordering requests, by timestamp.

5.2.2. Non End-to-End Mutual Exclusion Protocol

It is very difficult to analyze proof of correctness on failures of intermediate nodes in Pastry. However, our discussion is conservative in allowing requesters to retry, thus ensuring liveness and safety always.

When nodes do not fail, a similar argument as above can be applied in order to guarantee that mutual exclusion is not broken; the queue of requests can be completely built up and the next owner of the resource can be chosen from it. This guarantees fairness and upholds exclusive access principle since a node can get access to a resource if and only if it receives majority of the *GRANTREQ* messages. If no node receives a majority, all of them send out a *YIELD* message and try their requests again after a random time period.

In case of node failures, partial queue maintained at that node will be lost. This can result in fairness property to be broken, but the correctness of the algorithm is still upheld (since only one node can receive a majority of the replica votes). Those nodes (whose requests have been lost) will be able to detect this from the “ping/ack” protocol running between nodes, and can therefore send out their requests again. This latter mechanism helps to maintain the fairness property of the protocol.

CHAPTER 6

Experimental Results

6.1. Set-Up

The End-to-End (E2E), Non End-to-End (NonE2E), and Sigma mutual exclusion protocols have been implemented over the FreePastry [8] implementation of the Pastry routing substrate. This allows us to accurately compare our proposed solutions against the Sigma protocol in various different network scenarios, using a virtual network simulation model with message-based communication.

The test-bench code starts off by creating a set of nodes, arranged according to the Pastry network topology. The simulation then runs for a certain number of rounds and derives the experimental results, by averaging over ten such runs. In the simulation, the concept of a *round* is introduced in order to describe the amount of work done by the nodes in the network, which includes a certain number of requests for resources being issued by a set of randomly chosen nodes. Also in each round, a node releases one of its already held resources, with a 50% probability. This means that the average holding time of a critical section is 2 rounds. The parameters that can be tuned in the simulation include: (1) Number of nodes (2) Number of resources (3) Number of replicas per resource (4) Number of rounds (5) Number of requests issued per round.

Below are the default values for the simulation parameters, which will be used to generate the plots in the following sections:

Number of Nodes	Number of Resources	Number of Replicas per Resource	Number of Rounds	Number of Requests per Round
6500	65	10	20	20

Table 6.1: Default values for parameters used to generate the plots

The programming language used for implementing these protocols was Java version 1.4.2. The experimental data presented below was collected by running the protocols on a 2.4 GHz, 768 MB Windows XP machine.

6.2. Summary of Results

6.2.1. Scalability

One of our initial goals behind the proposed protocols was to come up with an efficient and scalable method for achieving mutual exclusion. This means that as the p2p network grows, the increase in overall bandwidth consumption should be minimized. The increase in bandwidth is caused by the increase in resources and clients needing access to those resources. Figure 6.2.1 (a) presents a comparison of the three protocols. Even though all the plots are linear with respect to number of nodes, the two proposed protocols require 1.5 to 2.5 times less messages than the Sigma protocol, which is a big improvement.

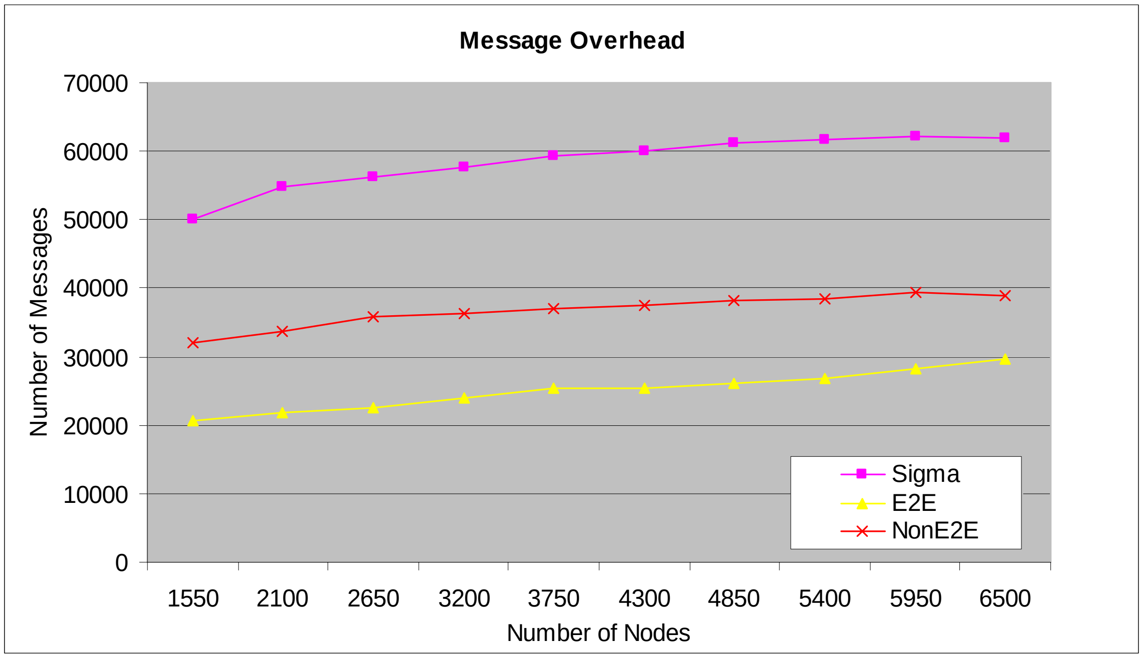


Figure 6.2.1 (a): A lower increase in bandwidth consumption, as the network grows

Figure 6.2.1 (b) shows the effect of increased network contention on the amount of bandwidth consumed. The number of nodes present is kept constant at 6500, but the rate of requests per round is steadily increased from 10 to 100. This is a nice measure as it demonstrates how the protocols would perform under heavy loads.

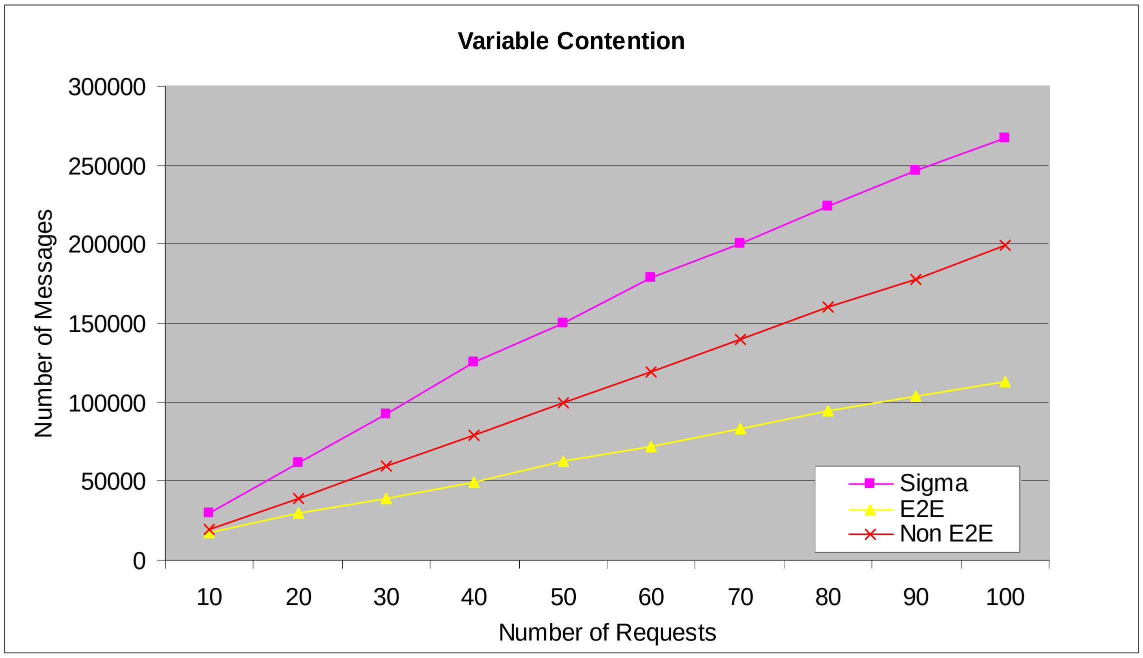


Figure 6.2.1 (b): Effect of increasing the number of requests issued per round on overall bandwidth consumption

Even though the number of *ENQUEUE* messages increases linearly with increasing load, the number of *tokens* sent remains constant over the same period, as shown in Figure 6.2.1 (c) plot. This is a good indication that passing of *tokens* (which contains a queue of waiting requests) around does not significantly increase the overall message overhead. Therefore no matter how many requests are queued up, the amount of bandwidth consumed is bounded and almost remains constant.

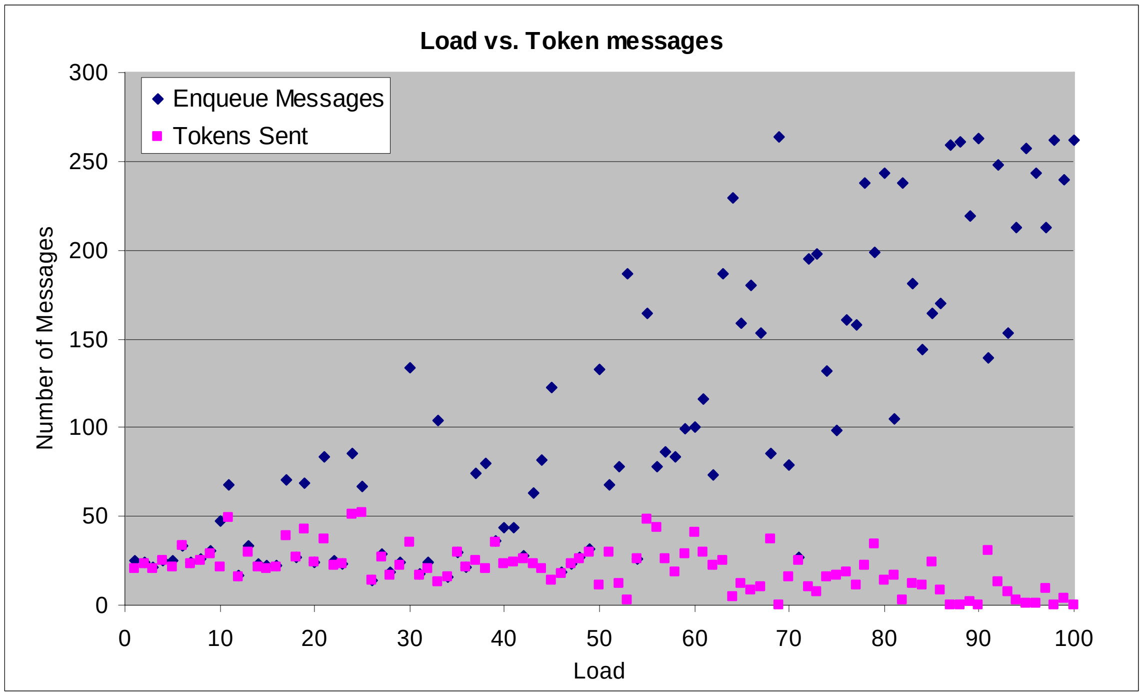


Figure 6.2.1 (c): Effect of increasing network load on the number of TOKEN and ENQUEUE messages generated, for the E2E Protocol. The number of nodes being considered here is 100 (to keep the plot uncluttered)

6.2.2. Reduced Replica Load

In a truly distributed protocol, each node should perform roughly the same amount of work and the responsibility of performing a particular task should be divided evenly amongst the nodes. In the Sigma protocol this was not the case. Responsible replicas for a given resource were given much more responsibility (i.e., greater load) when access to that resource was needed. On the other hand, both our proposed protocols reduce the load on the replicas, by dividing this responsibility among the corresponding quorum set nodes. Figure 6.2.2 shows a comparison plot of the three protocols. The proposed End-to-

End mutual exclusion protocol performs the best in this scenario (i.e., minimizes the load on the replicas). We are noticing this pattern because in the E2E protocol we are maintaining the queue of requests at the requester node (instead of it being stored at the replica, as was the case in Sigma). Due to our clever quorum set scheme, any one of the quorum set members can handle the request, thus reducing the replica load.

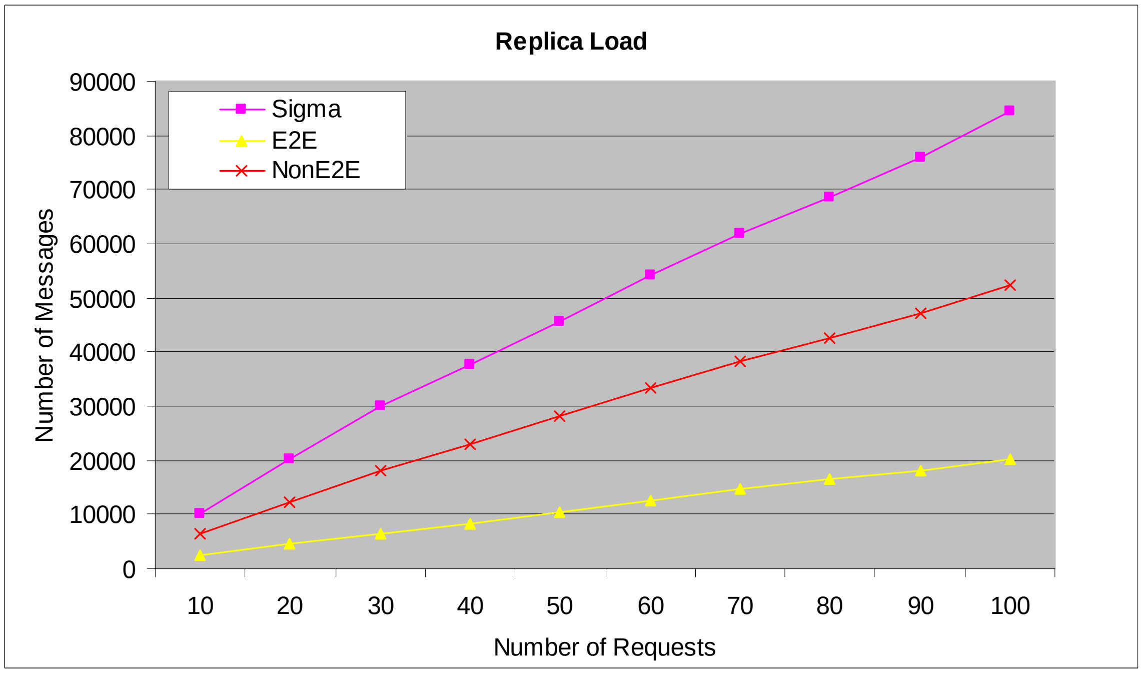


Figure 6.2.2: Comparison of the three protocols in terms of the load each of them places on the responsible replicas

6.2.3. Overall Load Distribution

In order to demonstrate the even load distribution of the proposed protocols, the following plots show the Cumulative Distribution Functions for them. The simulation parameters being used to generate the plots are shown in Table 6.2. Two CDF plots are generated for each protocol, which differ in the number of replicas that are responsible for granting access to a particular resource. The term load (as used in the plots below) refers to the number of messages that a node has to process.

Number of Nodes	**Number of Resources**	**Number of Replicas per Resource**	**Number of Rounds**	**Number of Requests per Round**
1000	10	10, 40	20	20

Table 6.2: Default values for parameters used to generate the CDF plots

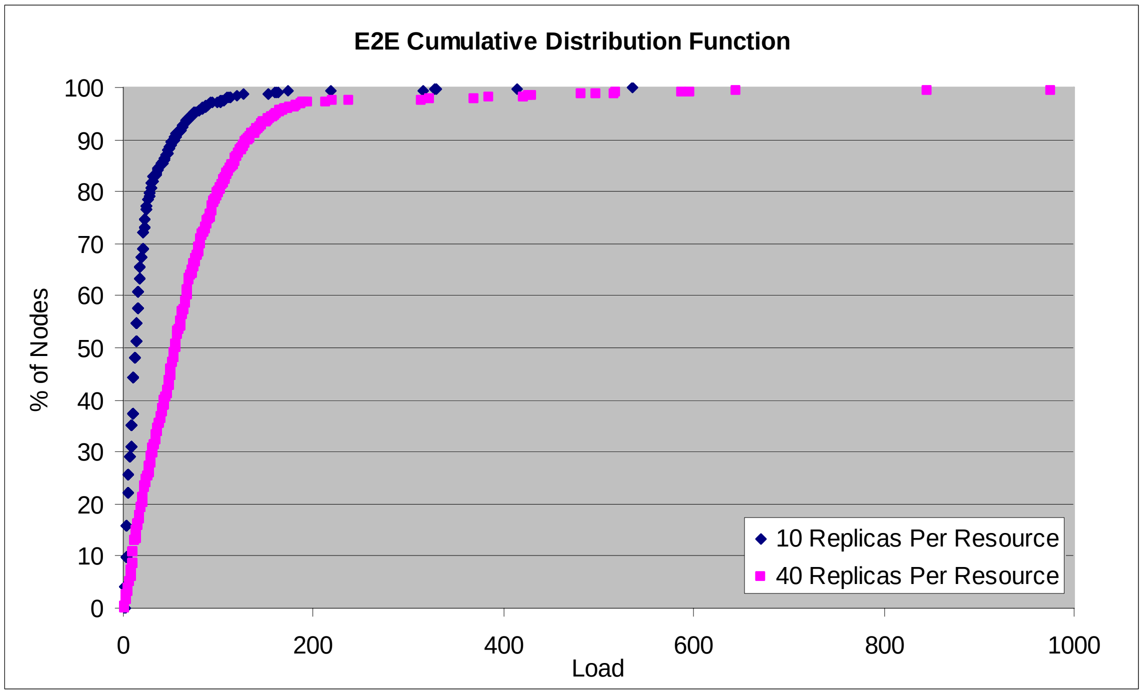


Figure 6.2.3 (a): Majority of the nodes have a load of 200 messages/round or less, except for a few outliers. By increasing the number of responsible replicas for a resource, the load on each node increases. This behavior is expected since the number of request/response messages exchanged between the replicas and the requesters would now be greater

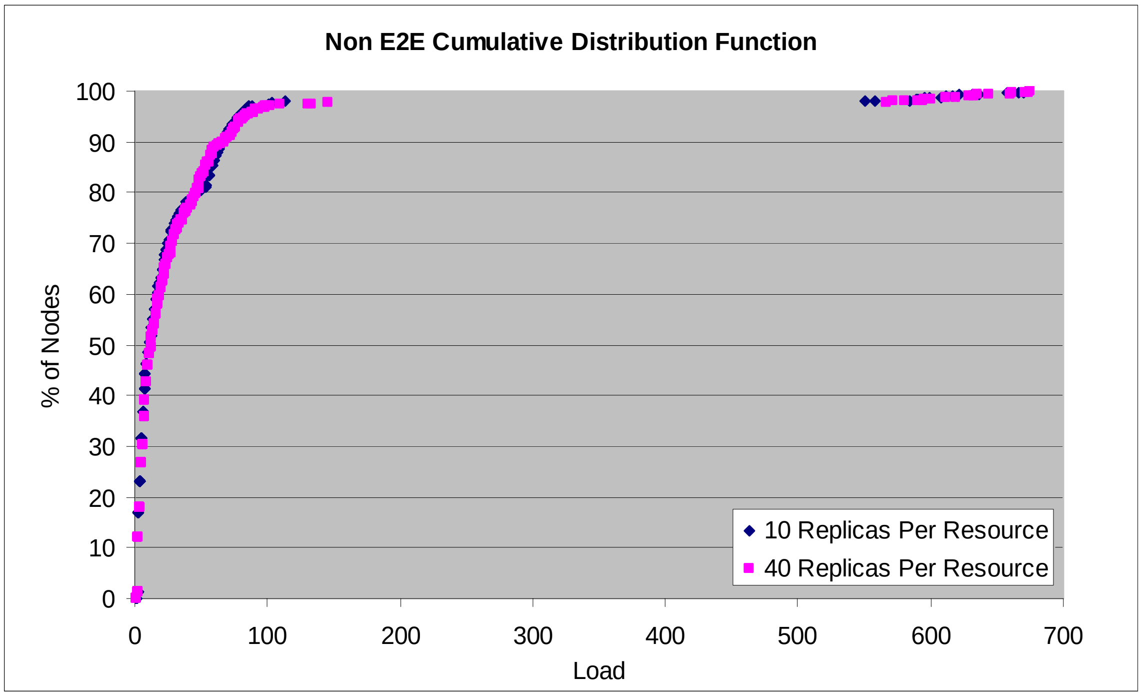


Figure 6.2.3 (b): A majority of the nodes experience a load of less than 180 messages/round. Unlike the E2E CDF plot, this one has a slightly larger number of nodes with loads in the range of 550-680 messages/round. Increasing the number of responsible replicas per resource does not have any effect on the load experienced by each node. This behavior is explained by the fact that in the Non E2E protocol, request messages are intercepted and stopped quicker than E2E (especially when number of replicas per resource is increased)

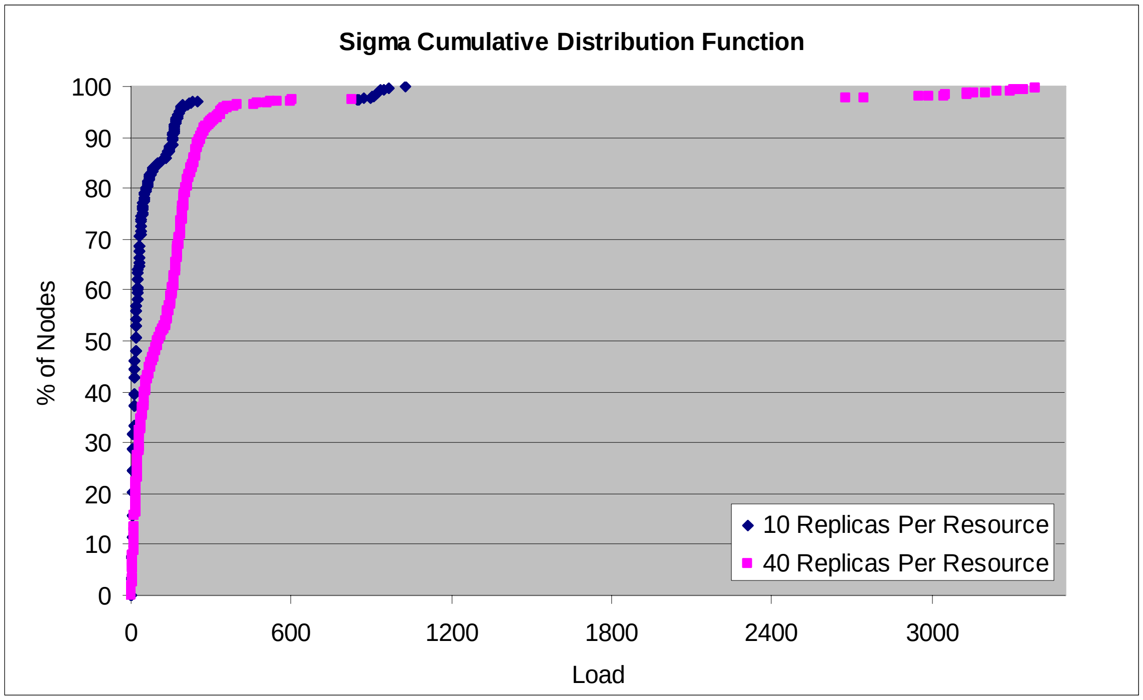


Figure 6.2.3 (c): A majority of the nodes experience a load of 700 messages/round or less. Similar to the NonE2E CDF plot, this one has a slightly larger number of nodes with loads in the range of 2500-3500 messages/round. Increasing the number of responsible replicas per resource has similar effect, as was viewed for the E2E Protocol in Figure 6.2.3 (a)

6.2.4. High Churn Rate

Rate of churn is defined as the sum of the number of nodes that are in transition (i.e., are either leaving or joining the system per round). In our simulation, n/2 randomly chosen nodes are failed and n/2 new nodes are added to the system, where n is the desired churn rate. The parameters used to generate the following plots are consistent with Table 6.1. Requesters are not churned in our experiments.

Figure 6.2.4 (a) shows the effect of increasing the rate of churn on the message overhead (i.e., bandwidth consumption), while Figure 6.2.4 (b) shows its effect on the time it takes to enter the critical section (after sending out a REQUEST message, when no other resource is trying to get access to that same resource).

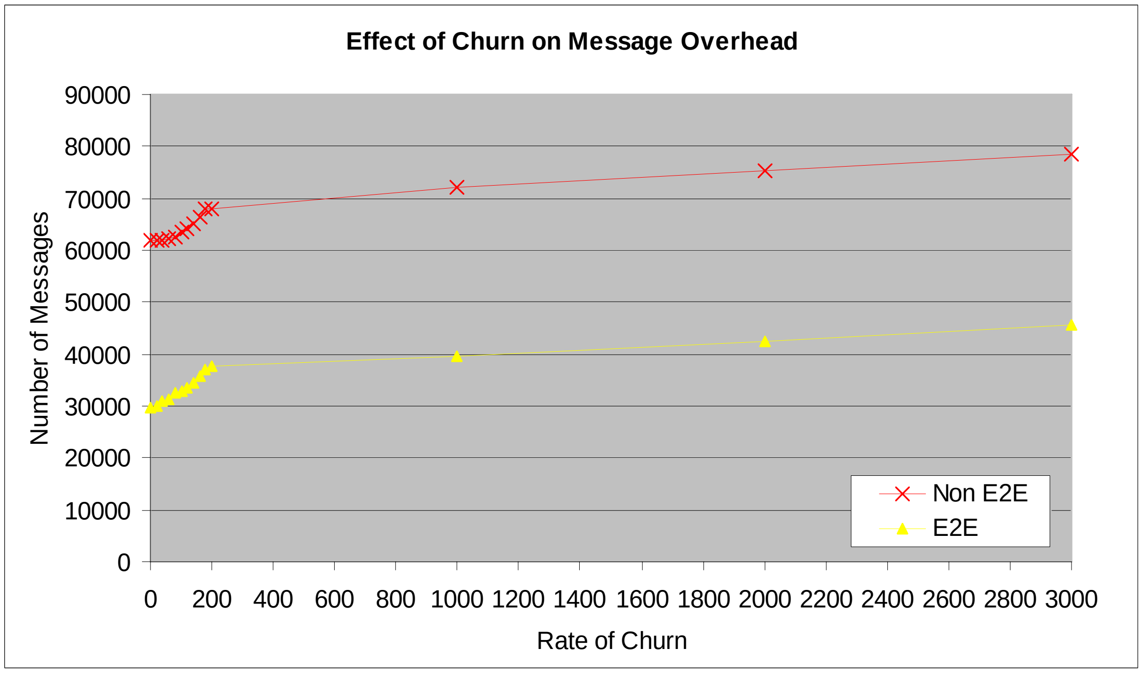


Figure 6.2.4 (a): As the rate of churn is increased from 0 to 200, the change in message overhead is shown in this plot. Both the protocols experience a slightly higher increase in message overhead (at first), but then increase at a much lower rate

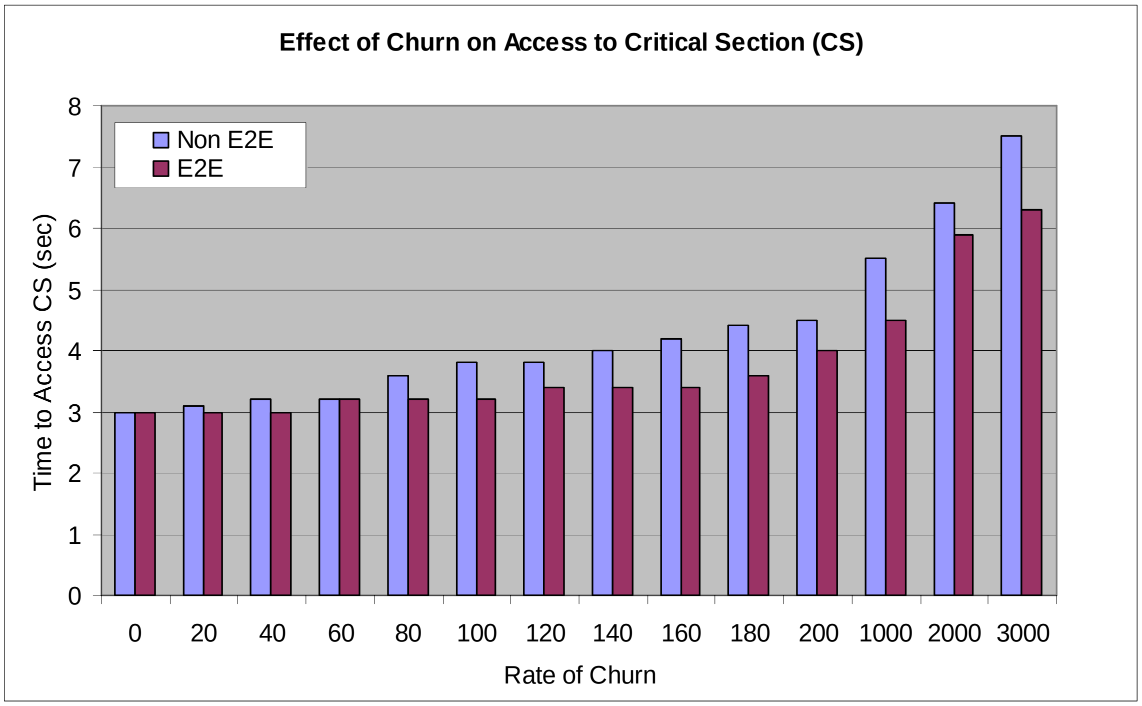


Figure 6.2.4 (b): By increasing the rate of churn, the time it takes for a node to get access to a resource (after sending a REQUEST message) ranges from 3 seconds all the way up to 7.5 seconds

CHAPTER 7

Closing Remarks

7.1. Conclusion

In this work, a couple of protocols for achieving mutual exclusion in a p2p system were presented, one of which conformed more to the End-to-End argument. Both protocols can be easily configured to run on any DHT, therefore providing a truly generalized solution to the addressed problem. The purpose behind presenting two different approaches to the same problem was to present the inherent tradeoff that exists between the amount of bandwidth consumed and better load balancing and fault-tolerance level required. An educated decision, based on specific application needs, must be made in order to fully utilize the potential of the chosen protocol. The proposed protocols, through their truly novel quorum and token-based schemes, were able to keep the load on the replicas relatively low even in the presence of a growing network and high churn rates.

In order to further strengthen and quantify our arguments, a detailed experimental analysis was conducted which compared the performance (in terms of bandwidth consumption and load) of the three protocols. Based on the results presented in Chapter 5, the proposed protocols performed the simulation tasks with 2 times fewer messages and also evenly divided the load among all the nodes.

7.2. Future Work

An important future direction of this research would be to port some existing Grid or p2p applications to use these protocols for handling their mutual exclusion needs. The ideal goal would be to present a general set of APIs for our protocols so that future applications can make use of our algorithms in a modular fashion, without worrying about the low-level mutual exclusion details.

References

[1] Bjurefors, F., Larzon, L., and Gold, R., *Performance of Pastry in a Heterogeneous System*, The Fourth IEEE International Conference on Peer-to-Peer Computing, 2004.

[2] Cai, M., Chervenak, A., and Frank, M., *A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table*, Proceedings of the SC2004 Conference, November 2004.

[3] Castro, M., Druschel, P., Kermarrec, A., and Rowstron, A., *SCRIBE: A Large-Scale And Decentralized Application-Level Multicast Infrastructure*, IEEE Journal on Selected Areas in Communications (JSAC), 20(8), October 2002.

[4] Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., and Stoica, I., *Wide-Area Cooperative Storage With CFS*, SOSP, 2001.

[5] Dijkstra, E.W., Solution of a Problem in Concurrent Programming Control, Communications ACM 8, 9 (Sept. 1965), 569.

[6] Foster, I., *The Grid: A New Infrastructure for 21st Century Science*, Physics Today, 2002.

[7] Foster, I., Kesselman, C., and Tuecke, S., *The Anatomy of the Grid Enabling Scalable Virtual Organizations*, Intl J. Supercomputer Applications, 2001.

[8] *FreePastry*. http://freepastry.rice.edu/FreePastry.

[9] Fu, Y., Chase, J., Chun, B., Schwab, S., and Vahdat, A., *SHARP: An Architecture For Secure Resource Peering*, SOSP 2003.

[10] Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I., *Load Balancing in Dynamic Structured P2P Systems*, INFOCOM 2004, Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, March 2004, 2253-2262.

[11] *GridPhyN*, http://www.griphyn.org.

[12] Gupta, I., Birman, K., Linga, P., Demers, A., and Renesse, R., *Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead*, IPTPS, 2003.

[13] Iyer, S., Rowstron, A., and Druschel, P., *SQUIRREL: A Decentralized, Peer-to-Peer Web Cache*, PODC, 2002.

[14] Kato, D., *GISP: Global Information Sharing Protocol — A Distributed Index for Peer-to-Peer Systems*, Second International Conference on Peer-to-Peer Computing, 2002.

[15] *Kazaa*. http://www.kazaa.com.

[16] Lam, S.S., and Liu, H., *Failure Recovery for Structured P2P Networks: Protocol Design and Performance Evaluation*, ACM SIGMETRICS Performance Evaluation Review 32, June 2004, 199-210.

[17] Lamport, L., *Time, Clocks and the Ordering of Events in a Distributed System*, Communications of the ACM 21, July 1978, 558-565.

[18] Lin, S., Lian, Q., Chen, M., and Zhang, Z., *A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems*, International Workshop on Peer-to-Peer Systems (IPTPS), 2004.

[19] Lin, S., Lian, Q., Chen, M., and Zhang, Z., *A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems*, Microsoft Research, Technical Report MSR-TR-2004-13.

[20] Maekawa, M., *A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized Systems*, ACM Transactions on Computer Systems (TOCS). 1985.3.2.

[21] Mahajan, R., Castro, M., and Rowstron, A., *Controlling the Cost of Reliability in Peer-to-Peer Overlays*, IPTPS, 2003.

[22] Maniatis, P., Rosenthal, D.S.H., Roussopoulos, M., Baker, M., Giuli, TJ., and Muliadi, Y., *Preserving Peer Replicas By Rate-Limited Sampled Voting*, SOSP, 2003.

[23] Mislove, A., Post, A., Reis, C., Willmann, P., and Druschel, P., *POST: A Secure, Resilient, Cooperative Messaging System*, USENIX HotOS IX, 2003.

[24] Muthitacharoen, A., Morris, R., Gil, T.M., and Chen, B., *Ivy: A Read/Write Peer-to-Peer File System*, OSDI, 2002.

[25] Naor, M., and Wieder, U., *Scalable and Dynamic Quorum Systems*, Principles of Distributed Computing (PODC), 2003.

[26] Ngan, T., Wallach, D.S., and Druschel, P., *Enforcing Fair Sharing of Peer-to-Peer Resources*, IPTPS'03, February 2003.

[27] Ntarmos, N., and Triantafillou, P., *SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Networks*, The Fourth IEEE International Conference on Peer-to-Peer Computing, 2004.

[28] Raymond, K., *A Tree-Based Algorithm for Distributed Mutual Exclusion*, ACM Transactions on Computer Systems (TOCS). 1989.7.1.

[29] Raynal, M., *A Simple Taxonomy for Distributed Mutual Exclusion Algorithms*, ACM SIGOPS Operating Systems Review. 1991.25.2.

[30] Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J., *Handling Churn in DHT*, Proceedings of the General Track: 2004 USENIX Annual Technical Conference, July 2004.

[31] Ricart, G., and Agrawala, A.K., *An Optimal Algorithm for Mutual Exclusion in Computer Networks*, Communications of the ACM. 1981.24.1, 9-17.

[32] Rowstron, A., and Druschel, P., *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*, Middleware, 2001.

[33] Rowstron, A., and Druschel, P., *Storage Management and Caching in PAST, A Large-Scale*, P*ersistent Peer-to-Peer Storage Utility*, ACM Symposium on Operating Systems Principles (SOSP), 2001.

[34] Saltzer, J.H., Reed, D.P., and Clark, D.D., *End-to-End Arguments in System Design*, ACM Transactions in Computer Systems 2, 4, November, 1984, 277-288.

[35] Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H., *Chord: A Scalable Peer-to-Peer Lookup Service For Internet Applications*, Special Interest Group for Data Communications (SIGCOMM), 2001.

[36] Suzuki, I., and Kasami, T., *A Distributed Mutual Exclusion Algorithm*, ACM Transactions on Computer Systems (TOCS). 1985.3.4.

[37] *The Gnutella protocol specification v 0.4*, Document revision 1.2, www.clip2.com.

[38] *The Human Proteome Folding Project*, http://www.grid.org/projects/hpf.

[39] Velazquez, M.G., *A Survey of distributed mutual exclusion algorithms*, Technical Report CS-93-116, Colorado State University, 1993.

[40] Walkerdine, J., and Rayson, P., *P2P-4-DL: Digital Library over Peer-to-Peer*, The Fourth IEEE International Conference on Peer-to-Peer Computing, 2004.

[41] Zhao, B.Y., Kubiatowicz, J.D., and Joseph, A.D., *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*, UCB Tech. Report UCB/CSD-01-1141.