LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

"Utopia lies at the horizon.
When I draw nearer by two steps,
it retreats two steps.
If I proceed ten steps forward, it
swiftly slips ten steps ahead.
No matter how far I go, I can never reach it.
What, then, is the purpose of utopia?
It is to cause us to advance."

— Eduardo Hughes Galeano

Author of *Open Veins of Latin America*

It has been a very long road to reach this milestone. Too many times it seemed even utopic that I would make it, but here I am. At this point, I believe it is important to stop for a few minutes and reflect on why and how I got here. The why has always been easy, I want to use my field of expertise (i.e., telecommunications) to improve the livelihood conditions in my home country, México. The how is more complicated, but I believe that at least I have to acknowledge and thank all the people who has been fundamental in the completion of this milestone. They have provide me with guidance, support, inspiration and love throughout (and before) this process. I will dedicate the rest of this section solely to this.

I want to thank my advisor Dr. Taieb Znati for his guidance in the development of this research and his continuous encouragement to improve it. In addition, I want to thank my

dissertation committee for their input on my research work and their teachings during my PhD and Master program.

I am thankful for “my team” of editors that kindly donated their time to help me improve my written work. Sandy, Bedda, Emilio, Ruth, Julie and Daniel have certainly played a fundamental role in improving the quality of this document.

I am grateful for all the organizations and individuals that provided me with financial support during my graduate studies. First of all, my wife, who kept me afloat during the last two terms of this endeavor. My alma mater, the National Autonomous University of México (UNAM) and the former Central Academic Computing Division (DGSCA) now DGTIC where I grew professionally, academically and personally. I will always cherish the years I spent as an intern, staff, faculty and manager. The CONACYT and the Fulbright-Garcia Robles scholarship program for their financial support. The graduate program in telecommunications for granting me a full tuition waiver during my master studies. The European Union Center of Excellence and European Studies Center (EUCE/ESC) at the University of Pittsburgh, for the Graduate Student Assistant appointment (GSA) that supported me during most of my PhD studies. Special thanks to Dr. Alberta Sbragia, whose leadership defined the GSA policies of the EUCE/ESC and who always trusted in my technical expertise. Jorge Reyes, who trusted me to work with him despite my limited handyman experience. From him, I learned not only to be proud of what you do, but also that even the messiest project can be done orderly and cleanly. Bosch’s Research and Technology Center, where I was an intern. The projects I developed there were professionally fulfilling. The Vira I. Heinz (VIH) Program for Women in Global Leadership for granting me the opportunity to work freelance, editing their website.

I also want to thank the extraordinary group of people that have provided me with emotional support one way or another throughout this endeavor: my wife Sandy (obviously), my family at large (incluyendo a todos los enanos), my closest friends Bedda and Emilio (always generous and supportive) and my friends of the squash federation (lead by the brave David Brumble).

I definitively want to thank those people who inspire me and provided me with role models. My parents first of all. Next, my brother Gilberto, who bought for me my first computer and was the first in the family to get a PhD; then he became the Dean of Engineering and now the future Chancellor at his university. He certainly has been a great career model (even thought matching his career path appears utopic). My sister (Dra. nena) who has managed to sprung a successful academic career together with two bright and beautiful *chilpayates*. Also, professor Alberta Sbragia, who managed to balance her teaching, research and leadership position at the EUCE/ESC. Finally, Professor Daniel Mosse, for his exemplary academic attitude, always willing to help students, thought provoking comments and his upbeat personality.

1.0 INTRODUCTION

The Peer-to-Peer (P2P) networking paradigm comprises several alternative distributed architectures to build large-scale virtual networks (e.g., unstructured and DHT-based P2P networks[1]) on top of an existing network infrastructure, generally the Internet. The primary objective of these networks is to enable the sharing of services and content among participants. The resources and tasks needed for end-to-end communication are contributed and performed (with several possible levels of decentralization) by the participating nodes. Generally, P2P networks are open access self-organizing systems formed by volunteer nodes; nonetheless, individual nodes are autonomous and self-interested in nature.

P2P technology has emerged as a viable solution for the deployment of large-scale distributed applications such as content distribution and file sharing. Currently, deployed P2P systems reach several million users, with several hundred thousand peers[2] connected simultaneously [1]. Recent measurement studies indicate that P2P applications still make up the majority of the Internet traffic worldwide [2] and more than half of all upstream traffic in the USA is still attributed to P2P applications [3][3].

---

[1] We present a brief description of these two architectures in Chapter 2.
[2] The terms user, node and peer are used interchangeably throughout this work in reference to a uniquely identifiable autonomous entity (e.g., a single computer) participating in the P2P network.
[3] However, real-time entertainment traffic constitutes the majority of the total traffic nowadays.

The shear size and open access nature of the P2P application environment generate important challenges for the design of reliable and efficient P2P networks. Intermittent node connectivity, also known as churn, is one of them. Not only because nodes' joins and leaves increase the network's overhead, but also because reliable access to resources (e.g., files, in a file-sharing P2P network) can be limited by this intermittent availability pattern. Another key challenge is how to minimize the incidence of non-cooperative behaviors, which can lead to unfairness, performance degradation and limited scalability. A well known example for this class of behavior is *free-riding*, which has been documented as a major problem in many deployed P2P networks [4] [5].

In this dissertation, we undertake the specific problem of content availability in large-scale P2P networks. The following section describes the challenges mentioned above in the context of content availability in P2P networks and outlines our proposed solution. We also present our thesis statement and describe how this dissertation proves it. For an overview of P2P technology and its operation we refer the reader to section 2.1.

1.1 CONTENT AVAILABILITY

Content availability refers to the ability of the system to make content (e.g., files) readily accessible to users. Presently, deployed P2P systems exhibit important content availability performance issues. For example, in BitTorrent 40% of the swarms[4] lack publishers (i.e., seeds[5])

[4] In BitTorrent, a swarm is a set of nodes that cooperatively download a single file by exchanging portions of it among them.
[5] In BitTorrent, a seed is a member of the swarm (i.e., node) that has a complete copy of the file being downloaded.

during more than 50% of their lifetime [6], and in 86% of the cases the participants (i.e., leechers[6]) are unable to reconstruct the original data [7].

P2P has characteristic features that demand different content availability solutions to those applied in other distributed systems. Large-scale, open access, heterogeneous resources and churn are just a few of these features. The case of redundancy is a perfect example. Traditionally, erasure coding is used in distributed systems to prevent data loss due to failures. However, it is generally assumed that performing repairs after such failures is not of major scalability or cost concern. In P2P systems, redundancy still is a plausible solution to protect the system against failures, but repairs have major cost and scalability implications. First, due to the rate of content errors generated in the system. When nodes leave the network or when hard-disks fail, the network needs to be reorganized to compensate for the content lost. Second, because access bandwidth is limited. Nodes commit their access bandwidth to repairs without impacting their performance. Consequently, redundancy for P2P networks must address not only availability as key design principle, but repair bandwidth cost as well. In addition, distributed self-organizing processes and data structures to manage the redundancy are key architectural requirements to device reliable and scalable access to content in P2P networks. In particular, we argue that a hybrid redundancy scheme, combining traditional MDS (Maximum Distance Separable) erasure coding and replication redundancy, can provide content availability reliably and efficiently. Our proposed redundancy scheme, named Proactive Repair (and described in detail in Chapter 4) uses erasure coding to achieve a good storage-availability tradeoff. In addition, our scheme generates replicas for each fragment proactively, so that when a single fragment is lost, it can be repaired using minimum bandwidth.

[6] In BitTorrent, leechers are members of the swarm that have not completed the transfer of the file.

Our proposed solution also includes self-organizing distributed algorithms to monitor the state information of our proposed redundancy scheme. We name redundancy maintenance process to this component. Building the algorithms for the redundancy maintenance process is easy using a Distributed Hash Table (DHT) P2P architecture (described with more detail in Chapter 2), which we assume in our solution. The basic lookup functionality in a DHT can be used to define a single point of contact (i.e., the index node) where nodes can gather information for each file. Nodes storing fragments would announce their availability at this point and nodes requesting a file could retrieve a list of targets from the same location. In addition, the node responsible for this location could evaluate the file availability and determine if a repair is needed. In which case, it would instruct the proper nodes to start a repair.

The correct operation of the components above is conditioned upon the full participation of nodes storing and repairing redundancy. However, to assume full cooperation limits the applicability of any solution. Thus, we embed an incentives-based mechanism into the operation of the system to foster cooperation and regulate fair exchange of resources among nodes, with the purpose of maximizing the feasibility of our solution. Our proposed incentive is a bartering mechanism for content availability (i.e., storage) versus performance (i.e., transmission bandwidth). Nodes receive a level of service proportional to their contribution towards the content availability of the system.

To characterize content availability in P2P networks, we present a multi-level failure model, similar to the one introduced in [8]. The purpose of this model is to describe the possible states of the system, on a per file basis (i.e. file availability), and the type of events that can cause a transition between these states.

The top level represents the ideal state. Whenever a piece of this information is lost, the system enters into the error state. We say that a content error (or simply an error) has occurred. If content errors accumulate without taking any corrective measures, the system will reach the failed state; at which point, the original data can not be retrieved from the system. Thus, in simple terms, the purpose of our research is to develop data structures and algorithms that would allow us to avoid the failed state efficiently.

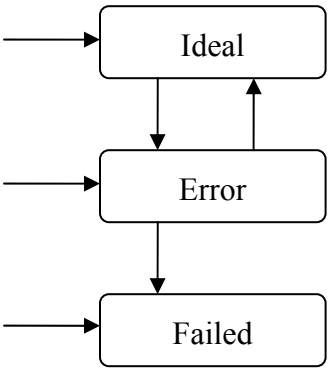


Figure 1. Content Availability System States and Transitions Diagram

In a P2P system, multiple events generate content errors. Some of these errors originate at the network and application layers while others occur at the lower layers. To the best of our knowledge, there is no initiative consolidating content failure models for different layers into a comprehensive content availability model for P2P. In that regard, our research presents a multifaceted content availability framework integrating the effects of content errors at different layers of the application stack and presents a robust and efficient set of mechanisms to alleviate such failures.

There is extensive research on how to recover from content errors at lower layers (e.g., disk failure and nonrecoverable read errors), but these initiatives ignore other sources of content errors prevalent in P2P networks. For instance, in the work presented by Weatherspoon and Kubiatowicz [9] content availability is assumed to be dominated solely by disk failure rates. The research on

how to handle content failures at the network and application layers is also extensive, but weak content availability requirements are usually analyzed [10] and content errors at the lower layers are completely ignored. In contrast, our research goal is to provide strong content availability guarantees while considering a diverse set of content errors. We categorize all lower layer errors as physical errors, and all upper layer errors as logical errors. Next, we briefly describe this taxonomy.

1.1.1 Physical errors

One of the key features of P2P systems is their support for large scale deployments. The shear number of nodes embedded into the system implies that storage resources are typically constituted by a very large pool of independent storage devices. As a result, despite the low probability associated with nonrecoverable disk read errors and disk failures, the large number of components embedded into the system translate this probabilities into an error rate that can impact the reliability of content [11]. Furthermore, the distributed nature of P2P system prevents the use of data protection mechanisms typically used in centralized storage facilities (such as RAID). Thus, the effect of disk failures (and other hardware related errors) is likely to be higher than in other computing architectures. In general, addressing this type of failures can be done easily with traditional reliability measures. For example, given a set of hardware related failure rates –which are usually modeled as i.i.d. events– and a desired availability level, we can achieve a required reliability figure by defining a minimum level of data redundancy to be hardcoded into the system.

1.1.2 Logical errors

In addition to the physical errors inherent in any computing system, P2P networks have many more sources of errors. Thus, reliability in P2P networks is more complex and dynamic than in some other distributed systems. For instance, PlanetLab's nodes [12] have server-like availabilities; thus, the redundancy requirements to achieve a desired level of content availability are much lower than in a traditional P2P system [10]. Furthermore, reliability in a P2P system can not be engineered using the same static reliability metrics used in some other distributed systems. We categorize as logical errors a wide variety of content error events such as routing inconsistencies (when these prevent access to content), nodes departures (i.e., churn) and even content-related user behaviors (e.g., users canceling data transfers or free riding[7]).

Addressing the content availability effects of logical errors in P2P networks is the most challenging task of our research due to the number, complexity and dynamism of the factors involved. Furthermore, the temporal, quantitative and qualitative contributions of individual nodes towards the overall content availability of the network are typically highly heterogeneous. Some users contribute large amounts of resources consistently during long periods of time, while others may contribute only a few or no resources at all to improve content availability. As a result, any content availability mechanism to be used should use a holistic approach to manage this diversity in behaviors. Mechanisms to manage the intermittent connectivity nature of peers should be complemented with mechanisms to compensate the heterogeneity in user-behaviors prevalent in P2P networks.

[7] In the context of this dissertation, free riders are nodes consuming resources without sharing a fair portion of their resources with other nodes in the overlay.

1.2 THESIS STATEMENT

The thesis of this dissertation is that *the combination of erasure coding and replication redundancy constitutes a flexible and cost effective way to provide reliable access to content in P2P overlay networks. Moreover, this hybrid redundancy scheme can be integrated seamlessly with incentive-based mechanisms to support fair allocation of resources and to improve the scalability of the system while recognizing the participants' autonomy and diversity.*

To validate the thesis statement above, we take an approach that combines analytical and experimental methods. The analytical portion shows that our proposed hybrid redundancy mechanism outperforms other redundancy schemes in terms of its repair bandwidth requirements. We also develop distributed self-organizing algorithms to automate the redundancy repair for our proposed scheme. In addition, we augment these algorithms with an incentive-based mechanism to promote cooperation and achieve fair exchange of resources among nodes. Together, these mechanisms constitute a holistic framework to improve content availability in P2P networks. The experimental work presented demonstrates the feasibility of our framework. In addition, it realistically incorporates the complex nature of a dynamic P2P routing architecture into our validation process. Our experimental work is based on a deployed DHT-based P2P application stack, named Bamboo [13].

1.3 RESEARCH OVERVIEW AND CONTRIBUTIONS

Redundancy and economic models for P2P networks are addressed extensively in the research literature [14], [15], [16], [17], [18]. However, to the best of our knowledge, the analysis and

integration of these components has not been proposed earlier in the literature to address the diverse set of content failures present in P2P networks. To do so, this work pursues two research thrusts:

i) Creation of a robust content availability model capturing the heterogeneous and dynamic nature of P2P networks
ii) Development of a multifaceted redundancy-maintenance mechanism to improve content availability in the presence of diverse sources of errors

The aim of the first thrust is to capture the effects of two types of content errors into our analysis: physical errors and logical errors. In addition, we want to capture the diverse habits, capabilities, needs, and interests of individual P2P participants. From this content availability model, it will be possible to portray how these heterogeneous capabilities and attitudes are reflected by the content error events in the network.

The goal of the second research thrust is to create and integrate scalable redundancy and incentive-based mechanisms capable of improving content availability in P2P networks.

To study content availability in P2P networks, we develop analytical models for file availability and redundancy maintenance. We use these models to demonstrate the superior performance of our proposed redundancy scheme against erasure coding and network coding redundancy. In addition, we present a content availability evaluation framework that defines the factors and levels to be analyzed experimentally.

This dissertation proposes the use of redundancy to compensate the negative effects of dynamic node membership and other content error events, together with a utility/cost economic model to achieve fair allocation of resources and self-organization. The redundancy-maintenance method proposed has low maintenance bandwidth, with minimal complexity and superior flexibility properties. In the incentive-based mechanism we propose, we assume that peers transfer

the items they posses upon request. Thus, peers' content availability contribution can be expressed simply as a function of the number of items hosted by a node. Nonetheless, we explore the use of additional qualifiers to improve the fairness of the mechanism, such as size, popularity and the amount of time each item has been shared,. Nodes receive an incentive in the form of performance (i.e., download bandwidth, as in BitTorrent) in exchange for hosting entire or fragments of files to improve the content availability of the network. In addition, the incentives-based mechanism presented provides a mechanism to achieve fair allocation of resources (i.e., each node receives a level of service proportional to its content availability contribution) while preserving node autonomy. The functional components for our system are illustrated in Figure 2.

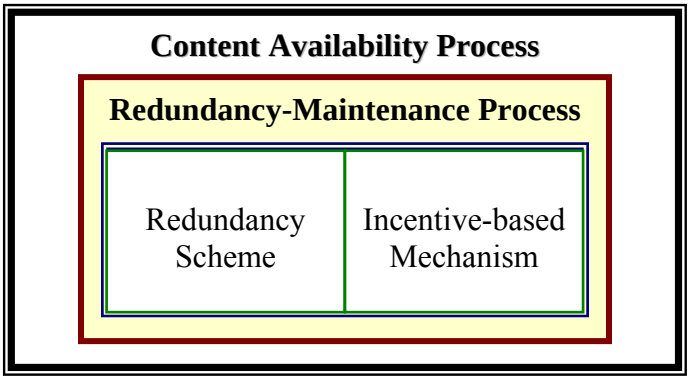


Figure 2. Content Availability System Proposed in this Dissertation

The relevant features of the proposed solution are simplicity, scalability, self-organization and flexibility. Simplicity is achieved by engineering a proactive redundancy-maintenance process that does not require complex data structures or elaborate scheduling algorithms. Scalability is achieved in two ways: by using less maintenance bandwidth than other redundancy schemes, and by implementing the redundancy-maintenance process in a completely distributed fashion. In addition, the redundancy-maintenance mechanism adapts its operation automatically in order to maintain its performance when the network conditions change through time.

The proposed solution is implemented as a DHT-based content availability system. The underlying routing architecture employed is named Bamboo [19] and it has been used earlier to test other distributed applications. Our system is tested using an emulated[8] wide area networking environment, named Modelnet [20], which captures the delay and bandwidth restrictions of P2P networks deployed over the Internet. This system architecture (Modelnet + Bamboo + Redundancy Maintenance Process) is used to conduct extensive simulations to demonstrate the content availability features of the proposed solution.

In summary, the contributions of this dissertation are:

- The creation of a broad content availability framework (Chapter 3.0)
- The definition of a new low-complexity and highly-flexible redundancy scheme that requires small amounts of repair bandwidth (Sections 4.1 and 4.3.2)
- The formulation of an analytical model to determine fragment availability, for code-based redundancy schemes, in P2P networks (Section 4.2.2.2)
- The formulation of analytical models to asses the redundancy repair cost for different code-based redundancy schemes (Section 4.3)
- The formulation of a redundancy-maintenance process (Section 4.4) with incentive-based mechanisms (Section 5.3) that:
    - Uses a proactive redundancy maintenance methodology that consumes the same or less maintenance bandwidth than other methods (Section 4.3.2)
    - Improves the content availability of items independently of their popularity
    - Is completely distributed (Section 4.4.1) and self-organizing (Section 5.2.2)
    - Adapts automatically to the dynamic conditions in the network (Section 4.4.3)
- A prototype implementation of the redundancy-maintenance process with incentive-based mechanisms as proof of concept (Chapter 6.0)

[8] However, we use the term simulation in our experimental work in reference to the entire simulated P2P networking environment..

Figure 3 illustrates how the topics presented in each chapter are integrated to create a content availability system for P2P networks under churn.

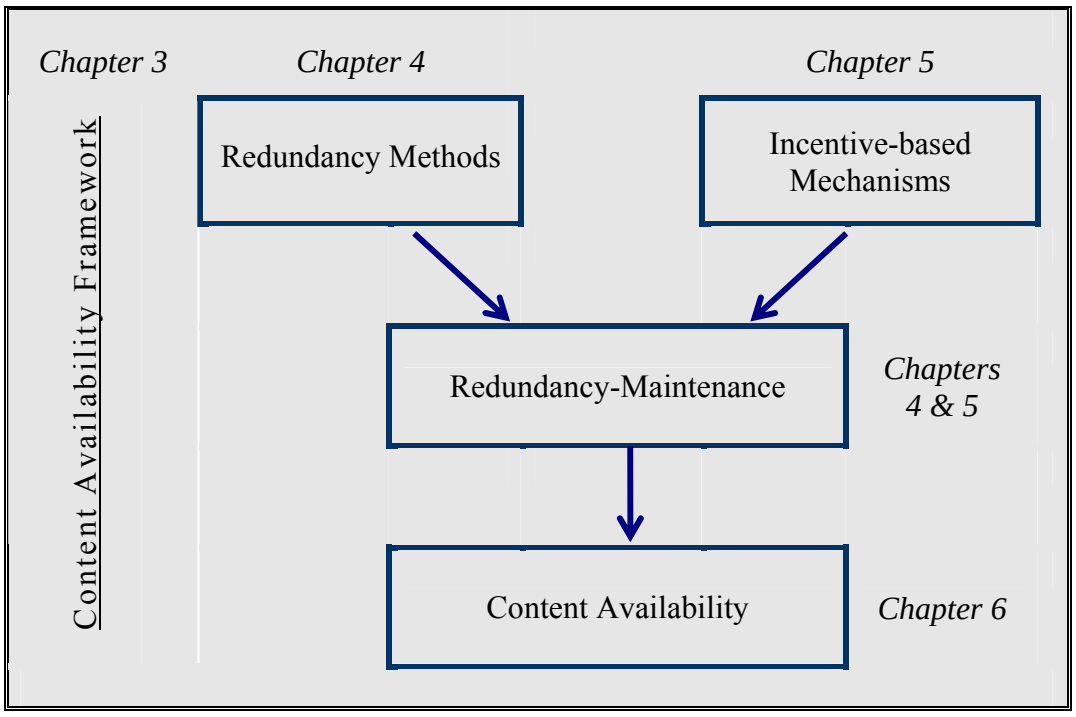


Figure 3. Dissertation Organization

1.4 DISSERTATION ORGANIZATION

This dissertation is organized in seven chapters.

Chapter 2 presents an overview of P2P and the research literature related with our research. As part of the background section on P2P, we include a description of the operation of the DHT routing substrate employed in the implementation of the proposed content availability system. In the literature review section, we describe research initiatives organized in four topics: churn models, redundancy methods, incentives mechanism and content availability.

Chapter 3 presents a framework describing the major factors affecting content availability in a P2P system. In addition, this section presents the levels used for each of these factors in the experimental portion of this work.

Chapter 4 introduces different redundancy schemes that can be used to improve content availability in P2P networks. In this chapter, we assess the effectiveness of our proposed redundancy scheme and compare its performance with other code-based redundancy methods.

Chapter 5 introduces economic models for P2P networks and describes the implementation of an incentive-based mechanism to promote the participation of nodes in a redundancy-maintenance process.

Chapter 6 presents an evaluation of our prototype implementation of a content-availability system that uses our proposed redundancy scheme in conjunction with incentive-based mechanisms to improve content availability.

Finally, Chapter 7 presents the conclusion of this study and possible directions for future work.

2.0 BACKGROUND AND LITERATURE REVIEW

The purpose of this chapter is to present an overview of the features of P2P technology as well as a review of a variety of subjects related with content availability in P2P networks. The rest of this chapter is organized as follows. Section 2.1 presents a description of P2P technology. Section 2.1.4 describes Bamboo [21], which is the DHT routing architecture employed for the construction of the redundancy maintenance system presented in this dissertation. Section 2.1.4 describes the different content ownership modes in P2P networks. Finally, Section 2.3 presents an overview of the research literature related with our research.

2.1 P2P TECHNOLOGY

Peer-to-Peer (P2P) computing or networking is a distributed application architecture where participants self-organize into virtual networks of nodes and logical links on top of an existing network infrastructure, generally the Internet. The participants are autonomous, self-interested nodes that share a portion of their resources to provide the services and content offered by the network.

P2P technology has emerged as a viable solution for the deployment of large-scale distributed applications such as content distribution (with BitTorrent as the most prominent example) and file sharing (with Napster first, and Gnutella, and many others later on). Currently,

deployed P2P systems reach several million users, with several hundred thousand peers[9] connected simultaneously at any time. The number of individuals, groups and organizations that have adopted the P2P computing paradigm is growing steadily, as does the amount of traffic exchanged. In recent years, content distribution sites exhibited tremendous grow; Mininova's[10] site for example, doubled its number of downloads in 2008 to 7 billion in a year [22]. In addition, recent measurement studies indicate that P2P applications make up the majority of the Internet traffic worldwide [2]. The emergence of real-time video traffic in the entertainment industry (Netflix in particular) has surpassed P2P downstream traffic during peak periods in North America [23], but more than half of all upstream traffic is still attributed to P2P applications [3][11].

2.1.1 P2P Taxonomies and Features

P2P computing covers a broad range of systems and applications that can be categorized using multiple taxonomies and characterizing features. For example, Milojivic et al., in [24] present taxonomies for the classification of P2P computing from a system, application, and market perspective; Keong et al., in [25] classify P2P networks as *unstructured* and *structured*, and employ a nine dimension taxonomy to compare them. More recently, Buford and Yu in [26] classify P2P networks in *unstructured* and *structured* categories, and also describe other classes of P2P overlays, such as hierarchical, service, semantic and sensor overlays. This section summarizes the properties of P2P systems and describes the fundamental characteristic of *unstructured* and

[9] The terms user, node and peer are used indistinctively throughout this work in reference to a uniquely identifiable autonomous entity (e.g., a single computer) participating in the overlay network.
[10] Mininova is an indexer site for the popular BitTorrent network.
[11] However, real-time entertainment traffic constitutes the majority of the total traffic nowadays.

structured overlays. For a comprehensive review of different classes of P2P overlays we refer the reader to [26].

Table 1. Peer-to-Peer Systems Properties

Property	Description
Decentralization	The responsibility for the operation of the system is equally distributed among participants. There is no central point of control. However, in many designs this property is relaxed and some nodes assume special roles.
Resource Sharing	The resources required to support the services and content offered by the network are provided by its participants. The nature and amount of resources depends on the application and the system's architecture. As a result of the decentralized and resource sharing properties, nodes in P2P overlay networks (i.e., peers) play dual roles as providers and consumer of resources.
Scalability	The minimum amount of resources that each node needs to commit to the overlay grows less than linear with respect to the network size. In addition, the performance metrics of the system, such as response time, does not degrade significantly with increased system size or load.
Autonomy	Participation in the overlay is voluntary. Every node decides unilaterally the extent of its participation, including the amount of resources committed and when to join and leave the system.
Self-organization	Nodes use local knowledge to make decisions that over time result in a better organization of the system. Nodes continuously adapt to the dynamic networking conditions in the overlay, assuming the responsibility for maintaining their own neighbor relationships despite the continuous arrival and departure of other nodes to and from the overlay.
Fault resiliency	The decentralized overlay architecture precludes the existence of single points of failure. This allows the system to continue its normal operation without significant performance degradation in the presence of node and communication errors. After a failure, the system is capable of reorganizing itself without the intervention of a centralized coordination entity (i.e., self-organization). This property also translates, at some degree, into a capability to avoid censorship and resist other types of attacks.
Cost of ownership	A premise of the P2P paradigm is that the aggregated value of the participants' resources is greater than the sum of the individual resources. Furthermore, given that these resources are contributed voluntarily, the deployment and/or maintenance cost of the system is amortized by its members. As a result, P2P technology constitutes a cost effective alternative for the deployment of network services, features and functionalities that would require modifications in the underlying network infrastructure.

Table 1 describes the fundamental properties applicable to most P2P systems. Depending on specific implementation decisions, different systems favor some properties over others. For

example, Napster used a pure P2P data transfer principle, but its content indexing implementation was completely centralized, thus not P2P.

There are two major P2P overlay architectures, *structured* and *unstructured*. Most P2P systems can be classified into one of these architectures based on the way relationships between nodes are built and maintained.

2.1.1.1 Unstructured P2P Overlays

In *unstructured* P2P overlays, peers join the network without any prior knowledge of the topology and form a random graph in a flat or hierarchical manner that usually exhibits small world phenomena. Nodes rely solely on their adjacent nodes for delivery of messages to other nodes in the overlay. This type of P2P systems usually supports data location using complex queries, but generally there are not guarantees (i.e., data location is a probabilistic process). *Unstructured* overlays can be further classified according to the data location/distribution model used in *centralized*, *distributed* and *hybrid* systems. Figure 4 illustrates this taxonomy.

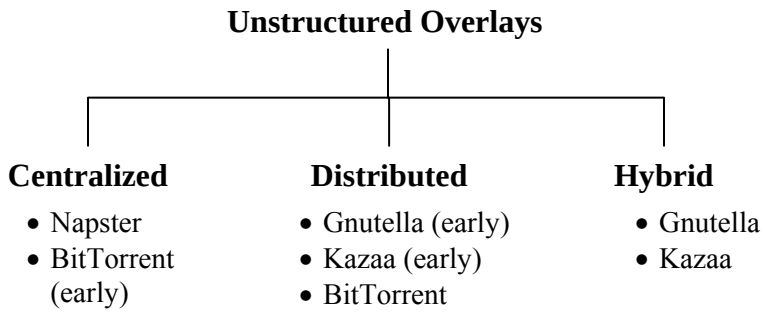


Figure 4. Unstructured P2P Overlays Taxonomy

Napster emerged in 1999 as a file-sharing application based on a centralized indexing mechanism and direct content transfer among participants. The popularity of P2P file-sharing applications rapidly transformed the distribution of Internet's traffic, making up more than half of

it as early as 2006 [26]. The content distribution application BitTorrent (before version 4.2.0) used a similar approach, but today its architecture includes alternative distributed mechanisms for coordinating data upload and download. The early implementations of file sharing applications Gnutella and Kazaa were completely distributed (i.e., nodes were organized in a random flat topology), but scalability problems with the message propagation technique (i.e., flooding) fostered the adoption of hierarchical architectures (i.e., hybrid) and alternative message forwarding mechanism, such as random walks, that have allowed these systems to scale successfully to hundreds of thousands of nodes.

2.1.1.2 Structured P2P Overlays

In s*tructured* P2P overlays, the network topology is formed according to specific criteria and algorithms to achieve robustness and improve performance. *Structured* overlays use a key-based virtual addressing space for node identification and for data placement or location. Nodes cooperate to maintain routing information about how to reach other members of the overlay and support a consistent exact-match data location functionality. Xuemin, et al., in [26] classify *structured* networks according to the number of hops needed to reach other nodes in multi-hop, variable-hop and O(1) hop.

Multi-hop *structured* overlays are further subdivided in logarithmic and constant degree systems. Logarithmic degree systems are subdivided in prefix-based routing and ring geometry. Figure 5 presents this taxonomy together with the name of two sample systems in each category.

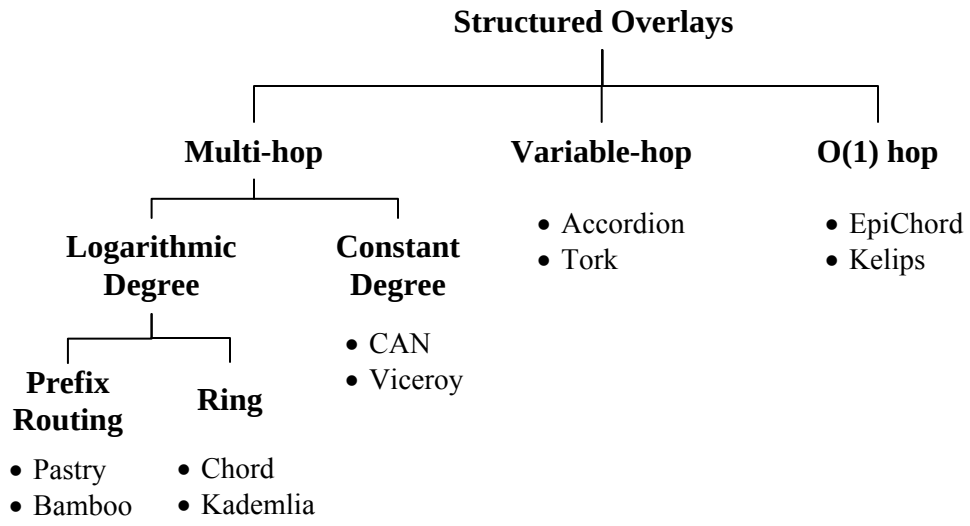


Figure 5. Structured P2P Overlays Taxonomy

Aberey et al., in [27] present a characterization framework for P2P overlays based on six key design aspects:

1. Identifier space
2. Mapping of resources and peers
3. Management of the identifier space
4. Graph embedding
5. Routing strategy
6. Maintenance strategy

This framework is presented in Figure 6 to illustrate the overall construction of *structured* P2P overlays. In *structured* systems, both nodes and resources are mapped into a common identifier space (functions F_P and F_R in the diagram). In the figure, nodes identifiers are represented by circles and resource identifiers are represented by rectangles. The terms nodeId and key are commonly used in reference to the identifiers of nodes and data elements respectively. Keys are associated to the node with a nodeId numerically closer to their own value. In addition, nodes organize themselves to form a geometry defined by the system architecture (e.g., ring, tree, etc.) and continuously maintain this topology to adapt to the dynamic networking conditions in the

overlay. The overlay's structure facilitates data location and provides performance bounds on the number of hops required to reach any node in the overlay, however, it does not support complex queries.

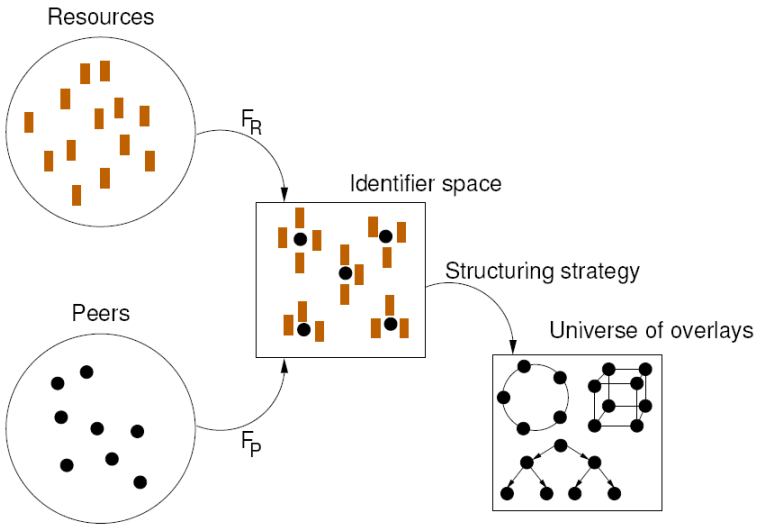


Figure 6. Overlays Network Design Decisions[12]

Most *structured* overlays use a Distributed Hash Table (DHT) to create the system's address space. This is accomplished by employing a globally known hash function to map nodes' IP socket and data objects' identifiers into a key-based address space (generally in the order of 128 bits long), and distributing the responsibility of managing a portion of this address space among the participating nodes.

Examples of structured overlays include mostly academic works, such as Bamboo [21] and Chord [28], but there are also deployed systems like eMule [29]. Most deployed *unstructured* overlays have adopted DHT-based indexing services to improve their data discovery mechanisms. In Section 2.2 we present a more detailed description of Bamboo's DHT implementation, which we use in the experimental portion of this work.

---

[12] Original figure taken from 27. Karl Aberery, et al., *The essence of P2P: A reference architecture for overlay networks*, in *Fifth International Conference on Peer-to-Peer Computing*. 2005: Konstanz, Germany.

2.1.2 P2P Architectures Comparison

During the last decade of research and development, the P2P application ecosystem grew enormously. Multitudes of P2P system architectures were introduced and in many cases, these architectures have evolved into multifaceted complex systems that are difficult to compare due to the diversity of features and functionalities involved. For example, presently there are at least twenty software implementations available for the Gnutella network with no less than a dozen optional features [30]. In addition, the Gnutella specification evolved from a flat topology with a broadcast-based content discovery mechanism into a hierarchical architecture with random walks and hash-based content discovery mechanisms. Thus, the reader should keep in mind that when comparing among different overlay architectures, the literature refers to the properties that characterize the fundamentals of each architecture, and not their current, state of the art implementations. In the paragraphs that follow, we compare several properties of structured overlay architectures versus unstructured overlay architectures and present a summary of our observations in Table 2.

Structured overlays are fully decentralized and most *unstructured* architectures are not. Decentralization is important in terms of fault tolerance and scalability, but presently, this difference does not seriously jeopardized the viability of *unstructured* overlays.

P2P architectures use diverse resource discovery paradigms. *Structured* systems provide an exact-match consistent data location functionality, which is analogous to their routing functionality. *Unstructured* systems on the other hand, use a probabilistic data location service (such as limited scope broadcast and random walks) that supports the usage of keywords and wildcards. In early *unstructured* overlays, successful data discovery was intimately related to the items' popularity, but presently many systems have incorporated hash-based indexing

functionalities in their architectures to provide consistent data location services (independent of the items' popularity).

Table 2. Comparison of P2P Architectures

Features	**Architecture**	
	Structured	***Unstructured***
Sample Systems	Bamboo & Chord	Kazaa & Gnutella
Network topology	Deterministic	Random (two layer) hierarchy
Decentralization	Full	Limited
Scalability	Yes	Yes
Resource Discovery	Distributed w/ exact-match (same as routing)	Limited-broadcast w/ keyword & wildcard support
Performance guarantees	Yes	Limited
Fair allocation of resources	n/a [13]	Yes [14]

Fair allocation of resources and cooperation among nodes is assumed by most P2P architectures. That is, nodes are expected to benefit from the system only as much or in proportion to what they contribute [18]. However, most deployed systems suffer from an uneven distribution of resources. To address this problem, systems incorporate additional mechanisms to promote cooperation and fair exchange of resources among peers. Currently, the inclusion of this type of mechanisms is an open issue and is not fundamentally limited by the architecture being used in the overlay.

[13] Bamboo and Chord do not enforce fair allocation of resources, but other deployed DHT-based systems do.
[14] Kazaa and Gnutella do not enforce fair allocation of resources, but BitTorrent does.

2.1.3 Content Availability

Deployed P2P networks organize large amounts of resources across the Internet. However, the availability of these resources is hindered (among other factors) by the occurrence of *i*) intermittent node participation, which is an unavoidable feature of the open nature of this environment and *ii*) the autonomous operation of participants, expressed as an heterogeneous, and sometimes disproportionate, contribution and consumption of resources among peers.

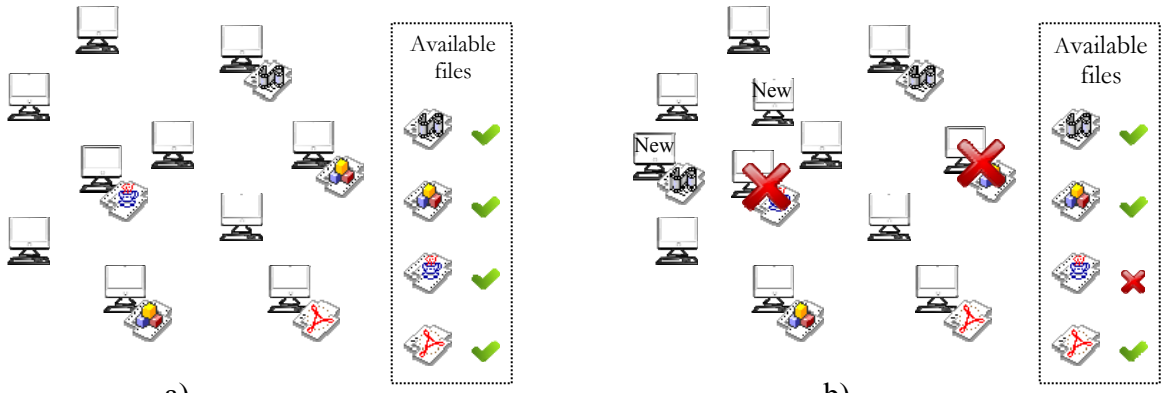
2.1.3.1 Effect of Churn

In most P2P networks a node's connectivity is transient. This phenomenon, called churn, is the main source of dynamism in the network. Understanding the impact of churn on content availability is the keystone to improve content availability in P2P overlay networks.

Participation in P2P networks is open, meaning that any node across the Internet can join the system. Churn is an unavoidable feature of the open nature of this application environment because users decide autonomously when and for how long to join the network, when to leave and whether to return. Thus, P2P communities are a dynamic conglomerate of heterogeneous hosts and resources where the participation of individual hosts is voluntary and transient.

Previous studies have shown that churn in P2P networks is prevalent across different applications and highly heterogeneous [31]. Nodes exhibit session lengths that can vary from a few minutes [32] to several hours or even days. The impact on content availability occurs when users leave the network, taking their content with them. In addition, users returning to the system get to decide unilaterally whether to share the content they previously obtained from the system. As a result, after a node departure, the network needs to reorganize the remaining peers to maintain reliable access to content.

Figure 7 presents a basic scenario that illustrates the effect of churn on content availability. In Figure 7.a every node can access the information stored in the network. There are five nodes sharing four unique data items. In Figure 7.b two of the nodes storing data have left the network and two new ones have joined. At this point, only three of the original data items are still available and the system has no means to recover the lost data.



a) *P2P Network*

b) *Churning P2P Network*

Figure 7. Churn's Effect on Content Availability

Figure 7.b also illustrates that out of the two items lost; only one of them is now inaccessible. This is due to the extra copy stored at another node. This suggests the trivial solution to the problem: generate copies of all data items in every node, but this is certainly inefficient and not scalable.

Instead, a scalable solution to this problem should:

i) Generate only enough redundancy in the system to handle the departure of nodes between maintenance epochs and

ii) Reorganize the remaining nodes to regenerate any lost data before items become inaccessible due to additional churn events

While this solution is expressed in simplistic terms, its realization involves major complexity. For example, how do we determine how much is *enough* redundancy? And how do we reorganize the remaining nodes?

Redundancy is a necessary component to achieve content availability in P2P networks [10]. However, the selection of a redundancy method cannot be performed using the same cost-performance tradeoffs as in other distributed systems. Reliability needs to be continuously repaired. That is, any data loss due to churn (and other logical or hardware related failures) needs to be regenerated before the content availability degradation becomes irreversible. Performing this redundancy-maintenance efficiently has been described as the key limiting factor for the scalability of distributed storage P2P applications [10]. In particular, the amount of bandwidth needed to maintain the reliability of a redundancy method is a fundamental scalability concern in P2P environments.

2.1.3.2 Effect of User Behavior

Participants in P2P networks are usually expected to voluntarily share resources towards a common goal[15]. Nevertheless, peers are autonomous and decide the extent of their participation in the network unilaterally. In the absence of proper incentives, users acting rationally in their own self-interest do not commit enough resources towards the network's overall objective. Improving content availability in P2P networks requires mechanisms capable of dynamically managing not only transient and heterogeneous node connectivity, but also diverse individual tradeoffs between the peers' goals and the network's content availability objective.

[15] In that regard, P2P file-sharing networks can be modeled using economic models as an instance of private provisioning of a public good.

This dissertation uses economic models to address the user behavior aspects of the content availability in P2P networks. In these models, the term participant is equivalent to the terms users or node. Thus, the terms user, peer, node and participant are used indistinctively throughout this work in reference to the computers that form the overlay network.

Economic and networking processes have multiple similarities: complexity, autonomous self-interested participants and dynamic time-varying conditions, are just a few of them. Economic concepts and models have been used effectively in the study and construction of distributed systems, including P2P. In the economic incentives model in particular, the basic principle is that for an efficient and fair allocation of resources, there must be incentives for providers to share their resources as well as encouragement for consumers to maximize the utility of the received resources [33]. In P2P networks, nodes play a dual role as both consumer and providers of resources and consequently, the objective of incentive-based mechanisms is to allow nodes to reach a balance between their provider and consumer roles.

2.1.4 Content Ownership in P2P Networks

In structured P2P networks, nodes and content are both mapped deterministically to IDs in the same key-naming space. According to their IDs, each participating node becomes responsible for a section of the key-naming space (namely, Distributed Hash Table). The network supports three basic content operations: *query*, *indexing* and *storage*. In a DHT-based system, queries are functionally the same as routing. Therefore, a query operation consists in routing for a given key and finding the node responsible for that portion of the key-naming space. A node managing the section of the key-naming space including the ID of content c_i is called the *root* or *home* node for content c_i. *Indexing* is a redirection service that links content-IDs with the nodes storing the

item(s), namely *holder* nodes. The operation of *indexing* can take two forms: register and fetch. The register operation is when a node adds or updates its information to the index and fetch is when nodes request the list of nodes storing a given item. The *indexing* operation consists of several steps. First, nodes *query* for the root-node of the file they are interested in. Second, nodes contact that node directly to either register or fetch information. Finally, a *storage* operation is simply a request to download or upload data. For example, if node A sends a download request to node B, then A is consuming resources from B, and if the request is an upload, then A is publishing content to B.

Nodes can play four alternative roles in the network, namely *publisher*, *index*, *holder* and *requestor*. The *publisher* node is the original creator of a data item. An i*ndex* node is the *root* node for a specific item. A *holder*, is a node storing a partial or complete copy of an item. Finally, *requestor* nodes represent users downloading information from other nodes.

Content ownership can take two forms; nodes keep copies of the files for which they are *root* nodes (assuming temporal ownership of the item), or they simply keep pointers to the actual location of the file, namely the *publisher* node. We will refer to these two variants as *root ownership* and *node ownership* respectively.

For retrieval of content, the *node ownership* model implies that *publisher* nodes periodically contact their *root* nodes to refresh their indexing information, and *requestors* perform a two-stage process to retrieve content from the overlay. First, they ask the *root* node for a list of candidates, and second, they attempt to contact one or more of these candidates directly to retrieve the item. Successful content retrieval is a dynamic process that involves the *publisher* node, the item's *root node* and the *requestor* node. For a successful retrieval of content two conditions must be met *i*) the item's indexing information (at the *root* node) is updated and *ii*) the remaining

session length of the *publisher* and *requestor* nodes are long enough to complete the transfer. Both of these conditions are influenced by churn. The first condition requires *publisher* nodes to continuously monitor the availability of their *root* nodes and find new ones as needed.

In the *root ownership* case, items are first uploaded to their *root nodes*. Still, *publisher* nodes might be required to monitor their root nodes to guarantee the availability of their contents. However, *holder* nodes do not interact with peers requesting an item. Instead, items are downloaded directly from the *root* nodes. This mechanism decouples, to some extent, the availability of items with the liveliness of their *publisher*(s), since the *publisher* node can be offline while the *root* node is still uploading an item to another peer. On the other hand, if the *root* node leaves the network, the item will need to be transferred once more into a new *root* node*;* regardless of any present or future demand for the item. This model is assumed in distributed storage applications such as CFS [34] and OceanStore [35]. The problem with this approach is that in order to minimize the maintenance overhead, the participant nodes must have high availability, which is not the case for open P2P application environments.

2.2 BAMBOO

Bamboo [21] is a reengineered version of Pastry [36], a DHT-based P2P routing architecture that uses a circular identifier space, with IDs 160 bits long organized as a sequence of digits base 2^b. In Bamboo, nodes maintain two sets of neighbors, the *left set* and the *routing table*, illustrated in Figure 8 by dashed and solid arrows respectively. The *leaf set* contains the *k* nodes preceding and the *k* nodes following the current node in the circular identifier space. The routing table is a set of nodes organized in matrix form. All the node identifiers (nodeIDs) in row *l* coincide in *l* digits with

the current nodeId and the value of the next digit determines their column in the routing table. That is, a node in row *l* and column *i* shares *l* digits with the current nodeId and its *l*+1 digit has a value equal to *i*.

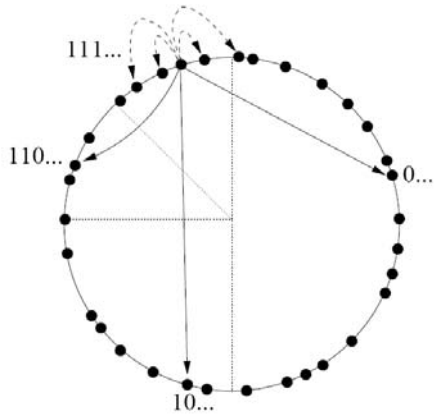


Figure 8. Neighbors in Bamboo.

The basic functionality in Bamboo (and other DHT-based routing architectures) is a distributed (*key*, *value*) lookup service. Given a target key D, nodes follow the following algorithm:

1) Check *leaf set*. If D lies within its *leaf set*, then it forwards the query to the nodeId numerically closest to D. If that node is the local node, routing terminates. If not, continue with step 2.
2) Route message. The node computes the longest matching prefix between D and its own nodeId, denoted by L. The value of the first digit in D different from the local nodeId can be denoted as D(L+1). If the node has a non-empty routing entry at row L, column D(L+1) it forwards the request to that node; otherwise it goes to step 3.
3) Forward to leaf. The message if forwarded to the member in the leaf set numerically closest to D.

The process above is performed by every node that the request is forwarded to. When the final destination is reached (step 1) a message is sent back to the originating node with the nodeId and the network address of the destination. In DHT-based systems, the processes of routing and

data location are functionally the same. That is, the lookup service we just described is used for both routing and resource discovery.

Figure 9 illustrates the basic procedure employed to map nodes and content into the overlay's identifier space. For nodes, their IP socket (or other form of unique identification) is used as input of the hash function to obtain a key value, namely the nodeId, which is used for routing and to claim responsibility for a portion of the identifier space. For data items, the identifier (e.g., file name) is used as input of the hash function, and the key value obtained determines which node is responsible for managing the lookup service for that item. The node with the numerically closest nodeId to the object's key value is called the root node or home node for the object.

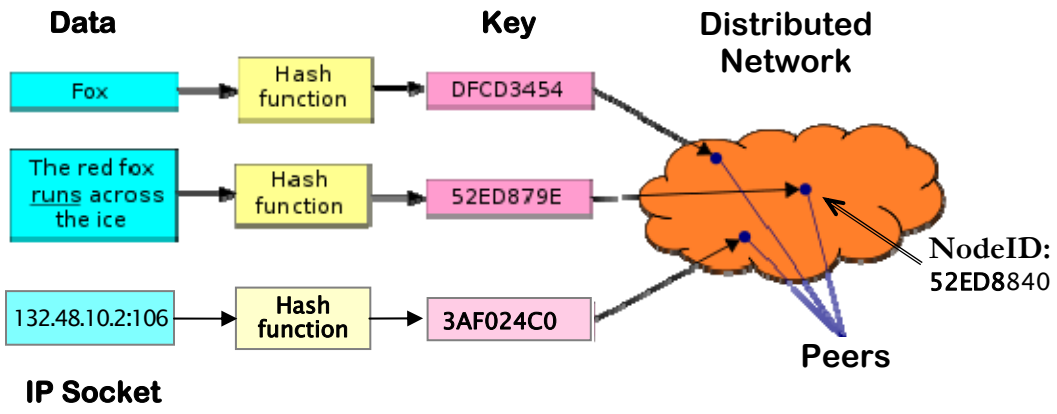


Figure 9. Distributed Hash Table

The scalability of DHT-based P2P system is derived from the structure of their routing tables. The distance between the local nodeId and the target key is reduced logarithmically at every step throughout the lookup process. Thus, DHT-based networks are able to provide a deterministic location service in O(log *N*) steps, where *N* is the number of nodes in the network. Other DHT-based systems have similar scalability metrics and also share other functional and structural similarities with Bamboo, but their description is beyond the scope of this work and comparative descriptions are available elsewhere [25, 37].

The resiliency of DHT-based architectures is derived from the routing geometry embedded in their neighbor set. That is, the pattern of neighbor links across the overlay, independently of the routing algorithms or state management algorithms used [38]. The work of Gummadi, et. al. presented in [38] describes that the addition of sequential neighbors significantly increases the *static resiliency*[16] of a system, but increases its latency. In addition, the flexibility in neighbor selection (FNS[17]) is better than using flexible route selection (FRS[18]) to improve the performance of the system (i.e., latency). In that regard, Bamboo's *leaf set* provides good static resiliency, and *proximity neighbor selection* is used to boost performance when more than one node can be used to fill in a routing table entry.

The creators of Bamboo summarize the factors allowing the system to handle high levels of churn efficiently in three functionalities: periodic recovery, adaptive timeouts and proximity neighbor selection. After a node failure, the remaining nodes need to reorganize their *routing table* and possible their *leaf set*. To avoid overwhelming the system with route repair messages, Bamboo opts for a periodic route maintenance policy that bounds the amount of bandwidth consumed. In addition, Bamboo maintains different timers for each of its neighbors (i.e., timeouts), which allow it to discern judiciously between node failures and network congestion or processor load. Lastly, Bamboo uses a two steps process to fill in its routing table entries. First, it performs a lookup for a random identifier with a required prefix *p* corresponding to a hole in its routing table and uses the returned value to fill it in. Secondly, nodes query their neighbors' routing tables to find alternative entries providing better latencies. The objective of the first process, called *global tuning*, is correctness, while the goal of the second one is performance. The system prioritizes correctness

[16] Static resiliency describes the capacity of the network to route messages after failures and before route repairs are performed. This is a fundamental property in dealing with churn.
[17] Flexible neighbor selection is a measure of the level of flexibility nodes have to fill in entries in their routing tables.
[18] Flexible route selection is a measure of the level of flexibility of the routing algorithm to select the next hop.

and improves its performance opportunistically when the amount of traffic being managed by the node allows it.

The results presented in Figure 10 are taken from Bamboo's technical report [13]. In these graphs, the capacity of Bamboo to handle extremely high levels of churn (i.e., small median session times) can be easily appreciated. Even when the median session length falls below eight minutes, Bamboo manages to perform most requested lookups with a mean latency that outperforms other systems. Nonetheless, we showed in [39] that if the maintenance intervals of Chord (or other DHT-based system) are tune up according to the expected level of churn, the performance level obtained is comparable to Bamboo.

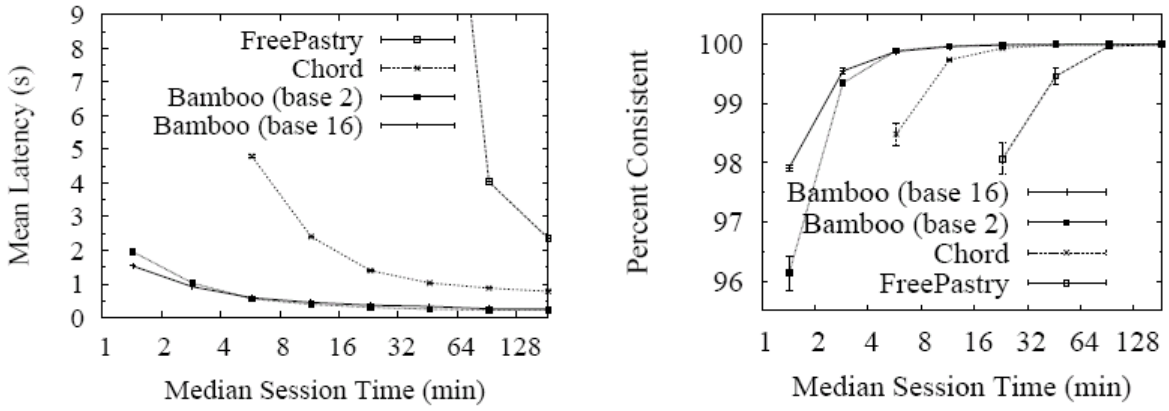


Figure 10. Bamboo's Performance Under Churn

Bamboo was selected over other DHT-based architectures to analyze the effects of churn on content availability because of its software architecture and its probed performance in churning environments. Bamboo is a mature open source DHT implementation written in java using an event-driven single-thread programming style. Thus, we can obtain accurate measurements of a deployed DHT[19] rather than the simplified DHT implementations available in P2P simulators [40]. In addition, there is a reasonable amount of documentation available to guide our

[19] Bamboo was offered as a public DHT service in PlanetLab during 2005-2009.

development efforts [19] without having to deal with detailed implementation issues related with the operation of a DHT application substrate.

2.3 LITERATURE REVIEW

2.3.1 Churn Models

The continuous and unsynchronized arrival and departure of nodes to and from the overlay network is called churn. When nodes leave the network, the content they were contributing also disappears. Thus, understanding the content availability problem space requires analyzing the fundamental properties of churn in P2P networks. Some research initiatives have tackled this subject using analytical models, and some others using empirical studies.

Yao, et al. in [41] highlight that the heterogeneity of lifetimes and off-line intervals is a fundamental property of P2P networks. The authors develop a model for heterogeneous user churn, from which they derive multiple closed form expressions that characterize the properties of nodes in unstructured P2P networks, including their residual lifetime distribution and isolation probability. In our work, we use a variant of this model to recreate the churn behavior of existing P2P networks. Their work is extended in [42] to accommodate non-stationary arrivals of nodes, and create a sampling technique to measure session lengths in unstructured P2P systems. These papers provide insightful findings with regard to the connectivity and availability properties of nodes in P2P networks, but their scope does not contemplate the implications of churn at the content level.

There is a long list of measurement-based research initiatives using probabilistic models to characterize the distribution of session lengths in deployed P2P networks. These characterizations are important to understand the properties of these systems and to synthesize churn events for simulation studies. Pareto [43, 44] and Weibull [45, 46] are the most commonly reported distributions in the research literature, but the results presented in these studies do not include node level characterizations. Yao, et al. in [41] explain that a perfectly shaped Pareto or any other distribution can be the product of a mix of multiple independent exponential distributions. Thus, it is not possible to conclude the real distribution of session lengths of individual nodes by just observing their aggregated behavior.

Luo, et al. in [47] present an alternative view for the characterization and synthesis of churn. Instead of analyzing the network as a single black-box, the authors divide the user population in multiple geographical regions and assign different user behaviors to them –as in our work presented in [48].– This model is used to generate the cyclical node membership exhibited in deployed P2P systems. The authors also provide a set of MatLab tools that automate the generation of churn logs for simulation studies.

Presently, the research community has not yet agreed in a general model for the distribution of session lengths of individual nodes or their aggregate distribution. As an alternative, Fernández-Casado, et al. in [49] developed a tool integrating several competing models to build a general purpose churn generator capable to generate complex churn logs for simulations. The availability of this type of tools facilitates the comparative analysis on the performance of P2P systems using different churn models. Nonetheless, most of the recent research literature in P2P networks uses public network traces [50, 51] to recreate the node dynamics of deployed systems. In our case,

these measurements lack the appropriate granularity and flexibility for analysis at the content level. Furthermore, some of them could present significant measurement bias [52].

In summary, there has been much progress on modeling and characterizing churn in P2P networks. However, the shear size and heterogeneity in this type of networks make it quite difficult to obtain definite fine grained measurements and models. Our research combines the results presented in diverse measurement studies with analytical models to build a flexible churn framework to analyze the fundamental content availability properties of P2P networks and possible methods to improve it.

2.3.2 Redundancy Methods

Redundancy is a mechanism commonly used to improve the reliability of systems by provisioning excess resources. For distributed data storage, there are multiple methodologies to achieve this reliability. Each of these methods uses different tradeoffs between data reliability and other system properties such as storage space or maintenance bandwidth. The research literature on redundancy in P2P environments can be categorized into three categories: *i*) studies matching a given networking condition (i.e., average node availability) with a redundancy method *ii*) studies defining the optimal parameter settings for a redundancy method given a set of requirements (e.g., required file availability and average node availability) and *iii*) studies defining new redundancy data structures. In addition to these categories, it is important to highlight that a major concern of using redundancy in P2P environments is the amount of bandwidth required to regenerate any data loss due to node departures. This is regularly referred as the *repair problem* [10], [53]. This concern generates a new cost vs performance space for the design of redundancy methods. The

fundamental constraint for this problem is that in P2P networks access bandwidth is scarcer and more expensive than storage space [10].

With respect to the first category, Rodriguez and Liskov in [10] argue that for system with high node availability (such as PlanetLab) replication redundancy should be preferred and furthermore, for other system settings the storage overhead savings of erasure coding might not be worth the associated cost, due to the added system complexity. Lin, et al, in [54] present an analytical expression in terms of storage overhead (S) and node availability (a) to determine which redundancy method provides better file availability. If $S*a<1$, replication performs better and for $S*a>1$ erasure coding performs best.

With respect to the second category, If the system needs to be engineered to achieve a required level of file availability (i.e. 0.99), Rodriguez and Liskov in [10] or Bhagwan et al. in [55] present analytical expression to obtain the optimal storage overhead for both erasure coding and replication. This optimal value is such that the product $S*a$ is always greater than one; thus, according to the arguments presented by Lin, et al, in [54], erasure coding redundancy is always preferred. In [56], Dimakis, et.al., contrast the redundancy repair cost of erasure coding, with two variants of network coding. Their results indicate that it is possible to construct coding methods with significant repair bandwidth savings over erasure coding over a wide range of target file availabilities. However, their results also indicate that as the network becomes unstable (i.e., lower average node availability) the performance of network coding can become inferior to the performance of erasure coding.

With respect to the third research category, that is, the introduction of new redundancy methods, the research literature is abundant. A recent survey of the field of network coding by Dimakis, et al. in [53] present recent advancements in network coding that reduce the redundancy

repair problem by orders of magnitude compared with standard erasure codes. In addition, this paper describes the different repair modalities of erasure coding, highlighting that minimum bandwidth with exact-repairs is the best suited network coding modality for distributed storage applications. In that regard, Rashmi, et al. in [57] present a minimum bandwidth exact-repair (MBR) explicit construction code for any combination of the parameters (m, k, d) where m is the total number of nodes, k is the number of blocks needed for the file reconstruction and d is the number of nodes required for the reconstruction of lost fragments. This is the first explicit code construction that allows the selection of the number of nodes (m) independently of other system parameters. Dominuco and Biersack in [58] present an alternative construction of erasure codes, called Hierarchical Codes. The authors argue that cost is improved because the average number of transfers needed is lower than in traditional erasure coding (despite needing a higher number of repairs). However, the authors account the number of transfers as cost rather than the product of the number of repairs times the average number of blocks transferred (i.e., bytes transmitted). For the mechanism presented by the authors, the task of choosing which blocks should be downloaded to perform a repair has been reduced compared to the cost of traditional erasure coding, but it is still a non trivial problem. Williams, et al. in [59] evaluate different redundancy techniques for P2P storage. They advise the use of hybrid schemes combining replication and either erasure coding or bucketing (data bundling) as a reasonable compromise between maintenance cost and availability. The authors conclude that replication should be used to simplify data access (most of the time) and erasure codes should be employed to achieve the last nines in the desired availability level. In that regard, Wu, et al. in [60] and Xu, et al. in [61] propose hybrid redundancy mechanisms that combine erasure coding and replication. In the work of Wu, et al. [16, 60] content resides on the user-nodes. These nodes regularly contact a set of M Indexer nodes to register their contents

(whole-file and fragments) and based on this information, Indexer nodes decide when to send redundancy repair instructions back to the user-nodes. Whole-file replication is assumed as a by-product of user activity. Assuming the existence of at least one whole-file replica available, the system only employs erasure coding to reach a target file availability level for those items with insufficient replicas available. In the work of Xu, et al. [61] on the other hand, the location of content is determined by a hash function. Both nodes and content are mapped into a virtual identification space in which several physical nodes share the same virtual ID and are responsible for maintaining a target number of replicas for each object. One shortcoming of all the initiatives mentioned above is the assumption that all nodes cooperate fully in the redundancy maintenance process when needed. In our research, the structure of our redundancy method is different. We use erasure coding as the foundation to achieve reliability, and replication as a mean to minimize (and simplify) the redundancy repair problem. Furthermore, we integrate the redundancy maintenance problem with economic models to overcome the problems of cooperation and fairness in the context of distribution of content in P2P networks.

2.3.3 Incentives and Content Availability

The next bodies of research related with our work are the use of incentives mechanism in P2P networks and the focus of our research, content availability itself. The research literature on economic-based mechanism for P2P systems is vast. The issues commonly addressed are fair allocation of resources and prevention of free-riding [62, 63], but content availability is rarely mentioned as a desirable property or by-product of these mechanisms. Furthermore, the resource most commonly managed by these mechanisms is bandwidth, which is a short-lived property of the system [64] and does not translate directly into our objective of improving content availability.

Geels and Kubiatowicz [14] argue that solutions for large-scale replica management should be based on economic models and they outline the benefits of adopting this approach. They introduce the term Replica Management Economy (RME) to describe this type of systems and highlight that automatic resource management, scalability and guarantees through mechanism design are the key advantages of using economic models to deal with this problem. The authors state that RME allow the level of node autonomy that is necessary in a heterogeneous environment like the Internet; regulating the interactions between nodes while fostering cooperation across domains. The authors explain that one of the directions of future research in this field is the design of utility functions to rate the worthiness of alternative actions by a RME player. Our research advances on this path.

To the best of our knowledge, the only two research initiatives targeting content availability in P2P networks specifically are incentive-based mechanisms. Antoniadis, et al. in [18] present an incentive mechanism to control the minimum amount of time that nodes should participate in the system, as well as the minimum number of files that they should share throughout that time. Bai, et al. in [33] on the other hand, analyze the use of bundling[20] to improve the availability of contents (in BitTorrent in particular) improving even the download time experienced by peers when publishers exhibit high unavailability. Our research differs from these works in three aspects: *i*) we want to investigate the use of incentive-based mechanisms that would not require nodes to change their churn behavior, *ii*) we want to explore the system design tradeoffs in the context of DHT-based P2P systems and *iii*) our system does not differentiate between different types of content; thus, it foster content diversity (regardless of popularity), minimize node accountability and preserves node autonomy. That is, the system allows nodes to

[20] Bundling consist on handling more than one object together to be transferred as a unit.

decide unilaterally how much information they want to store; in the understanding that their performance is a function of their contribution.

An additional aspect related with the construction of incentives mechanisms is the definition of redundancy maintenance strategies. In that regard, Data and Aberer in [65] employ a Markov model to analyze the performance of different redundancy maintenance strategies in P2P networks. They determine that a randomized lazy redundancy maintenance mechanism offers significant advantages over existing deterministic and procrastination mechanisms. Yet, they assume full cooperation among nodes in their analysis. Sit, et al. in [66] present a proactive replication system that is capable to maintain high content availability using only several kilobytes per second of bandwidth. The authors argue that "creating redundancy constantly at a limited rate is simple, flexible and effective approach to maintain data durability". However, their evaluation is based on the availability of server-like distributed systems (PlanetLab) and full cooperation among peers. In contrast, our work uses randomized redundancy maintenance strategies integrated with incentives mechanisms. In addition, the structure of our redundancy method can be understood as a proactive redundancy repair strategy, which in the long-run minimizes the system's repair bandwidth.

3.0 CONTENT AVAILABILITY FRAMEWORK

In this chapter we present a framework for studying content availability in P2P networks. The construction of this framework pursues two fundamental goals: First, to guide our analysis by organizing the diverse factors that can determine the performance of a P2P network and second, to describe the fundamental properties and models that characterize the architecture and operation of a P2P network. The elements portrayed into each component of this framework determine the settings for the various setups in the experimental portion of this work.

The layout of this chapter is as follows. Section 3.1 describes the overall structure of our proposed framework. Section 3.2 describes the models embedded into each of its components, and Section 3.3 presents the parameters to be used in each component during the experimental portion of this study and describes the rationality for the settings selected.

3.1 FRAMEWORK STRUCTURE

Our content availability framework has six components, named *User Behavior*, *Node Availability*, *Network Structure, Content, Performance Metrics* and *Incentives and Redundancy*. Figure 11 depicts our framework.

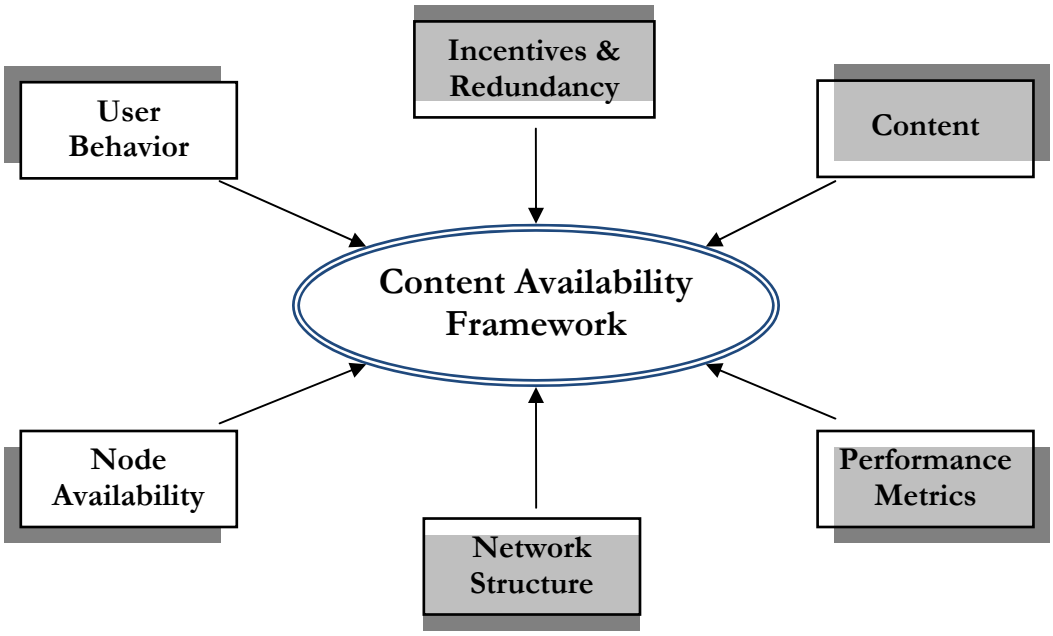


Figure 11. Content Availability Framework

In conjunction, the six components of our framework constitute a complete characterization of a P2P system, from a content availability perspective. The properties and models included in our framework define not only the fundamental characteristics of the network (e.g., routing architecture) but also the performance metrics to be used as a measure of the success or failure of the system.

The first component in our framework is called *User Behavior*. It models the human attitudes that affect the performance of P2P systems. The second component, *Node Availability,* models the intermittent connectivity of individual nodes. That is, this component models when nodes are online and offline. The third component, *Network Structure*, models the elements related to inter-node communication. Consequently, the network routing architecture, the underlying Internet topology and other network-related metrics fall within this component. The fourth component, *Content*, characterizes the properties of the content being offered in the P2P network. The fifth component, *Incentives and Redundancy*, characterizes and models the two fundamental

mechanisms we propose as solution to improve content availability in P2P networks. Finally, the sixth component, named *Performance Metrics*, describes the response variables used to measure the performance of the system and the effectiveness of the mechanisms we propose. The following sections will further describe the elements incorporated into each component.

3.2 FRAMEWORK MODELS

3.2.1 User Behavior Component

The *User Behavior* component models the fundamental user attitudes toward P2P networking that could affect the performance of a P2P system. It has been widely documented that P2P networks are highly heterogeneous [67]. For instance, some users share lots of resources while others share little to nothing at all [68]. In this work, we recreate the heterogeneous nature of users behaviors in P2P networks using a two class user profiling technique. We name these two classes *Benefactors* (*Be*) and *Peers* (*Pe*). Figure 12 illustrates our user profiles model and the properties that characterize each of the two profiles we use in our framework.

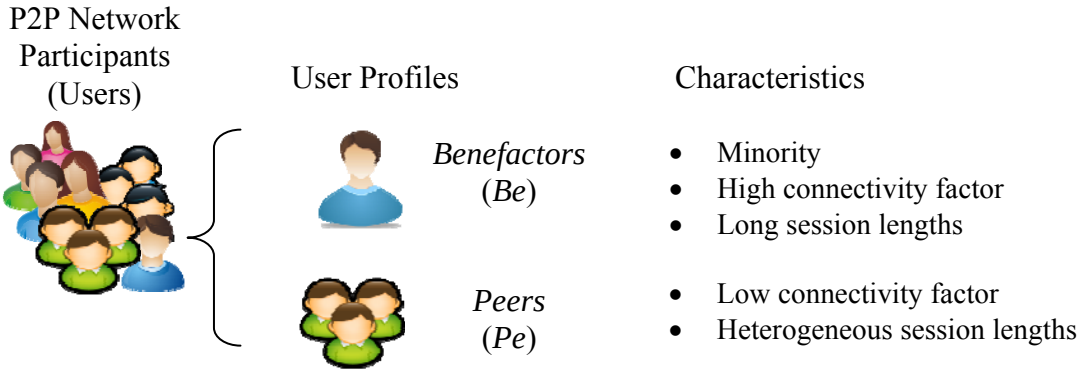


Figure 12. User Behavior Model: User Profiles

Given a set of nodes, a small fraction of them is assigned at random to the *Benefactors* user class and the rest are assigned to the *Peers* user class. Both profiles are characterized using two fundamental properties: *connectivity factor* and a probability density function for the average session length of the nodes in each class. We define the term *connectivity factor* as the probability of finding a node online during its lifetime. The average session length is defined using a set of probability density functions to be described in the next component, node availability.

With the *Benefactors* user class, we want to portrait a set of users that contributes resources generously and consistently to the network. Thus, we assume that *Benefactors* (*Be*) are users who stay connected to the overlay network during long periods of time and exhibit a high *connectivity factor*. In our analysis we want to investigate if the stability of the overlay network depends on the contributions of the *Benefactors* user class. That is, we want to analyze if the portion of the network population within this class can play a crucial role in the performance and the type of applications that can be deployed in P2P networks.

We use the *Peers* (*Pr*) user class to complement the *Benefactors* user class recreating the heterogeneous node participation pattern reported in deployed P2P networks. To accomplish this goal, we adapted the heterogeneous user churn model presented by Yao, et. al. in [41]. In their model, both the online and offline intervals are selected independently using two probability density functions (pdf). In our model, we use a single pdf to define the online interval and the offline interval is obtained using the *connectivity factor* parameter.

3.2.2 Node Availability Component

The *Node Availability* component captures the churn behavior of individual nodes. That is, the rates at which nodes join and leave the overlay network during their lifetime. An On/OFF state

model is used to characterize this behavior. Figure 13 presents this model. The "ON" state represents the time interval during which nodes are active members of the overlay network. During this time, nodes generate lookup traffic and attempt to download items from other nodes. The "OFF" state represents the time elapsed offline until the node rejoins the network.

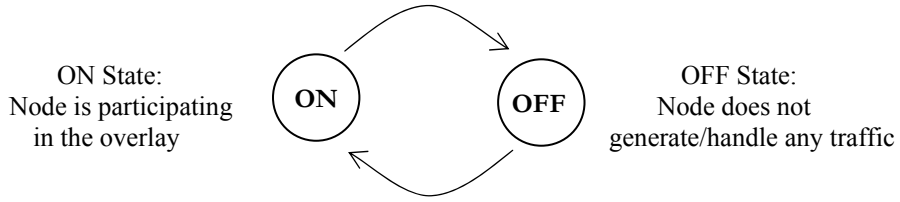


Figure 13. Node Churn Model

The term node availability (*a*) is used in the research literature to characterize the probability of nodes being active. To avoid confusion with other availability metrics that we use throughout this dissertation, the term *connectivity factor* (*Cf*) is used instead. For the ON/OFF model presented above, connectivity factor is measured as the quotient of the average ON interval over the sum of the average ON and OFF intervals of a node. Throughout this work the average online interval is called average session length. Within the *Node Availability* component $\overline{ON}$ represents the average session length, and $\overline{OFF}$ represents the average offline interval. Thus, the *connectivity factor* can be expressed as follows:

$$Cf = \frac{\overline{ON}}{\overline{ON} + \overline{OFF}} \tag{3.1}$$

In our description of the User Behavior component, we mention that $\overline{ON}$ and *Cf* are the characterizing factors for each user class. Consequently, rather than using equation (3.1), as presented above, we employ the following variant in our model:

$$\overline{OFF} = \overline{ON} \cdot \left(1 - \frac{1}{Cf}\right) \tag{3.2}$$

3.2.3 Network Structure Component

The *Network Structure* component models the way nodes are organized and how they interact with each other to maintain the overlay. The fundamental elements that can determine the behavior of the system under churn include the underlying network topology, the network size (N) and the P2P network architecture (i.e., routing mechanism). For this component, we assume that the properties of the underlying physical network and the overlay network topology can be modeled independently.

For the properties of the physical network, we assume a WAN setting and we employ existing tools to define the characteristic bandwidth and delay properties of the network. In particular, we employ Inet-3.0 [69] to generate our model of the physical network.

In our model, Network size (N) represents the total number of nodes participating in the overlay network and the term overlay network size (N_O), is used to denote the average number of simultaneously active nodes in the network. These two elements are related by a constant, called *churn factor* (Ch). That is, $N_O = Ch \cdot N$.

The structure of our framework does not preclude the analysis of multiple P2P network architectures, but the scope of our current work is limited to structured systems. Thus, the architecture of the network is assumed to be DHT-based.

3.2.4 Content Component

The *Content* component models the data items stored in the overlay network and the exchanged patterns of these items among nodes. For our characterization of content we categorize factors into subcomponents named *offer*, *demand, ownership model* and *initialization*. In addition, this

component describes the different roles that nodes can play with respect to content. The main factors included in this component are presented in Figure 14.

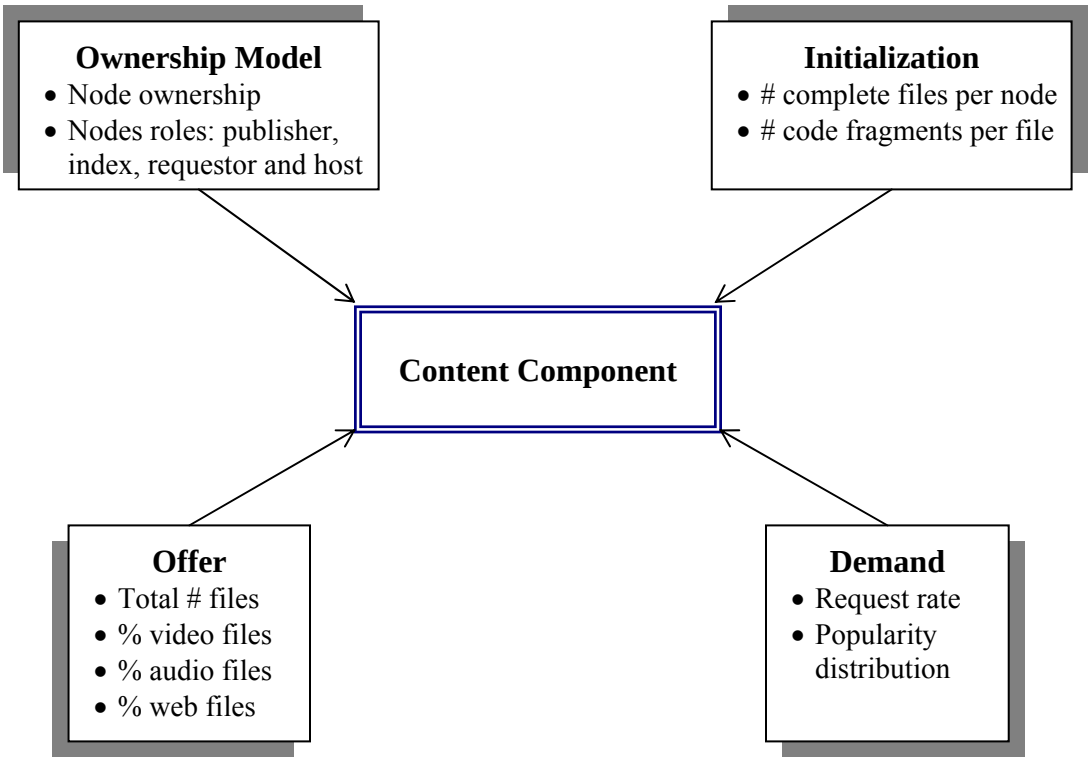


Figure 14. Content Component Factors

The factors included in the *offer* subcomponent are quantitative in nature. We use the term content space in reference to the total number of files (*M*) to be assigned to the nodes participating in the network. These files are categorized by type into video, audio and web-like items and initialized in size according to results presented in the research literature [70]. This type/size breakdown is expressed by the tuple <*A,V,W*> where *A* is the percentage of audio files, *V* is the percentage of video files and *W* is the percentage of web-like content. The *demand* subcomponent is much simpler. It defines the item selection process (i.e., request rate and items popularity).

The content component defines the initial assignment of data items to individual nodes. That is, before starting the simulation. In addition, the relative frequency at which each individual

file is requested by peers and the average time elapsed between requests are also defined in this component.

In Section 2.1.4 we described two content ownership models to describe the process involved in the retrieval of content (creation, storage, indexing, etc.) and the roles that nodes can play with regard to content management (*publisher*, *index*, *holder* and *requestor*). In our research, the *node ownership* model is assumed. In addition, the following content indexing and retrieval model is assumed for the rest of the dissertation. Nodes play three alternative roles, namely *holder*[21], *index* and *requestor*. Content *holders* have one or more items in their storage space and they regularly contact their corresponding *index nodes* for indexing purposes. Consequently, *index nodes* have a list of all the nodes with items (i.e. *holder nodes*) that map to the portion of the key-naming space they manage. When a *requestor node* searches the overlay network for an item, it only needs to reach the *index node* to obtain a list of all the available nodes holding a copy of that item.

3.2.5 Incentives and Redundancy Component

The purpose of this work is to improve content availability in P2P networks. To do so, we propose the use of redundancy and incentive-based mechanisms. The first of these two mechanisms, redundancy, is a common practice in multiple fields as the mean to build reliable systems out of unreliable components. The second mechanism, incentives, is commonly used to model interactions among autonomous self-interested participants and as a mean to accomplish cooperation among them towards a common goal.

[21] The term *publisher* node is replaced by *holder* node to generalize data possession to the case where content availability is provided using redundancy.

In our research, we use redundancy to accomplish reliability under the assumptions that, in P2P networks, idle repair bandwidth is scarcer and more expensive than idle storage space [10]. Thus, we have a redundancy optimization problem, which needs to be formulated as a repair bandwidth minimization problem. In addition, we assume that given the heterogeneity of resources and behaviors among P2P networks' participants, incentives-based mechanisms is a proper tool to model their interactions, foster their cooperation and achieve self-organization (for the redundancy repair process of the redundancy scheme we propose).

Given the importance of these mechanisms in our work, the analysis and description of these components is presented in separate chapters. Redundancy is discussed in Chapter 4.0 and Incentives in Chapter 5.0.

3.2.6 Response Variables

For complex systems such as P2P networks it is not possible to fully characterize their operation using a single metric. Li et. al. in [71] propose a two-metric framework named cost-performance to evaluate the design tradeoffs of DHT-based P2P networks. The cost metric accounts for the amount of traffic exchanged among nodes to maintain the operation of the network and the performance metric reflects lookups delay. In our previous research [39], we shown that in some settings, the lookups success rate is an vital response metric, which we named efficiency. If the system efficiency is poor, the metrics in the cost-performance framework become irrelevant. In addition, for our content availability analysis these metrics (cost, performance and efficiency) need to be qualified at the content level. We need to define the additional traffic that should be considered

in the cost metric and whether performance should be still measured using lookups delays or other time-delay metric, such as the delay between the start of a file lookup (i.e., root lookup) and the initial transfer of data. Furthermore, since our proposed mechanisms to improve content availability in P2P networks are distributed in nature, we need metrics reflecting not only the aggregate behavior of the system (i.e., mean cost), but also the distribution of these metrics among nodes as a function of their level of participation.

In our study, we define two metric categories that we named aggregate and spread. In the aggregate category, we use three metrics: lookup success rate, content retrieval success rate and redundancy repair bandwidth. The first two metrics characterize the efficiency of the system and the third one measures cost. In the spread category, we use only one metric, which is transmission bandwidth between nodes versus content contribution, where content contribution is measured as a function of the number of data items (both fragments and complete files) stored by nodes. Therefore, the spread metric is in fact the combination of two system properties. In addition to the metrics mentioned above Table 3 presents one more response variable that is important to characterize the operation of our system, the level of churn in the network. We measure this property as the mean (or median) session length of nodes throughout a simulation.

Table 3. Response Variables

Metric	Description	Units
Cost	Redundancy-maintenance cost	Bytes/file/second Bytes/file/node/second
Efficiency: • *Routing-level* • *Content-level*	 lookup success rate content downloads success rate	%
Spread	Distribution of transmission bandwidth versus content contribution	n/a
Churn	Mean/median session length	Minutes

In addition to the metrics mentioned above, which are quantitative in nature, the evaluation of our system requires a set of metrics for which we can not derive simple quantitative results. So is the case of fairness for the mechanisms we propose. In particular, we expect that our spread metric will reflect that the performance (i.e., transmission bandwidth) of nodes remains proportional to their contribution towards the reliability of the system (i.e. content contribution).

3.3 FRAMEWORK PARAMETERS

Our content availability framework can be used to study the impact of *i*) the overlay community composition in terms of classes of user behaviors (*Benefactors* and *Peers*), *ii*) churn in terms of the node's average session length and their distribution, *iii*) the content characteristics in terms of number of resources and their query distribution and *iv*) content management strategies to improve content availability. The last topic is the focus of this dissertation.

Table 4. Content Availability Framework Parameters

Component	Number of elements	Factor(s)	Description
User Behavior	1	Ω_{Be}	Network make (user type distribution)
Node Availability	2 pairs	$E[g_n(t)]$, a_i	Average session length and average availability for each user-class
Network Structure	1	N	Network size
Content	1	M	Content space size
Incentives & Redundancy	2 sets	*Redundancy-Set* *Incentives-Set*	Redundancy method parameters & Incentive-based mechanism parameters

A full factorial design considering all the elements described in this framework has not been completed due to the extensive number of variants possible. Instead, only a reduced number

of elements have been selected to conduct our content availability study. Table 4 summarizes the elements we use as factors for the experimental portion of this dissertation.

The following paragraphs describe the levels selected for each of the factors listed in Table 4; as well as the initialization of other elements not considered as factors. Chapter 6.0 presents results for the different combinations of parameters selected.

3.3.1 User Behavior Component

Every node in the overlay network is mapped to one of the two classes used in our framework. Thus, a network make (i.e., portion of nodes in each user class) can be expressed simply with percentages. Ω_{Be} and Ω_{Pr} denote the percentage of nodes in the *Benefactors* and *Peers* user classes, respectively. Since $\Omega_{Be} + \Omega_{Pr} = 1$, only one of these values is needed to fully characterize the user population. In our experimental work we use three levels for $\Omega_{Be.}$ The selection of levels for the percentage of *Benefactors* assumes that this user class is small in most P2P application environments. The values considered are presented in Table 5.

Table 5. User Behavior Parameters

Parameter	Levels
Benefactors (Ω_{Be})	5, 10 and 20

3.3.2 Node Availability Component

To define the churn behavior of every node we use two parameters, session length and *connectivity factor*. The *connectivity factor* is a constant parameter value for each user class and for the session length we use a probability distribution function (pdf). For the *Benefactors* user class, we use a

Depending on their resources, each cluster machine hosts 88, or 176 virtual machines (VN) running one or two instances of Bamboo for a total of 1,840 overlay nodes. The FreeBSD machine uses Modelnet to enforce the wide-area delay and bandwidth restrictions of a 4000-node wide-area network topology with 1,344 client nodes connected to 836 distinct stubs by 10 Mbps to 100 Mbps links. Figure 16 illustrates the structure of the *net-model* employed in the experimental portion of the dissertation.

3.3.4 Content Component

The content component has three elements: content space size, query distribution, and type/size breakdown. The first two elements are a single value parameter, while the third one is a tuple with three values plus a model characterizing the file sizes for each data type. In addition, this section describes the procedure used to initialize the content space of individual nodes before a simulation starts.

The content space used in the experimental portion of this work is constructed as follows. A maximum number of unique items is defined for each simulation setting. In most cases, a value of 500 items is used, but 400, 750 and 1,500 unique data items are also employed in some experiments. Each item is assigned a unique numerical identifier and is assigned type and size properties according to the distributions indicated in Table 10, as reported in [70].

The selection of items each node holds at the beginning of a simulation is crucial for the performance and the evolution of the system's content. The scenario that we want to evaluate focuses on the long-tail portion of the popularity distribution. That is, those rare items for which the system has just a few whole-copies. For items highly demanded, content availability is a natural by-product of demand. Injecting additional redundancy into the system for these items

The proactive redundancy method utilized in the experimental portion of this work has three parameter values: the number of blocks a file is originally split (*n*), the storage overhead used by the mechanism (*S*) and the number of replicas that the hybrid mechanism will generate for each unique fragment (*r*). In the case of replication redundancy, which is also tested experimentally for comparison purposes, only the storage overhead parameter (*S*) is used.

Table 12. Redundancy Mechanism Parameters

Factors	Levels
Proactive Redundancy	$n = \{6, 8\}$ $S = \{2, 3, 4\}$ $r = \{2\}$
Replication Redundancy	$S = 5$
Transmission bandwidth	80, 125 [kBps]
Maintenance Epoch Policy	Static, Adaptive<, Adaptive≤, Smooth α=0.4 and Smooth α=0.8
Target number of fragments (MIN_SEG, TARGET)	0.5, 0.6, 0.7, 0.8, 0.9

3.3.5.2 Incentives-Based Mechanism

The incentive-based mechanism presented in this dissertation employs two variants of a sigmoid function. The first one is called TB-Utility and the second one is called RP-Cost. Table 13 presents the factors and levels employed for these two equations (described in Section 5.2.5) during the evaluation of the redundancy-system presented here.

Table 13. Incentive-Based Mechanism Parameters

Function	Parameters' Levels	
	Shape	*Shift/Bias*
RP-Cost	3.5, 5.5	1.25, 1.85
TB-Utility	1.25, 2.25, 3.25	0.05

4.0 REDUNDANCY

The most popular applications of P2P technology are content distribution and file-sharing. In both cases, content durability is limited by the fading popularity of items and the diverse failure content error patterns present in this type of networks; namely intermittent peer connectivity disk failures and network errors. Content redundancy can improve reliability of these systems, but given the large scale and high levels of churn in P2P networks, redundancy needs to be maintained in a timely manner. In this chapter, we analyze different aspects of redundancy for P2P networks and present an automated redundancy repair mechanism for P2P networks under churn.

Redundancy is the provision of excess resources to improve the reliability of a system. In the context of P2P networks, redundancy can be used to improve both the availability and the durability of content by distributing whole or code-based copies of a file among the network's participants.

In this chapter, we address three questions with regard to redundancy. First, how can we determine the proper level of redundancy to guarantee a desired file availability level? Second, what redundancy schemes perform better in P2P environments in terms of their repair cost? And third, how should we automate the maintenance of such redundancy? The term repair cost describes the amount of information exchanged between peers to restore the redundancy state information lost due to content errors. We tackle the questions above by building an analytical

framework that demonstrates the cost and performance advantages of our proposed redundancy scheme versus other schemes; in particular, erasure coding and network coding.

Our analytical formulation is organized in two stages. First, we describe file availability for a system where data is encoded and stored using a *k*-out-of-*N* redundancy scheme with replication. Second, we analyze the redundancy repair cost for different constructions of the *k*-out-of-*N* redundancy scheme with replication. The file availability analysis portraits the static properties of the redundancy system (i.e., resiliency) and the repair cost formulation focuses on how to maintain these properties in the long term (i.e., maintainability).

Table 14. Redundancy Notation: Redundancy Scheme(s) Parameters

Parameter	Description
k	*Reception efficiency*. Minimum number of unique fragments needed to reconstruct a file
N	Total number of unique fragments generated.
S	*Coding Gain*. $S=N/k$
M	Total number of fragments generated (including replicas)
d	*Repair degree*. Nodes needed to reconstruct a fragment using network coding redundancy. $k \leq d < N$
r	Replication gain. Number of per-fragment replicas maintained. $r \geq 1$
R_i	Replication gain for fragment type i. $R_i \geq 1$

Table 15. Redundancy Notation: File Availability

Parameter	Description
$A_{\mathcal{F}}$	File Availability (for a *k*-out-of-*N* redundancy system)
A_f	Fragment availability (for a *1*-out-of-R_i redundancy system)
Φ	Target file availability (i.e., $A_{\mathcal{F}} \geq \Phi$)
ϕ	Target fragment type availability (i.e., $A_f(R_i) \geq \phi$)
$f_{i,j}$	Fragment i, replica j. $i = 1,\ldots N$; $j = 1,\ldots,R_i$
$x_{i,j}^{m}$	Fragment i, replica j stored at node m. $m = 1,\ldots,M$
q^m	Availability of node m. $0 < q^m < 1.0$

Table 16. Redundancy Notation: Repair Cost

Parameter	Description
a	Availability. Can be used in reference to nodes, disks or other system component (e.g., a_n = node availability, a_f = fragment-replica availability)
δ	Maintenance epoch. Time interval at which repairs are performed
g_n	Distribution of nodes session lengths, μ_n =mean
g_d	Distribution of disk lifetimes, μ_d =mean
α	Fragment size
β	*Repair unit*. Data exchanged between two nodes during a repair
p_e^r	Probability of error-free disk read operation
p_e^t	Probability of error-free repair unit transmission
b_e^r	Non-recoverable disk error rate
b_e^t	Transmission bit error rate
L	*Redundancy loss*. Number of nodes that left the system
Ω	*Repair cost*. Total amount of information transferred during a maintenance epoch
T	Long-run redundancy repair interval (10 hrs)
τ	Time needed to transfer a repair unit
F	File size
B	Repair bandwidth. Average repair bandwidth between a pair of nodes.

4.1 REDUNDANCY SCHEMES

In our analysis we consider three different code-based redundancy schemes. Maximum Distance Separable (MDS) erasure coding, exact minimum bandwidth regenerating (exact-MBR) network coding and our proposed scheme, named Proactive Replication (PR). The basic structure of all these methods is a *k*-out-of-*N* redundancy scheme.

In the MDS scheme, the file content is encoded to generate a set of *N* unique fragments, each of which is stored at different nodes. The encoding scheme is such that any *k*-out-of-*N*

fragments are sufficient to reconstruct the original file. In such scheme, a file of size F results in each node storing a fragment of size $\alpha=F/k$ bytes. The regeneration of a lost fragment typically requires regenerating the original file, resulting in a repair overhead cost of $\alpha*k$ bytes.

Similarly to MDS, exact-MBR uses a k-out-of-N code. However, exact-MBR differs from MDS in the amount of information stored at each node and in the number of nodes required to reconstruct a single lost fragment. We base our analysis in the exact-MBR construction presented by Rashmi in [79]. In this scheme, nodes store fragments of size $\alpha=2*d*F/(k*(2d-k+1))$ bytes, where $k\leq d<N$ represents the repair degree. To reconstruct a single fragment, information from d distinct nodes is required, resulting in d different transfers of size $\beta=\alpha/d$ bytes from each node.

Our proposed redundancy scheme, PR, also uses k-out-of-N erasure codes. Our scheme, however, produces r replicas of each fragment. The replicated fragments are then stored at $N*r$ different nodes. This strategy obviates the need to recreate the original file to repair a single lost fragment. Thus, the cost for such repairs is only F/k bytes. To calculate file availability for this new redundancy scheme, we analyze two different scenarios. First, the case when node availability is heterogeneous and second, for homogeneous node availability.

4.2 FILE AVAILABILITY

We assume the existence of a community of nodes that cooperatively store information. This community is formed by a fully connected set of nodes $\mathbb{M}=\{n^1, n^2, \ldots, n^M\}$ that are available with probabilities $\mathbb{P}=\{q^1, q^2, \ldots, q^M\}$ respectively. Data is stored in the system using a k-out-of-N redundancy scheme with replication. The structure of this redundancy scheme is as follows. For a file of size F, we define the set $\mathbb{F} = \{f_1, f_2, \ldots, f_N\}$ as a collection of N unique fragments of equal

size (e.g., F/k) such that any subset of k elements from $\mathbb{F}$ suffices to reconstruct the original file. Let $\mathbb{G}$ be the set of all fragments contained in the system, which is a superset of replicas of fragments in $\mathbb{F}$:

$$\mathbb{G} = \left\{ \bigcup_{f_i \in \mathbb{F}} \bigcup_{j=1}^{R_j} f_{i,j} \right\} \tag{4.1}$$

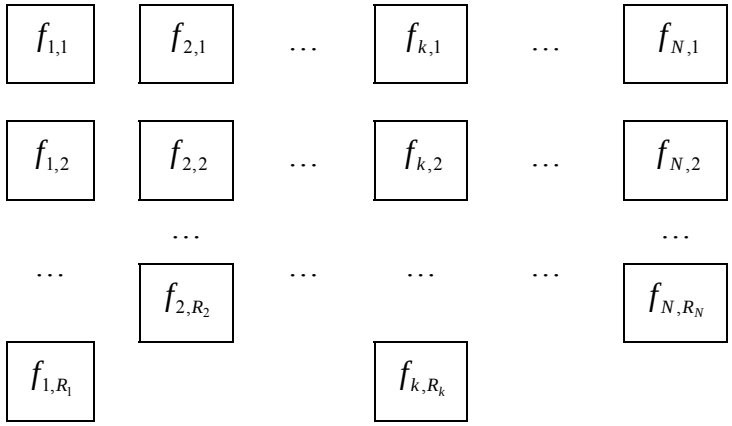


Figure 17. Redundancy Data Structure

Figure 17 shows a graphical representation of set $\mathbb{G}$. There is a total of N unique fragment types and each fragment type i (f_i) is replicated a total of R_i times, where R_is are not necessarily the same. When R_i=1 $\forall i$, and node's availability is homogeneous, file availability can be obtained using the well-known formulation for a k-out-of-N redundancy scheme, the binomial distribution. For our proposed redundancy scheme, PR, there are not analytical models available (to the best of our knowledge). Consequently, we develop our own to analyze the file availability properties of PR.

4.2.1 Heterogeneous node availabilities

The availability of a file stored using the data structure described above (for R_is>1) is given by:

$$A_{\mathbb{F}} = \Pr[\text{at least } k \text{ fragment types are available}] \tag{4.2}$$

This is a *k*-out-of-*N* redundancy scheme, where the availability of each fragment type is a 1-out-of-R_i redundancy scheme. For this redundancy structure we want to solve the following optimization problem:

$$\text{minimize:} \qquad \text{Cost}=\sum_{i=1}^{N} C(R_i) \tag{4.3}$$

$$\text{subject to:} \qquad A_{\mathfrak{F}} \geq \Phi \tag{4.4}$$

The availability of the file ($A_{\mathfrak{F}}$) for a set of nodes, M, with heterogeneous availabilities has been studied extensively and does not have a closed form solution. The methods reviewed by Kuo and Zuo in [80] for evaluating the reliability of a *k*-out-of-*N* system are enumerative in nature; thus, using them to solve our optimization problem is computationally expensive. Alternatively, we adopt a decomposition strategy to obtain a feasible closed form solution for our optimization problem. By doing so, we obtain an alternative formulation that can be easily implemented in an iterative algorithm.

Let κ_i denote the probability that at least one fragment type *i* (f_i) is available:

$$\begin{aligned}\kappa_i &= \Pr[\text{at least one fragment type } i \text{ is available}] \\ &= 1 - \Pr[\text{no fragment type } i \text{ is available}] \\ &= 1-\prod_{j=1}^{R_i}(1-q^m \chi_{i,j}^m)\end{aligned} \tag{4.5}$$

where, $x_{i,j}^m$ is an indicator function such that

$$x_{i,j}^m = \begin{cases} 1 & \text{if } f_{i,j} \text{ is hosted in node m} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

Let ϕ be a target fragment availability that we want to guarantee for all fragment types f_i such that $\kappa_i \geq \phi$. Now, using (4.5) and (4.6) in (4.4) we can obtain a closed form solution for the availability of the file:

$$A_{\mathcal{F}} = \sum_{i=k}^{N} \binom{N}{i} \cdot \phi^{i} \cdot (1-\phi)^{N-i} \tag{4.7}$$

Since equation (4.7) has the form of the binomial distribution, we can use the normal approximation to the binomial distribution to derive an optimal value (i.e., minimum) for ϕ that would satisfy the original constraint of our optimization problem (i.e., $A_{\mathcal{F}} \geq \Phi$):

$$\phi = \left(\frac{\sigma_{\varepsilon}\sqrt{S/k} + \sqrt{\sigma_{\varepsilon}^{2}(S/k) + 4(S + \sigma_{\varepsilon}\sqrt{S/k}}}{2(S + \sqrt{S/k})} \right)^{2} \tag{4.8}$$

Where S is the coding gain of the MDS erasure coding scheme (i.e. N/k), k is the reception efficiency and σ_{ε} is the number of standard deviations for the required level of file availability. For example, for a target file availability of two nines (0.99), k=8 and N=12, ϕ=0.8142.

Now, our initial optimization problem can be rewritten as follows:

minimize: $\text{Cost} = \sum_{i=1}^{N} C(R_i)$ (4.3)

subject to: $A_f(R_i) \geq \phi \quad \forall i$ (4.9)

What we have accomplished is to reduce the complexity of our file availability problem. Now, we have to guarantee the availability of each fragment type independently. Furthermore, if we consider the cost function to be a non-decreasing concave function of R_i and the availability of the set of nodes storing the same fragment type, the original optimization problem is reduced to a set of N independent optimization problems; one for each f_i plus an additional set of constraints to guarantee that each node stores at most one fragment (4.12) and that the total number of fragment does not surpass the total number of nodes (4.13):

minimize: $C(R_i) = d * \sum_{j=1}^{R_i} (1 - q^{m} \chi_{i,j}^{m}) \qquad \forall i$ (4.10)

$$\text{subject to: } A_f(R_i) = 1 - C(R_i) = d * \sum_{j=1}^{R_i} (1 - q^m \chi_{i,j}^m) \geq \phi \quad \forall i \tag{4.11}$$

$$\sum_{m=1}^{M} \chi_{i,j}^m \leq 1 \qquad \forall i, j \tag{4.12}$$

$$\sum_{i=1}^{N} R_i \leq M \tag{4.13}$$

In addition, if the availabilities (q^m) of all the nodes is assumed to be i.i.d. and equal to a, equation (4.11) can be used to obtain a closed form solution for R_i:

$$\begin{aligned} A_f(R_i) &= 1 - \prod_{j=1}^{R_i} (1-a) \geq \phi \\ &= 1 - (1-a)^{R_i} \geq \phi \\ &\textit{thus} \\ R_i &\geq \frac{\log(1-\phi)}{\log(1-a)} \end{aligned} \tag{4.14}$$

Algorithm 1 presents the pseudo-code for finding a feasible solution for the optimization problem defined by equations (4.10) through (4.13). Notice that the fragment availability is calculated using a recursive formula ($A_f(R_i+1) = A_f(R_i) * (1-q)+q$) to minimize the processing cost of the algorithm.

In order to minimize cost, nodes have to be assigned to a fragment type subject to:

$$\begin{aligned} \min \Delta\text{cost} &= C(R_i + 1) - C(R_i) \\ &= d * \sum_{j=1}^{R_i+1} (1 - q^m \chi_{i,j}^m) - d * \sum_{j=1}^{R_i} (1 - q^m \chi_{i,j}^m) \\ &= d * (1 - q^m \chi_{i,R_i+1}^m) \end{aligned} \tag{4.15}$$

and

$$\max \Delta A_f = A_f(R_i+1) - A_f(R_i+1)$$

$$= \left[1 - \left(\prod_{j=1}^{R_i} (1 - q^m \chi_{i,j}^m) \right) \cdot (1 - q^m \chi_{i,j+i}^m) \right] - \left[1 - \prod_{j=1}^{R_i} (1 - q^m \chi_{i,j}^m) \right]$$

$$= \left(\prod_{j=1}^{R_i} (1 - q^m \chi_{i,j}^m) \right) \cdot \left(1 - (1 - q^m \chi_{i,j+1}^m)\right)$$
$$= \left(\Psi(R_i)\right) \cdot q^m \chi_{i,j+1}^m \quad (4.16)$$

Procedure optimizeRedundancy()

Purpose: Find a mapping of fragment replicates to nodes

{*N*: total number of unique fragments}
{*M*: set of nodes with their availability probabilities q^m }
{MIN_AVA: minimum availability needed for each fragment type }

```
1:    for fi = 1 to N do
2:     node = select next node
3:     assign node to fi
4:     Afi = availability node
5:    end for
6:    for fi = 1 to N do
7:     while true
8:          if Afi < MIN_AVA then
9:               node = select next node
10:              assign node to fi
11:              p=availability of node
12:              Afi = Afi*(1-q)+q
13:         else
14:              break
15:         end if
16:    end while
17:    end for
```

Algorithm 1. Optimization of Redundancy Resources

Equations (4.15) and (4.16) translate into a simple, and intuitive, node selection strategy: use the nodes with the greatest node availability first. For a given set of node availabilities, Algorithm 1 can determine quickly if a target level of file availability is feasible. We implemented our algorithm in Matlab and by synthetically generating a random set of node availabilities, we can easily determine the minimum node-set required to achieved a required level of file availability. For example, Figure 18 shows some sample results assuming uniformly distributed node availability between 0.15 and 0.45. The target file availability (of 0.99) is not feasible until the node set is larger than 63 nodes.

```
>> p=unifrnd(.15,.45,50,1);
>> [sol p q]=optimizeRedundancy(p);
Using provided node availability vector...
        Target file availability:0.9900
        Target fragment (type) availability = 0.8142
.....................................................
        Unfeasible solution!! ran out of nodes
>> p=unifrnd(.15,.45,63,1);
>> [sol p q]=optimizeRedundancy(p);
Using provided node availability vector...
        Target file availability:0.9900
        Target fragment (type) availability = 0.8142
.....................................................................
DONE, Cost=42.5836
Solution:
    0.44    0.44    0.44    0.42    0.41    0.40    0.40  …    0.39    0.39    0.38
    0.38    0.38    0.37    0.35    0.34    0.31    0.30  …    0.25    0.23    0.18
    0.38    0.37    0.36    0.35    0.33    0.31    0.30  …    0.24    0.21    0.18
    0.38    0.37    0.35    0.34    0.32    0.30    0.29  …    0.24    0.20    0.17
    0       0       0       0       0       0.30    0.29  …    0.24    0.19    0.17
    0       0       0       0       0       0       0     …    0.23    0.19    0.17
    0       0       0       0       0       0       0     …    0       0.19    0.17
    0       0       0       0       0       0       0     …    0       0       0.16
```

Figure 18. Algorithm 1's Sample Output

4.2.2 Homogeneous node availabilities

File availability has been used to define the fundamental relationships between the parameters of a redundancy scheme and the reliability requirements of a system. Assuming i.i.d. node availabilities, file availability is traditionally presented as:

$$A_{\mathcal{F}} = \sum_{i=k}^{N} \binom{N}{i} \cdot a^{i} \cdot (1-a)^{N-i}, \; N{=}S{*}k \tag{4.17}$$

where:

File availability, $A_{\mathcal{F}}$. Measures the reliability of the redundancy scheme. It is frequently expressed as one or more nines of availability (e.g., $A_{\mathcal{F}} \geq 0.99$).

Node availability, *a.* Captures the unreliable nature of the components of the system.

Reception Efficiency, k. Denotes the number of fragments needed to reconstruct the original data

Coding gain, S. Is the fraction of the total number of fragments in the system over the number of fragments needed for reconstruction. Assuming all fragment to be unique, $S = N/k$.

For the construction of a redundancy system, $A_{\mathfrak{F}}$ an a are usually given. $A_{\mathfrak{F}}$ in the form of a system requirement and a as an environmental condition. The optimal values for S and k can be engineered to achieve different design objectives. For example, MDS provide an optimal tradeoff between reliability and storage space [79]. Given the tuple a, k and $A_{\mathfrak{F}}$, an optimal value (i.e., minimum) for S can be obtained (either numerically or analytically [10]) and similarly, given a value of a, S, and $A_{\mathfrak{F}}$ an optimal value for k can be determined [54].

In a system without maintenance, the file availability formulation is used to define both reliability and cost for redundancy scheme, but for a system with maintenance, file availability only determines the minimum parameter settings to achieve a required level of reliability. Cost on the other hand, has to be engineered using additional metrics and guidelines. In particular, we need to define a fragment availability model capable to capture the redundancy repair costs for our redundancy scheme as well as for other redundancy schemes. In the next two sections we introduce the cost metric traditionally used to compare different redundancy schemes and then we describe our fragment availability model.

4.2.2.1 Storage Overhead

In a k-out-of-N redundancy system for data storage, cost is measured by the ratio of the total amount of storage used by the redundancy scheme and the storage space occupied by the original data. In a P2P environment though, repair cost is more important because access bandwidth is

scarcer and more expensive than storage space [10]. Nonetheless, storage overhead remains a fundamental performance measure of the system that needs to be taken into consideration.

The next figure presents the minimum storage overhead for four redundancy schemes: MDS erasure coding, exact-MBR network coding, our proposed hybrid redundancy method, PR, with r=2, and Replication.

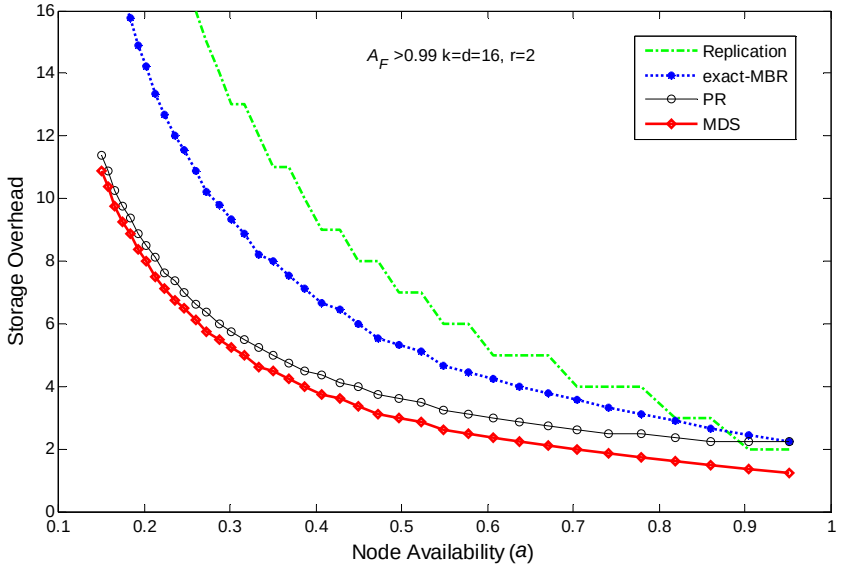


Figure 19. Minimum Storage Overhead

These results illustrate that code-based redundancy methods use less storage overhead than replication redundancy, especially at low average node availabilities. For exact-MBR network coding, the coding gain parameter, S, uses the same value as MDS, but the storage overhead at each node is higher. For a file of size F, the amount of information stored by each node is F/k in MDS, while for exact-MBR each node stores $2*F*d/k(2*d-k+1)$ [81]. For PR, each node stores F/k bytes, but the total amount of storage used is larger than MDS' storage. The PR scheme stores r copies of each unique fragment versus the single copy stored by MDS redundancy. However, for low node availabilities the additional storage overhead of PR is minimum compared with MDS. In terms of storage overhead savings, MDS is the most efficient method, but its redundancy

maintenance cost is high. Exact-MBR network coding redundancy and PR redundancy utilize additional storage in order to minimize repair maintenance cost, which we analyze in the following sections.

In addition to storage overhead, the results presented in Figure 19 can be used to determine another important system metric. The total number of nodes needed to store a file. We refer to this property as node-set. For replication redundancy, this parameter's value is the same than storage overhead. For exact-MBR and MDS, the node-set is the product of the coding gain times the reception efficiency, $N = S*k$. For PR, the node-set is the product of the coding gain, the reception efficiency and the replication factor, $m = N*r = S*k*r$. Consequently, in terms of the number of nodes needed to store a file, replication redundancy requires the least number of nodes, followed by MDS and exact-MBR, and finally, by PR. For MDS and exact-MDS, small node availabilities require larger node-set values which translate into an added coding/decoding complexity. For PR, this increment in coding/decoding complexity also applies, but at a lower degree. In our opinion, large node-set values should be avoided not only for their added coding/decoding complexity, but for their practical implications. As the number of nodes needed to store a single file increases, managing the node-set becomes more complex. For example, for a=0.3, MDS and exact-MBR require a minimum coding gain of S=5.25, which implies a redundancy-set with at least 84 nodes.

4.2.2.2 Fragment Availability Model

For the calculation of file availability using a k-out-of-N redundancy system most of the research literature equates the availability of individual fragments with the availability of the host holding them. In our case, we use a broader model for the availability of these fragments. This model reflects the probability of hardware errors in addition to the traditional host availability component. Furthermore, our formulation derives host availability based on an analytical model of the node's

session length probability distribution, rather than system traces [10, 56, 58]. This approach has two advantages. First, results can be obtained for a wider range of network conditions. That is, we are not limited to the availability of traces. Second, the relationship between the system maintenance epochs and the average node availability is expressed explicitly, which allow us to define a repair frequency for a redundancy maintenance algorithm that would maximize the performance of the system.

In our model, a_f denotes the availability of a single fragment. This metric is the product of two probabilities. First, the probability that the node storing the fragment is available. Second, the probability that if that node is available, no hardware errors will prevent access to the fragment. Other research initiatives consider host availability [56] or disk failures [9] as the only sources of content errors. Conversely, our model is broader. It can be customized to model environments where host availability is dominant as well as environments where hardware and communication errors are important (e.g., wireless networks). We consider the probability of disk failures and the probability of data corruption as the main sources of hardware errors. In turn, we use disk read errors and transmission errors to construct the data corruption component of our model. For a given system maintenance epoch δ, fragment availability is given by:

$$a_f^{\delta} = \Pr[\text{node is available} \mid \delta] * \Pr[\text{no hardware errors} \mid \delta] \quad (4.18)$$

The first term in the right side of equation (4.18) measures the residual lifetime probability of the node storing the fragment given that the node has survived one maintenance epoch. In other words, this probability measures if the fragment can be used during the next maintenance interval to reconstruct the original file or to perform repairs. We use a_n to denote this probability. The probability that a fragment is available after a maintenance epoch, can be calculated using the expression derived in [9]:

$$a_n^{\delta} = \Pr[\, x - \delta \mid x > \delta \,] = \int_{\delta}^{\infty} \frac{x \cdot f_n(x)}{\mu_n} \cdot \frac{x-\delta}{x} dx = \frac{1}{\mu_n} \int_{\delta}^{\infty} f_n(x) \cdot (x-\delta) dx \qquad (4.19)$$

where the term $(x-\delta)/x$ reflects the probability of storing a fragment early enough in a node's session so that it is still available after the next maintenance epoch, *gn* is the node's session length probability distribution, and μ_n is the expected value of this distribution (i.e., mean value).

The second term in the right side of equation (4.18) has three factors: the availability of disks storing the fragment, the probability of non-recoverable read errors, and the probability of corrupted data transmissions. We use a_e to denote the resulting probability, a_d^{δ} denotes disk availability, p_r denotes uncorrupted read operations and p_t denotes successful data transmissions.

Pr[no hardware errors|δ] = Pr[disk available|δ]*Pr[no read errors]*Pr[no tx errors]

$$a_e^{\delta} = a_d^{\delta} \cdot p_r \cdot p_t \qquad (4.20)$$

Substituting (4.19) and (4.20) in (4.18) we have

$$a_f^{\delta} = a_n^{\delta} * a_e^{\delta}$$

$$= a_n^{\delta} * a_d^{\delta} \cdot p_r \cdot p_t \qquad (4.21)$$

The disk availability component, a_d^{δ}, in equation (4.20) can be obtained using a similar approach to the one used for a_n^{δ}. That is, we can use equation (4.19) by replacing g_n and μ_n with the disk's lifetime probability distribution and its respective mean value, which we denote by g_d and μ_d respectively.

$$a_d^{\delta} = \int_{\delta}^{\infty} \frac{x \cdot f_d(x)}{\mu_d} \cdot \frac{x-\delta}{x} dx = \frac{1}{\mu_d} \int_{\delta}^{\infty} f_d(x) \cdot (x-\delta) dx \qquad (4.22)$$

To derive the probabilities of uncorrupted disk read operations and successful data transmissions we use a simple binomial model and assume that bits fail independently. Let α

denote the size of a fragment, b_r denote the disk's non-recoverable read error rate, and b_t denote the transmission bit error rate. Then, p_r and p_t can be obtained as follows:

$$p_r = (1 - b_r)^{\alpha} \tag{4.23}$$

$$p_t = (1 - b_t)^{\beta} \tag{4.24}$$

We assume that to retrieve the data needed for repairs, nodes must read the whole fragment. The size of a fragment is denoted by α, and β denotes the amount of information transferred to another node during a repair. In MDS and PR redundancy $\beta = \alpha$, and in exact-MBR redundancy $\beta = \alpha/d$. Placing (4.23) and (4.24) in (4.21) we get the following expression for the availability of a fragment.

a_f^{δ} = Pr[node is available | δ]*Pr[no hardware errors | δ]

$$a_f^{\delta} = a_n^{\delta} * a_d^{\delta} * (1 - b_r)^{\alpha} * (1 - b_t)^{\beta} \tag{4.25}$$

With this, our fragment availability model is complete. It comprises the system maintenance epoch (δ) the availability of nodes (a_n^{δ}) and disk (a_d^{δ}), and the probability of disk-read errors ($(1 - b_r)^{\alpha}$) and finally the probability of failed transmissions ($(1 - b_t)^{\beta}$). With this model, we can formulate a repair cost metric for our proposed redundancy scheme and compare it against other redundancy schemes.

4.3 REDUNDANCY REPAIR

When nodes leave the system, the reliability of the redundancy method deteriorates. To avoid permanent data loss, the system needs to reconstruct any redundancy lost regularly. The amount of

data transferred between nodes to maintain the reliability of the system is referred as repair cost. We use $\Omega_{\mathfrak{F}}$ to denote this metric.

The two fundamental system components that determine $\Omega_{\mathfrak{F}}$ are redundancy scheme, and maintenance epoch. The structure and parameters of the redundancy scheme determine the amount of information to be transmitted to reconstruct the redundancy lost. The system maintenance epoch determines the frequency and number of repairs to be performed during each maintenance interval. For longer maintenance epochs, a larger number of nodes leave the system.

We assume that at time t=0 we have m hosts cooperatively storing a file's fragments. We also assume that nodes store single fragments and that their availability, a_f^{δ}, is homogeneous and stationary. Let $L_{\mathfrak{F}}^{\delta}$ denote the expected number of nodes lost during a maintenance interval (with length δ). The expected value of $L_{\mathfrak{F}}^{\delta}$ can be obtained as the expected value of a binomial distribution with a total of m items:

$$\begin{aligned} E\left[L_{\mathfrak{F}}^{\delta}\right] &= \sum_{i=1}^{m} l_i \cdot \Pr[L_{\mathfrak{F}}^{\delta} = l_i] \\ &= \sum_{i=1}^{m} i \cdot \binom{m}{i} \cdot (a_f^{\delta})^{m-i} \cdot \left(1 - a_f^{\delta}\right)^{i} = m \cdot (1 - a_f^{\delta}) \end{aligned} \tag{4.26}$$

The number of hosts storing the file (i.e., m) is determined by the parameters of the redundancy method. Notice that the expected value of the binomial distribution is obtained for $1 - a_f^{\delta}$ because we want the number of offline nodes, rather than the number of nodes that remain active.

Once the number of fragments lost is known, we can calculate the cost of repairing them. We assume that all the redundancy lost during a maintenance epoch is fully repaired during the next maintenance interval. Let T denote a long period of time (e.g. 10 hours). We use the term

long-run repair cost to refer to the average redundancy repair cost of the system during T. Thus, repair cost is a recurrent process performed T/δ times in average. Given T and δ, the long-run maintenance cost of a k-out-of-m system using MDS, exact-MBR and PR redundancy is given by:

$$\Omega_{\mathbb{S}}^{\delta} = \begin{cases} (L_{\mathbb{S}}^{\delta} + k) \cdot \beta_{EC} \cdot \dfrac{T}{\delta^2} & MDS \\ (L_{\mathbb{S}}^{\delta} \cdot d) \cdot \beta_{MBR} \cdot \dfrac{T}{\delta^2} & exact\text{-}MBR \\ (L_{\mathbb{S}}^{\delta} + \gamma \cdot k) \cdot \beta_{PR} \cdot \dfrac{T}{\delta^2} & PR \end{cases} \tag{4.27}$$

In all the expressions in equation (4.27), the last term correspond to the frequency of repairs (1/seconds) and the remaining terms correspond to their size (bytes). The cost contributions of these components increase in opposite directions. For short maintenance epochs, the frequency component is high and size is small[23]. Conversely, for long maintenance epochs the frequency component gets smaller, but the size component becomes larger. Thus, a fundamental problem is to determine if there is an optimal setting to balance the cost of these components for each redundancy scheme.

The first expression in equation (4.27) corresponds to MDS erasure coding redundancy when no nodes are available with a complete copy of the item. In this scenario, nodes stores $\alpha = F/k$ bytes; where F is the size of the file. For repairs, nodes connect to k available nodes and transfer $\beta = \alpha = F/k$ bytes from each node to reconstruct the original data and then transfer the L fragments lost to new nodes. The research literature refers to this scenario as the redundancy repair problem due to the potential high overhead cost [82].

The second expression in equation (4.27) corresponds to exact minimum-bandwidth regenerating (exact-MBR) network coding redundancy. Authors present this type of redundancy as

[23] However, for short maintenance epochs, the size efficiency of EC is worst (i.e., highest overhead).

an alternative to traditional MDS erasure coding [79, 82]. Exact-MBR codes can repair individual fragments without reconstructing the original file first, but they incur in an additional storage overhead in every node (compared with MDS). Some of these solutions are not practical because the repair degree is m - 1, like in [78]. However, the exact-MBR code presented by Rashmi, et.al., in [79] allows the selection of the repair degree, d, independently of the other parameters. For this exact-MBR code construction, the size of each fragment is $\alpha = 2*d*F/k(2*d-k+1)$. For repairs, nodes contact d available nodes and transfer $\beta = \alpha/d$ bytes from each.

The third expression in equation (4.27) is another alternative to the redundancy repair problem of MDS erasure coding, a hybrid redundancy method that we name Proactive Replication, PR. Our approach differs from previous analysis of hybrid redundancy mechanisms in several aspects. First, we are presenting a full analytical formulation for the properties of this redundancy method (i.e., the file availability and repair cost equations); as opposed to using it as a simplified redundancy repair scenario where a complete file copy is assumed to be available all the time, like in [10] or [59]. Second, the structure of our hybrid redundancy method is different from other hybrid methods presented in the literature. Again, in [10] a complete copy of the file exists in parallel to the EC redundancy data. In [60] MDS is an auxiliary mechanism to replication; when not enough replicas are available, the system generates MDS blocks to reach a target file availability. In our approach, we use replication on top of MDS. The MDS component is used for reliability and replication is used to minimize repair cost. This mechanism can be conceptualized as either a new hybrid redundancy method, or as a proactive redundancy repair policy. In either case, the main feature of the mechanism is a significant reduction of the repair cost. In PR, the size of each fragment is $\alpha = F/k$. For repairs, if there is at least one fragment replica available, nodes contact a single node to transfer $\beta = \alpha = F/k$ bytes. When there is no fragment replica available,

$$
\begin{aligned}
R(\tau) &= \Pr[\textit{success repair} \mid \tau]^{-1} \\
&= \left(\Pr[\textit{nodes available} \mid \tau] \cdot \Pr[\textit{no hardware errors} \mid \tau]\right)^{-1} \\
&= \left((a_n^{\tau})^2 * a_d^{\tau} \cdot p_r \cdot p_t\right)^{-1} \\
&= \rho^{-1}
\end{aligned}
\tag{4.30}
$$

and

B = Repair bandwidth between a pair of nodes (4.31)

Individual repairs are successful with probability ρ. This probability depends on the probability that each pair of nodes exchanging data remains active long enough to complete the data transfer (i.e., τ seconds) and the probability of no hardware errors; which in turn depends on the amount of information stored and transferred (i.e., α and β). To obtain the value of τ, we assume a constant value for the amount of bandwidth committed to repairs. Let B denote this value, then $\tau = \beta/B$. The probability ρ is obtained using a similar approach to the formulation of a_f^{δ}, with the exception that in this case, two nodes must remain active:

$$R(\tau)^{-1} = (a_n^{\delta})^2 * a_d^{\delta} * (1-b_r)^{\alpha} * (1-b_t)^{\beta} \tag{4.32}$$

4.3.2 Repair Cost

The following figure presents the repair cost across a wide range of average node availabilities for MDS, exact-MBR, PR and the ideal version of MDS redundancy. The ideal version of MDS assumes that fragments can be repaired transferring only β bytes (i.e., without reconstructing the original file first). The parameters used for these calculations are listed in Table 17. For the expected value of the disk lifetime distribution we use the annual failure rate (AFR) reported in [83]. For the probability of bit read errors, we assume the disk specifications of a consumer-class SATA hard-drive [84] and for the transmission bit error rate, we assume the value reported in [85].

epochs? The obvious answer is average node availability, but this implies tracking down the sessions (or residual lifetimes) of peers. This presents its own set of challenges [44] and might not reflect accurately fragment availability at the file granularity level. Instead, we propose the use of the redundancy loss model presented earlier in equation (4.26):

$$L_{\mathbb{F}}^{\delta} = \sum_{i=1}^{m} i \cdot \binom{m}{i} \cdot (a_f^{\delta})^{m-i} \cdot \left(1 - a_f^{\delta}\right)^{i} = m \cdot (1 - a_f^{\delta}) \tag{4.26}$$

This model establishes a direct relationship between average fragment availability and the number of fragments lost during a maintenance interval. Therefore, instead of measuring node availability, we can simply count the number of fragments lost during a maintenance epoch. Furthermore, we can account for this metric at different levels of the granularity. For instance, we can track only the total number of missing fragments (i.e., the MDS component) or the total number of fragments with 1, 2 or *r* copies available (i.e., MDS plus replication). Using the number of fragment lost (or alternatively, the percentage of fragments still available) is simpler to implement and manage than nodes' sessions lengths. For the rest of this work, it is assumed that the availability properties of the overlay network are obtained using this metric.

Next, we address the architectural and system-level design issues related with the creation of an automated redundancy maintenance mechanism.

4.4 REDUNDANCY MAINTENANCE

The previous section provides us with insights about the selection of a redundancy method for P2P environments under churn. For the remainder of this work, we assume the use of a Proactive Replication, PR, redundancy scheme. The coding component of this method is a

4.4.1 System Architecture

The system is modeled as a three layers architecture, shown in Figure 25. At the bottom layer, we have a set of core functionalities. The middle layer manages the redundancy state information of the system and the upper layer handles the exchange of data among nodes. The operation of the upper layers depends on the functionality provided by the underlying level(s).

The bottom layer, core services, implements the fundamental functionalities on top of which the system is build. In particular, a key-based lookup service for location/indexing of content. The objective of this component is to define a single (still dynamic) point of contact for content retrieval and redundancy management. All structured P2P systems support this functionality, and some of the most important deployed unstructured P2P systems (such as Gnutella and BitTorrent) have it implemented. This functionality maps content identifiers to a key-based naming space (key space) that provides an easy and consistent access to contents independently of their popularity rank. Nodes cooperatively manage this location/indexing service by assuming responsibility for a portion of the key space, which works as a rendezvous point for the nodes storing content and those requesting it.

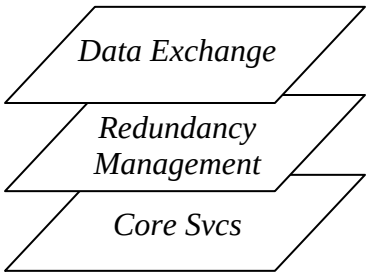


Figure 25. System Architecture for Redundancy Maintenance

The middle layer, redundancy management, uses the information stored in the key space to monitor the redundancy state information of the system. We assume that all nodes storing items

Nodes alternate between roles according to the event being handled. For example, if the node receives an item registration message, the index agent gets activated to process the request. Agents operate at the content level, which implies that nodes can execute multiple instances of the same agent concurrently, each associated with a different item. For instance, a peer can be an Index for items i_i and i_{i+1}, a Holder of item i_j and a candidate for item i_k.

Holder nodes maintain a list of index nodes for the items they currently store, contacting them periodically to indicate the availability of these items. Index nodes maintain a list (i.e., index) of all holder nodes with copies of a particular item. This indexing information reflects the current redundancy state information of the system; thus, it can be used to determine when to perform repairs. When a repair for item i_i is needed, holder nodes for this item are notified to start a repair process, which consist on contacting one or more candidate nodes and transfer data to them. Upon finalizing the data transfer, candidate nodes become holder nodes for item i_i.

4.4.3 System Processes

The redundancy maintenance system we are proposing can be modeled as a dynamic system of autonomous interacting agents. The interaction of these agents is circumscribed to three fundamental (interconnected) processes:

- Registration
- Redundancy Evaluation (maintenance)
- Redundancy Repair

Figure 26 shows the complete procedural flow of the automated redundancy maintenance system we are proposing. Peers are indicated by circles and labeled according to the agent being executed at each stage. The procedures and functions for the algorithms we are presenting are

without the proper incentives, nodes will avoid cooperating in the redundancy maintenance process, which could lead to the overall unfairness and performance degradation of the network. In other words, individual nodes will try to obtain a maximum gain from the system committing the least amount of resources possible unless we define cooperation rules to balance the performance and capabilities of individual nodes with the overall performance goal of the system. We address this concern in the next chapter. For the rest of this section we assume that nodes cooperate fully.

```
Procedure Registration()

    Purpose: Send keep_alive messages to item's root nodes

    {n: total number of items this nodes holds}
    {item_key: key value of item i}
    {item_seg: fragments of item i that this node holds}

 1:  for item = 1 to n do
 2:      Get item_key
 3:      Get item_seg
 4:      while ( next_registration_time(item)-now <= 0 ) do
 5:          if root(item) == null then
 6:              root(item)=lookup(item_key)
 7:          end if
 8:          send registration_msg(root(item), item_key, item_seg)
 9:          success = wait_for_ack()
10:          if success then
11:              set next_registration_time(item) = minimum + random(mean)
12:          else
13:              root(item) = null;
14:              set next_registration_time(item) = now
15:          end if
16:      end while
17:  end for
```

Algorithm 2. Items Registration Process at Holder Agent

The processes we are presenting define the set of rules and algorithms that nodes should follow to cooperatively improve the availability of content. The first process, registration, is the only self-initiated process. The other two processes are triggered in response to events generated by the registration process. Registration starts when *Holder* agents attempt to contact their index nodes to announce the availability of the content they hold. The procedure performed by *Holders*

is presented in Algorithm 2. The objective of this algorithm is simple, to push the state information of each node sharing content into the overlay.

Unless noted otherwise, it is assumed that key values are used to uniquely identify items within the algorithms presented.

To improve the efficiency of the registration process, the system allows the aggregation of items. That is, when several items share the same root node, a single registration message is used. At the receiving end of the registration process, *Index* agents process each registration message and trigger the redundancy evaluation process, if needed. Algorithm 3 shows the pseudo code for the *Index* agents. The interaction between the *Index* agent for item i_i and all its *Holder* agents provide the fundamental functionality of the system, a dynamic distributed content indexing service; as described earlier for the core services layer of the system architecture.

Table 19. IndexEntry Data Structure

Field	Description
File_ID	Contains a hash key
Last_evaluation	Contains a time stamp
Last_unique	Contains a file availability metric
Redundancy-SetTable	Contains an array of: *Peer* — Contains an IP socket *Fragment(s)* — Contains a bit map *Last_registration* — Contains a time stamp

Index agents maintain an IndexEntry data structure, shown in Table 19, for each of the items they are responsible for. The information, per file, consists of four elements: the item ID in the key-naming space, one time stamp, the result of the last file availability evaluation and a table with all the nodes in the redundancy-set of the item. For this redundancy-set table, three values are recorded for each node: peer ID (i.e., socket), a bit-map describing the portion of the original data stored by each peer and a time stamp of the last registration message received from each node.

When an *Index* agent determines that a node (which recently joined the overlay), is the root node of one or several of the items it is currently indexing, it transfers the respective IndexEntry data structure to the new node. On the other hand, when an *Index* node leaves the overlay, the IndexEntry data structure is lost. In this case, the data structure is regenerated once the *Holder* agents detect the failure of their current root node and contact a new one.

The redundancy evaluation process is triggered in an *Index* agent only at the beginning of each maintenance epoch. The MAINT_WINDOW constant defines this interval. By default, the system uses a value of 30 seconds. Only the registration messages received within this interval trigger the redundancy evaluation process; all others are just used to update the IndexEntry data structure.

```
Procedure Handle_Registration(registration_msg)

    Purpose: Update IndexEntry for all items in registration_msg and evaluate if repairs are needed

    {x: number of items contained in registration_msg}
    {EPOCH: default redundancy maintenance interval}
    {shift_epoch: adapts maintenance epoch dynamically. Updated by Evaluate_Redundancy process}
    {MAINT_WINDOW: interval during which redundancy evaluation can be triggered}

 1:   peer = getHolder(registration_msg)
 2:   now = current time
 3:   for item = 1 to x do
 4:       id = get_key(item)
 5:       segments = get_fragments(item)
 6:       IndexEntry = getIndexEntry(id)
 7:       update IndexEntry.Redundancy-SetTable using (id, peer, segments, now)
 8:       if ( now – IndexEntry.last_evaluation ) > (EPOCH + shift_epoch) then
 9:           IndexEntry.last_evaluation = now
10:       end if
11:       elapsed = now – IndexEntry.last_evaluation
12:       if elapsed < MAINT_WINDOW then
13:           initiate redundancy evaluation using peer
14:           IndexEntry.last_evaluation = now;
15:       end if
16:   end for
```

Algorithm 3. Items Registration Process at Index Agent

The pseudo code for the redundancy evaluation process for *Index* agents is presented in Algorithm 4-6. Algorithm 5 and Algorithm 6 are mutually exclusive; these are two variants of the

algorithm used to adapt the system maintenance epochs dynamically. Algorithm 5 uses a smoothed average of the number of unique fragments available to control the maintenance epochs of the mechanism, while Algorithm 6 uses the number of fragments lost (or gained) during a single maintenance epoch.

The pseudo code for the redundancy repair procedure performed at *Index* nodes is presented in Algorithm 7. The process is quite simple; the *Index* agent gets a random candidate node for each of the bits the holder node *peer* can repair. These candidates are obtained from the IndexEntry data structures of other items *Index* node is root, or directly from its routing table. The only condition for this selection is that the candidate node must not be listed in the IndexEntry data structure of the item being repaired. In addition, a set of backup targets is added to the message in case that any of the preselected candidates is not longer available.

```
Procedure Evaluate_Redundancy(peer)

    Purpose: Evaluate if repairs are needed, and update the next maintenance epoch.

    {peer: node that triggered this process}
    {MIN_SEG: minimum number of unique segments required}
    {DO_SMOOTH: controls which algorithm is used to update the maintenance epoch}

 1:  remove stalled entries
 2:  unique = unique segments available
 3:  if unique > MIN_SEG then
 4:      if DO_SMOOTH then
 5:          shift_epoch = Update_MaintenanceEpoch()
 6:      else
 7:          shift_epoch = Update_MaintenanceEpoch_L()
 8:      end if
 9:      if cpl < 1.0 then
10:          missing = get IndexEntry missing fragments
11:          peer_seg = get fragments peer has
12:          if (missing & peer_seg > 0) then
13:              initiate redundancy repair using peer and missing & peer_seg
14:          end if
15:      end if
16:  end if
```

Algorithm 4. Redundancy Evaluation by Index Agent

The redundancyFix message generated by *Index* nodes is received by a holder node and processed using the pseudo code presented in Algorithm 7. Upon receiving this message, the holder node will attempt to push a replica of the segments requested. The initial target for this data transfer is the candidate node previously selected by the *Index* node (Algorithm 8). Holder nodes use an auxiliary data structure named pending replicas to monitor the progress of each repair. If the replication fails or stales for any reason, the holder node will retrieve an alternate candidate node from the pending replicas data structure and to complete the redundancy repair with it. Once the repair request is completed, or all target entries have been exhausted, the pending replicas data structure for item *i* is cleared.

```
Function Update_MaintenanceEpoch()

    Purpose: Adjust the length of the maintenance epochs of the system.

    {m: total possible number of unique segments}
    {alpha: smoothing average factor, default = 0.4}
    {TARGET: number of fragments available [k…m]; default 0.3*m }

 1:  unique = unique segments available
 2:  avg = alpha*last_unique+(1-alpha)*unique
 3:  if avg < TARGET then
 4:      result = -10 seconds
 5:  else
 6:      if avg == m then
 7:          result = 0
 8:      else
 9:          if avg > 0.8*m then
10:              result = 5 seconds
11:          else
12:              result = 2.5 seconds
13:          end if
14:      end if
15:  end if
16:  last_unique = avg
17:  return shift_epoch + result
```

Algorithm 5. Smooth Maintenance Epoch Mechanism at Index Agent

In summary, the automated redundancy maintenance mechanism we are proposing is composed of three fundamental processes: registration (with instances at *Holder* and *Index* nodes), evaluation (performed exclusively by *Index* nodes) and repair (initiated by *Index* agents, but

executed entirely by *Holder* agents). The effectiveness of these mechanisms depends greatly on nodes participating on these processes. To promote this participation while preserving node autonomy, we propose the use of economic incentives. These incentives, define a set of simple, yet effective, rules for fair exchange of resources in the overlay that can be easily integrated into the automated redundancy maintenance process describe above.

```
Function Update_MaintenanceEpoch_L()

    Purpose: Adjust the length of the maintenance epochs of the system.

    {m: total possible number of unique segments}
    {TARGET: number of fragments available [k...m]; default 0.3*m}

 1:  unique = unique segments available
 2:  loss = last_unique-unique
 3:  if loss > 0 then
 4:      if loss > TARGET then                    result = -10 seconds
 5:      else
 6:          if loss > TARGET/2 then              result = -2.5 seconds
 7:          end
 8:      end
 9:  else
10:      if loss == 0 then
11:          result = 0          // in Adapt>=0, result=2.5
12:      else
13:          if loss < -TARGET/2 then    result = 5 seconds
14:          else    result = 2.5 seconds
15:          end if
16:      end if
17:  end if
18:  last_unique = unique
19:  return shift_epoch + result
```

Algorithm 6. Adaptive Maintenance Epoch Mechanism at Index Agent

Procedure RedundancyFix(*peer*, *segs*)

Purpose: Send a redundancy repair request to peer.

{*peer*: holder node that will perform the repair}
{*segs*: bit-map with the list of fragments to be repaired}
{BACKUP: it controls whether to include backup targets or not}

```
 1:  for i=1 to num_bits_in(segs) do
 2:      target = select random candidate
 3:      seg = select random bit in segs
 4:      add (target,seg) to redundancyFix_msg
 5:  end for
 6:  if BACKUP then
 7:      for i=1 to 6 do
 8:          target = select random candidate
 9:          add(target,nil) to redundancyFix_msg
10:      end for
11:  end if
12:  send redundancyFix_msg to peer
```

Algorithm 7. RedundancyFix at Index Agent

Procedure handle_RedundancyFix(*msg*)

Purpose: Perform redundancy repairs.

{*msg*: redundancyFix message received}

```
 1:  id = get item key from msg
 2:  for i=1 to size(msg) do
 3:      target = get target i
 4:      seg = get fragment i to be repaired
 5:      if seg <> nil then
 6:          push seg to target
 7:          store (id, target, seg) in pending replicas
 8:      else
 9:          store (id, target, nil) in pending replicas
10:      end
11:  end for
```

Algorithm 8. RedundancyFix at Holder Agent

figure shows the results we expect to achieve. Notice that the behavior of nodes still is random and heterogeneous in nature, but their content availability contribution exhibits an upward trend.

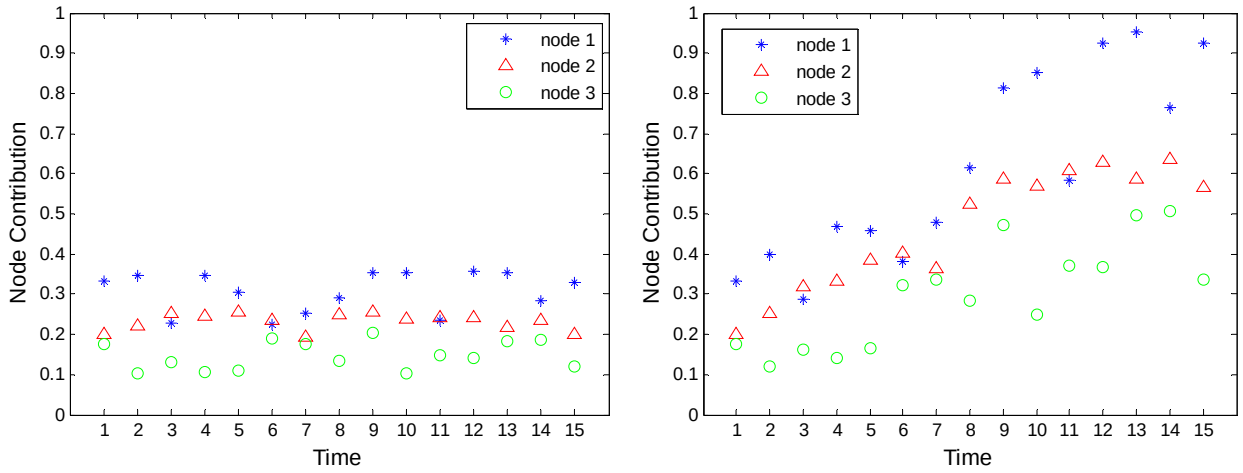


Figure 27. Effect of Incentive-based Mechanism: a) Normal System, b) System with Incentives

5.2.2 Design Guidelines

Four design properties are sought for the incentive-based mechanism presented in this dissertation: *incentive*, *penalty*, *decentralization*, and *adaptability and lightweight* [17]. The *incentive* property refers to the creation of a positive encouragement component that can affect the behavior of nodes so that the utility of the system increases. In contrast, the *penalty* property defines the negative effect that non-compliant nodes may face by contributing few to no resources towards the common goal. The *decentralization* property is adopted as a fundamental architectural consideration to improve the scalability and robustness of the system. Last, the *adaptability and light weight* property refers to the capability of the incentive-based mechanism to adapt efficiently to different networking and node behavior scenarios while minimizing the produced overhead.

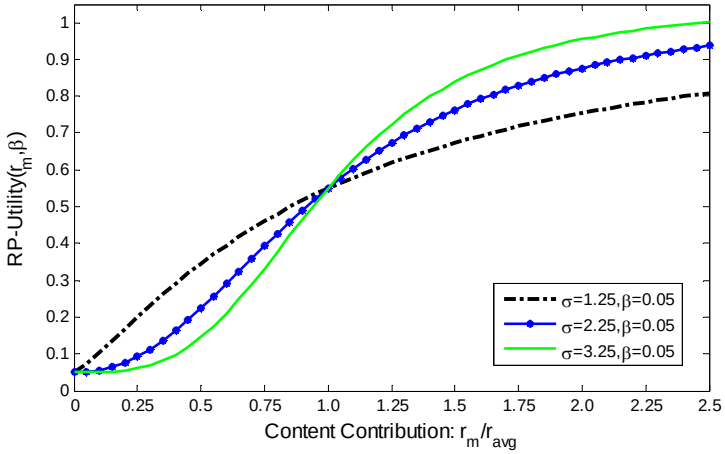


Figure 30. Transmission Bandwidth Function

5.3 INCENTIVE-BASED MECHANISM CONSTRUCTION

The use of the *RP-cost* and *TB-utility* functions in the redundancy-maintenance process is described using a finite state machine (FSM) representation of this process. The naming and operation of the states of this FSM correspond to the economic agent model presented earlier in Section 5.2.3.

5.3.1 Redundancy-Maintenance Finite State Machine

The FSM in Figure 31 presents the state transitions of a single physical node using the incentive-based mechanism. The physical node could be performing multiple roles (i.e., agents) simultaneously, but for the sake of clarity throughout this description it is assumed that the node can only be in one state at a time. Throughout this description the term node is used in reference to the local physical node and the term peer is used in reference to other physical nodes.

Function getSpeed(*cont*)

Purpose: calculate maximum bandwidth that can be assigned to this request
Returns: maximum bandwidth, speed [kBps]

{ *cont*: is the content contribution of the requesting node }
{ MIN_TXBW: minimum transmission bandwidth }

```
1:  upBW = freeUploadBW()
2:  if upBW > MIN_TXBW then
3:      if cont>0 then
4:          speed = MIN_TXBW + upBW*utility(cont)
5:      else
6:          speed = MIN_TXBW + ( upBW*random() )%MIN_TXBW
7:      end if
8:  else
9:      speed = 0
10 end if
```

Algorithm 9. Function getSpeed()

There are procedures used in the function getSpeed() for which an algorithm has not been presented. These procedures are freeUploadBW() and random(). The first is a function that returns the total amount of upload bandwidth not yet committed by the node and random() is a uniformly distributed random variable between zero and one. The random() function is used for non-compliant peers and for compliant peers with zero contribution. It is possible to use a different function for non-compliant peers, but this does not represent any additional gain based on our *penalty* design guideline (Section 5.2.2).

The pseudo-code of the function utility() used in the getSpeed() function is presented in Algorithm 10. This function uses a procedure named IndexTable.GetAverageContribution(). The purpose of this procedure is to calculate the average contribution of peers that register their items at the current local node (*Index* agent). If this value is larger than a predefined average node contribution (TARGET_CONT), the node will use this value instead for normalizing the content contribution of the requesting peer. It is assumed that a larger than expected average content contribution is due to the existence of a larger than expected content space. In such a case, the system would be able to adapt its behavior accordingly.

```
Function Utility(cont)

    Purpose: calculate utility function for requesting node
    Returns: utility value, u [UTIL_BIAS,1]

    { cont: is the content contribution of the requesting node }
    { TARGET_CONT: default normalization value for content contribution }
    { SIGMA_UTIL: shape parameter }
    { UTIL_BIAS: bandwidth bias parameter}

1:  avg = IndexTable.GetAverageContribution()
2:  if avg > 1.5*TARGET_CONT then
3:      norm = avg
4:  else
5:      norm = TARGET_CONT
6:  end if
7:  a = (cont/norm)^SIGMA_UTIL
8:  u = UTIL_BIAS + a/(1+a)
```

Algorithm 10. Function Utility()

5.3.1.4 Index State

When a node receives a *registration request*, it transitions into the *Index* state and evaluates whether the originating peer should receive a *repair request*. Algorithm 4 presents the pseudo code employed by the *Index* node for this evaluation. As described in Section 4.4.3, Index nodes are in charge of decisions in the redundancy repair process. They are the brain of the process and the actual cost associated with the transfer of information is responsibility of other peers. In this way, *Index* nodes do not have a direct incentive to misbehave.

The number of repairs to be made for a single file can vary greatly depending on the set of nodes participating in the storage process. One approach to minimize this maintenance cost is selecting nodes with the highest availability, but this represents a negative incentive for highly available nodes and a scalability concern as well. In our implementation, we use an alternative approach to balance the load of the redundancy mechanism instead. The IndexTable data structure, presented in Table 23, keeps a list of the nodes within its array of IndexEntry elements (see Table

19 in section 4.5.3 for a description of this data structure). When the list of targets is built, nodes with the lowest contributions are added first.

Table 23. IndexTable Data Structure

Field	Description
Content	Contains an array of < IndexEntry>
Nodes	Contains an array of <peers>

5.3.1.5 Candidate State

Nodes transition to the *Candidate* state when they receive a Storage request from a *Holder-publisher* node. In order to accept/reject the Storage request, *Candidate* nodes evaluate their cost function. If the value is above a randomly generated number, the request is accepted and the *Holder* node is notified that it can initiate its upload (with a maximum bandwidth proportional to the content contribution of the publishing *Holder* node). The pseudo code employed by *Candidate* nodes is listed in Algorithm 11 and Algorithm 12.

```
Procedure AcceptReplica(cont)
    Purpose: accept/reject Replica request from Holder peer
    { cont: is the content contribution of the Holder peer }
    { mycont: is the content contribution of this node }
1:  if cost(mycont) > random() then
2:      bw = utility(cont)
3:      send accept message(bw)
4:  else
5:      send reject message()
6:  end if
```

Algorithm 11. Procedure AcceptReplica()

```
Function Cost(cont)

    Purpose: calculate cost function for cont contribution
    Returns: cost value, c [0,1]

    { cont: is the content contribution of this (or requesting) peer }
    { TARGET_CONT: default normalization value for content contribution }
    { SIGMA_COST: shape parameter }

1:  avg = IndexTable.GetAverageContribution()
2:  if avg > 1.5*TARGET_CONT then
3:      norm = avg
4:  else
5:      norm = TARGET_CONT
6:  end if
7:  a = (cont/norm)^SIGMA_COST
8:  c = 1 - a/(1+a)
```

Algorithm 12. Function Cost()

5.4 ENHANCED CONTRIBUTION METRIC

In Section 5.2.4 we defined node contribution, r_m, as the product of two functions, *fragment count*, $F(\chi^m)$ and *contribution gain*, $G(\chi^m,t,s,p)$, where χ^m is an indicator function (taking the value of one) for the fragments $f_{i,j}$ stored at node m. In addition, we defined two alternative formulations for $G(\chi^m,t,s,p)$. First, a unitary *contribution gain* (i.e., $G(\chi^m,t,s,p)=1$) and second, a *contribution gain* with time, size and popularity components. The purpose of our second formulation is to augment our incentives mechanism with the means to induce additional changes in user behavior.

The reasoning behind enhancing our contribution metric with time is that by rewarding nodes for the time they share content, the incentive mechanism can also promote lower churn rates, which would reduce the redundancy maintenance traffic. With the addition of size and popularity, we expect that by rewarding the nodes that store rare, or large, items, we could further reduce the system's redundancy maintenance load. We assume that repairing large files' fragments is more

6.0 EVALUATION

The structure and operation of the PR redundancy scheme and the incentive-based mechanism we propose in this dissertation share a simple design philosophy. Nonetheless, their interplay in a heterogeneous and dynamic P2P environment is too complex to be evaluated analytically. As an alternative, this chapter presents an experimental evaluation. The layout of this chapter is has follows. Section 6.1 presents the software platform used in this evaluation. Section 6.2 describes a base system configuration, against which the alternative system configurations can be compared. Section 6.3 discusses some of the metrics obtained in the experiments and describes the presentation of results. Section 6.4 examines the impact of different system variants on the content availability of the network. Section 6.5 presents the system's response using different settings for its redundancy and incentives components and Section 6.5.3.2 describes the capacity of the proposed mechanisms to adapt to different networking conditions, such as varying churn rate and content space size.

6.1 SIMULATION-EMULATION PLATFORM

Our redundancy-system has been implemented using Bamboo [21] and Modelnet [20]. Bamboo is our DHT application substrate and Modelnet is used to emulate wide area networking conditions using a cluster of computers. A description of Bamboo operation is presented in Section 2.2 and a

The incentives-based mechanism defines a mechanism for fair exchange of resources among members, which is independent of the redundancy maintenance process. That is, nodes are assigned resources in proportion to their current content participation in the system. Even if nodes decide not to participate anymore in the redundancy maintenance process, they get a fair share of resources. This feature is illustrated in Figure 43, which presents two experimental scenarios where 30% of the nodes do not participate actively in the redundancy maintenance process. In the first scenario (left graph), the non-compliant nodes (i.e., nodes that are not performing any redundancy maintenance activity) have a wide array of contribution values. Consequently, the transmission bandwidth they receive is also spread across the full spectrum of transmission bandwidths. In the second scenario (right graph), the non-compliant nodes have only minor content contributions. As a result, the transmission bandwidth they receive from other nodes is at the lower end of the spectrum. Thus, if we assume that the problem of free-riders in P2P networks is associated with nodes that do not contribute content, our incentive-based mechanism does a good job at limiting their impact on the performance of the system. We explore this feature further in the following section.

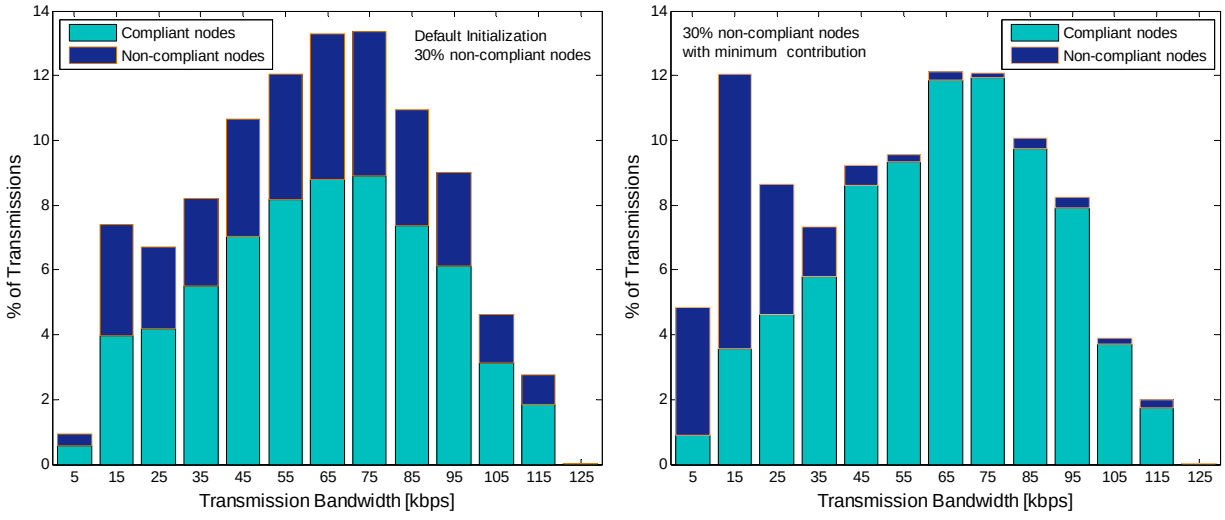


Figure 43. Fair Allocation of Resources in Incentive-based Mechanism

Gain2 and Gain4 schemes on the other hand, include a penalty component. Gain2 uses a unitary popularity gain though. For the size contribution gain, nodes storing less than 4MB/*k* receive a contribution gain penalty of 0.5, and as nodes increase their average fragment size contribution, they receive larger size contribution gains, up to 3.0 when their average fragment size reaches 12MB/*k*. In the Gain4 scheme, the popularity and size content gains are merged. Nodes receive a penalty when s^{p+p_0} is less than $s_{avg}^{1.6}$ (where s_{avg}=4MB/*k*) and a reward otherwise, up to a maximum size-penalty *contribution gain* of 3.0. We included a variant without popularity, Gain2, assuming that in practice measuring the popularity of individual items could be a complex task; especially in a dynamic distributed system. Thus, we include this variant to analyze the behavior of a *contribution gain* formulation that does not require any global knowledge to operate.

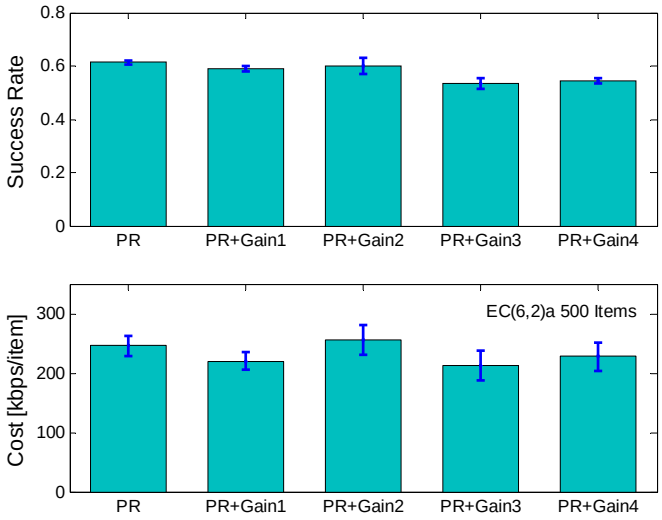


Figure 44. Cost and Efficiency for Contribution Gain Functions

Figure 44 presents the cost and efficiency metrics for the four simulations sets in Table 27 and for PR with a unitary *contribution gain*. Considering a confidence interval of 95%, the cost metrics obtained can not be differentiated. That is, the results obtained are statistically equivalent. For the efficiency metric, the differences between sets are minimal. Nonetheless, to determine

conclusively whether these *contribution gain* variants are equivalent or not, we analyze additional system properties.

The objective of the *contribution gain* variants is to modify a node's contribution metric. Thus, we present the effects of the different *contribution gain* variants on the RP-Cost and the TB-Utility functions.

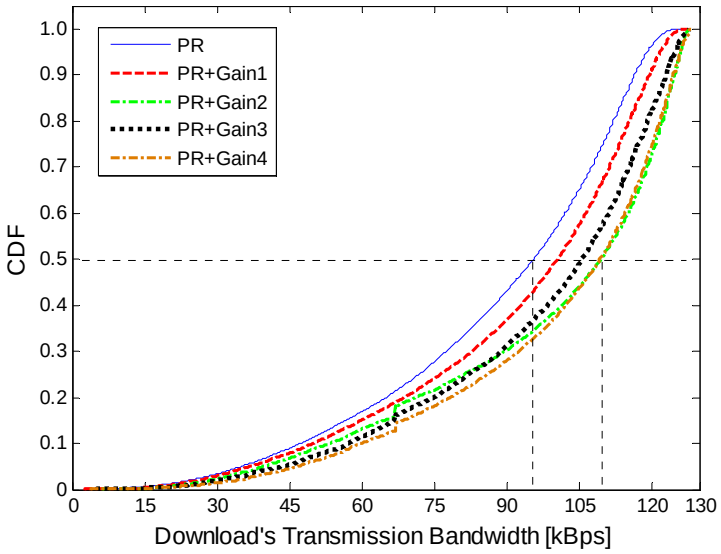


Figure 45. Download's Transmission Bandwidth CDF for Contribution Gain Functions

Figure 45 presents the CDF of the transmission bandwidths (i.e., TB-Utility function) that nodes receive for their download requests. PR, (with a unitary *contribution gain*) has the lowest median transmission bandwidth while Gain2 and Gain4 present the highest. Given that these *contribution gain* variants produce similar efficiency metrics, we consider that the best variant is the one resulting in better transmission bandwidths. Consequently, based on the experimental results obtained, we conclude that using an enhanced *contribution gain* function improves the system. All the *contribution gain* variants presented in Figure 45 improve the system's median transmission bandwidth with respect to PR.

6.6.4 Content Space Size

We mentioned earlier that the main scalability concern for content-sharing networks is the amount of bandwidth required to preserve content availability [10]. In that regard, the proposed mechanisms circumvents the redundancy maintenance problems of erasure coding by proactively replicating fragments. This way, the amount of information needed for a repair is significantly reduced. The next set of results presents the system's efficiency and cost metrics for various content space sizes.

With respect to efficiency, the system can sustain a good efficiency metric for up to 750 unique items, but its efficiency drops to 0.5 for 1500 items. With respect to cost, these results indicate that the system consumes less bandwidth per file in the larger content size experiments, which is important for scalability. However, in the 1,500 unique items experiment, this improvement in cost is not relevant since there is a serious efficiency loss. In addition, to properly establish the scalability of the redundancy system with respect to the content size, the units of the cost metric need to be kbps per node, rather than kbps per file. In that regard, we have included a third set of results in Figure 51 presenting this alternative cost metric. In this case, it is clear that for bigger content space sizes each node needs to commit additional bandwidth, but fortunately, this increment in bandwidth is sublinear with respect to the content space size. For instance, the cost for 750 items is 113.84 kbps per node, which represents an increase of 34.2% with respect to the cost of 84.83 kbps per node for the content space of 500 unique items. For the system configuration of 1,500 items, additional elements at play explain the abrupt drop in efficiency.

The decay in system efficiency for larger content sizes is not necessarily caused by a degradation of the redundancy repair mechanism. The initial content contribution of each node (described in Section 3.3.4) is performed in two stages. In the first stage, nodes receive complete

files, and in the second, nodes are assigned fragments. The number of items for which at least one node has a complete copy assigned is random, and the distribution of fragments among nodes is random as well. As a result, the redundancy-system might end up with insufficient resources for some items. In the different experiments conducted using a content space size of 500 unique items, the percentage of items with at least one complete copy assigned to nodes ranges between 0.5 and 0.67. We believe that this initial content distribution is a crucial factor for the efficiency of the system. In fact, this explains why none of the experimental setups surpasses the 69% efficiency mark.

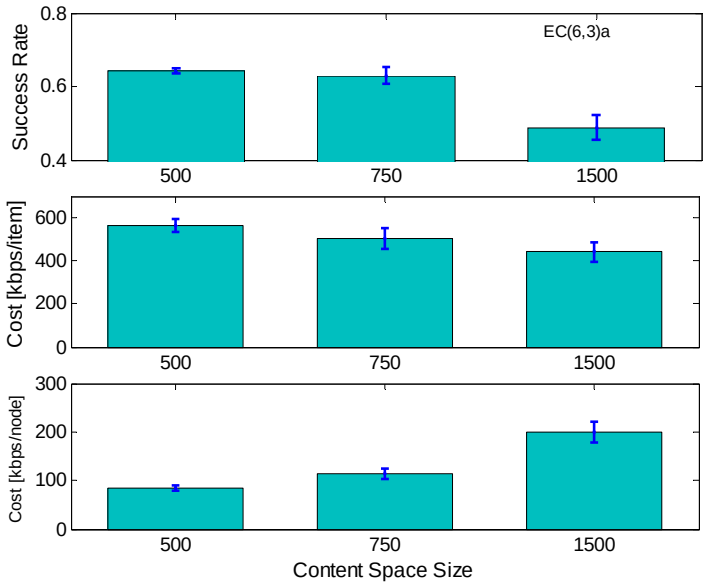


Figure 51. Cost and Efficiency for Different Content Space Sizes

Figure 52 presents a scatter diagram of the content availability of items throughout the simulation for the three experimental content space sizes mentioned earlier. For the 500 and 750 items configuration, there is little difference in the availability of contents. The availability of each item rarely drops below 0.8. However, for the 1,500 configuration, the graph illustrates the content availability problems in the system.

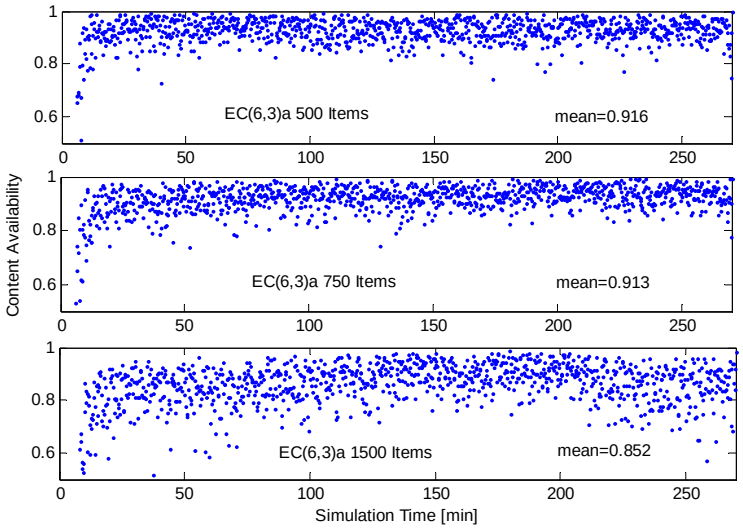


Figure 52. Content Availability for Different Content Space Sizes

APPENDIX A

TRANSIENT REMOVAL

The experimental portion of this work emulates a wide area P2P network using a cluster of computers interconnected by a switched LAN. Validation of the P2P routing substrate implementation is not required because the system uses exact copies of a deployed DHT application substrate. However, the price to be paid for using this architecture is time. Nodes execute a fully functional P2P routing stack in real time; thus, each experiment takes several hours to complete. In that regard, removing the transient behavior is critical not only for the accuracy of the measurements, but also for the cost of each experiment.

For the average session length of nodes, this work employs two mechanisms to reproduce the heterogeneous and unsynchronized behavior of nodes in a controlled environment. The first mechanism is the initialization process in our two-class node behavior model to recreate the heterogeneous distribution of user sessions seen in deployed systems. The second mechanism is pre-computing the initial session (or departure) for each node before starting the simulation. The second mechanism highly reduces the cost of each experiment, since other techniques to remove the transient component of the simulation require the system to be running for considerably longer intervals.

Overlay network size is the system property that better reflects transient behavior in an experiment. This metric depends on the number of simultaneously active nodes. That is, the superposition of the ON/OFF model of multiple independent nodes. Fast churning nodes become unsynchronized quickly, but nodes with longer session lengths (i.e., *Benefactors*) can take several hours to have their arrival and departures times unsynchronized. To minimize the length of this transient behavior, the initial session or offline interval of nodes is pre-computed according to the pseudo-code shown in Algorithm 13.

```
Procedure PerfecSimulation(N)

    Purpose: Pre-compute initial node state for simulations
    Returns: Tuple (state, length) for each node

    { N: Total number of nodes in simulation }
    { T: Time interval used to reach steady state }
    { lifetime[]: Array containing the cumulative ON/OFF intervals of each node }
    { state[]: Array containing ON/OFF state of each node }

 1: for i = 1 to N do
 2:     while lifetime[i] < T do
 3:         if state[i] = ON then
 4:             lifetime[i] += nextSession()
 5:             state[i] = OFF
 6:         else
 7:             lifetime += nextOffline()
 8:             state[i] = ON
 9:         end if
10:     done while
11:     length[i] = lifetime[i] – T
12: done for
```

Algorithm 13. Procedure PerfectSimulation()

The cost savings of this mechanism versus other initialization options are presented in Figure 53. The curve for the *perfect simulation* [94] mechanism reaches steady state in approximately fifteen minutes, while the other mechanisms reach steady state in no less than one hour. In the All-ON variant, every node starts online and the average overlay network size does not reach the steady state for three hours. This is due to the effect of long on-line sessions of the

benefactors user class. Conversely, when all nodes start in the offline state (All-OFF) or each node is selected at random to start either in the ON or OFF state (Random Start), the system reaches a steady state faster. Nonetheless, it is still about ninety minutes for both mechanisms, which is at least six times longer than using the *perfect simulation* technique.

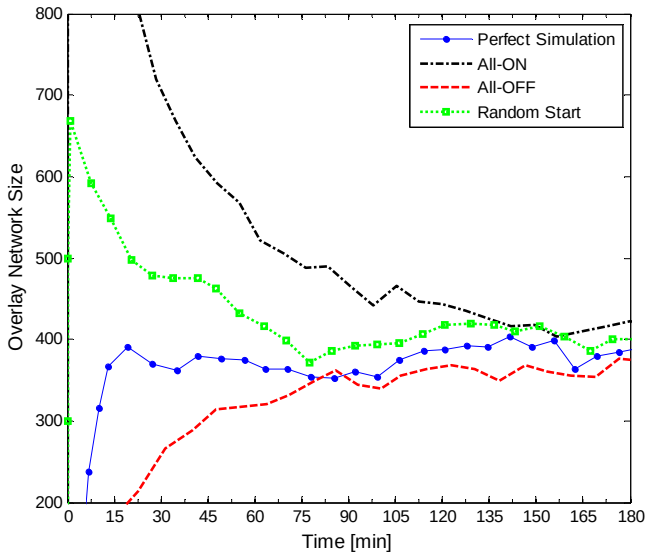


Figure 53. Alternative Node Initializations for Transient Removal

In all the measurements reported in the experimental section of this work, the values within the first fifteen minutes of each experiment are always excluded.

30. *Gnutella*. Wikipedia 20 July 2011 [cited; Available from: http://en.wikipedia.org/wiki/Gnutella#Software.

31. D. Stutzbach and R. Rejaie, *Understanding Churn in Peer-to-Peer Networks*, in *Technical Report*. 2005, University of Oregon.

32. Gummadi, K.P., et al., *Measurement, Modeling, and Analysis of Peer-to-Peer File-Sharing Workload*, in *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles* 2003, ACM.

33. Xin Bai A , Dan C. Marinescu A , and et. al., *A Macroeconomic Model for Resource Allocation in Large-Scale Distributed Systems.* Parallel and Distributed Computing, 2008.

34. Dabek, F., et al. *Wide-Area Cooperative Storage with CFS*. in *ACM SOSP* 2001.

35. S. Rhea, et al. *Pond: The OceanStore Prototype*. in *2nd USENIX conference on File and Storage Technologies (FAST)*. 2003.

36. Antony Rowstron and Peter Druschel. *Pastry: Scalable, Distributed Object Location And Routing For Large-Scale Peer-To-Peer Systems*. in *IFIP/ACM Middleware*. 2001. Heidelberg, Germany: ACM

37. Androutsellis-Theotokis, S. and D. Spinellis, *A survey of peer-to-peer content distribution technologies.* ACM Computer Surveys, 2004. **36**(4): p. 335-371.

38. Gummadi and et al. *The Impact of DHT Routing Geometry on Resilience and Proximity*. in *ACM SIGCOMM*. 2003.

39. Octavio Herrera-Ruiz and T. Znati. *Static Resiliency vs Churn-Resistance Capability of DHT-Protocols*. in *ISCA PDCS* 2005. Las Vegas, Nevada.

40. Naicken, S., et al., *The State of Peer-to-Peer Simulators and Simulations* ACM Computer Communications Review, 2007. **37**(2): p. 95-98.

41. Z. Yao, et al. *Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks*. in *IEEE ICNP*. 2006.

42. X. Wang, et al., *Robust Lifetime Measurement in Large-Scale P2P Systems with Non-Stationary Arrivals*, in *IEEE P2P*. 2009.

43. F. E. Bustamante and Y. Qiao, *Friendships that last: Peer lifespan and its role in P2P protocols*, in *International Workshop on Web Content Caching and Distribution*. 2003.

44. X. Wang, Z. Yao, and D. Loguinov, *Residual-Based Estimation of Peer and Link Lifetimes in P2P Networks.* IEEE/ACM Transactions on Networking, 2009. **17**(no. 3).

45. D. Stutzbach and R. Rejaie. *Understanding Churn in peer-to-peer networks*. in *ACM Internet Measurement Conf. (IMC)*. 2006.

46. M. Steiner, T. En-Najjary, and E. W. Biersack. *A Global View of KAD*. in *7th ACM Internet Measurement, IMC'07*. 2007. San Diego, California, USA.

47. Qiuming Luo, et al., *A Novel Model and a Simulation Tool for Churn of P2P Network*, in *Parallel and Distributed Computing, Applications and Technologies*. 2010.

48. Octavio Herrera and T. Znati, *Modeling Churn in P2P Networks*, in *40th Annual Simulation Symposium*. 2007.

49. Enrique Fernández-Casado, Marc Sánchez-Artigas, and P. García-López, *Affluenza: Towards Universal Churn Generation*, in *P2P*. 2010.

50. Boxun Zhang, Alexandru Iosup, and D. Epema, *The Peer-to-Peer Trace Archive*, in *Parallel and Distributed Systems Report Series*. 2010, Delft University of Technology.

51. Godfrey, B. *Repository of Availability Traces*. 2010 [cited; Available from: http://www.cs.illinois.edu/~pbg/availability/.

52. Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker, *Understanding Availability*, in *IPTPS*. 2003.

53. AG Dimakis, K Ramchandran, and Y Wu and C Suh. *A Survey on Network Codes for Distributed Storage*. in *IEEE Surveys*. 2011.

54. W. K. Lin, D.M. Chiu, and Y. B. Lee, *Erasure Code Replication Revisited*, in *IEEE P2P*. 2004.

55. Ranjita Bhagwan, Stefan Savage, and Geofrey M. Voelker, *Replication Strategies for Highly Available Peer-to-Peer Storage Systems*, in *Tech Report*. 2002, UC San Diego.

56. Alexandros G. Dimakis, et al., *Network coding for Distributed storage Systems*, in *INFOCOM*. 2007: Anchorage, Alaska.

57. K. Rashmi, N. B. Shah, and P.V. Kumar, *Optimal Exact-Regeneration Codes for Distributed Storage at teh MSR and MBR Points via Product-Matrix Construction.* IEEE Trans. Information Theory, 2010.

58. Alessandro Dominuco and E. Biersack, *Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems.* in Proceedings of Peer-to-Peer Computing, 2008: p. 89-98.

59. Chris Williams, et al. *Redundancy Management for P2P Storage*. in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid* 2007.

60. Fan Wu, Tongqing Qiu Yuequan Chen, and Guihai Chen, *Redundancy Schemes for High Availability in DHTs*, in *IPSA 2005*. 2005.

61. Guangping Xu, Gang Wang, and J. Liu, *A hybrid redundancy approach for data availability in structured P2P network systems*, in *International Symposium on Pacific Dependable Computing*. 2007, IEEE.

62. Nishida, H. and T. Nguyen, *A Global Contribution Approach to Maintain Fairness in P2P Networks.* IEEE Transactions on Parallel and Distributed Systems, 2010. **21**(6): p. 812-826.

63. Rameez Rahman, et al. *Improving efficiency and fairness in p2p systems with effort-based incentives* in *ICC*. 2010.

64. Tsuen-Wan Johnny Ngan , et al., *On Designing Incentives-Compatible Peer-to-Peer Systems* in *2nd Bertinoro Workshop on Future Directions in Distributed Computing (FuDiCo II: S.O.S.)*. 2004: Bertinoro, Italy.

65. Anwitaman Datta and Karl Aberer. *Internet-Scale Storage Systems under Churn – A Study of the Steady-State using Markov Models*. in *Sixth IEEE International Conference on Peer-to-Peer Computing*. 2006.

66. Emil Sit, et al., *Proactive Replication for Data Durability*, in *5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*. 2006.

67. Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. *A Measurement Study of Peer-to-Peer File Sharing Systems*. in *Multimedia Computing and Networking (MMCN)*. 2002.

68. Shanyu Zhao, Daniel Stutzbach, and Reza Rejaie. *Characterizing Files in the Modern Gnutella Network: A Measurement Study*. in *SPIE/ACM Multimedia Computing and Networking*. 2006. San Jose, CA.

69. *Inet topology generator v 3.0*. [cited; Available from: http://topology.eecs.umich.edu/inet/.

70. Mauro Andreolini and Riccardo Lancellotti. *Analysis of peer-to-peer systems: workload characterization and effects on traffic cacheability*. in *12th Annual Meeting of the IEEE / ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004)* 2004. Volendam (NL).

71. J. Li, et al., *A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs under Churn*, in *INFOCOM*. 2005: Miami, FL.

72. Subhabrata Sen and Jia Wong, *Analyzing peer-to-peer traffic across large networks.* IEEE/ACM Transactions on Networking (TON), 2004.

73. R. Bolla, et al., *A measurement study supporting P2P file-sharing community models.* Computer Networks, 2009(53): p. 485-500.

74. Moritz Steiner, Taoufik En-Najjary, and E.W. Biersack, *Analyzing Peer Behavior in KAD*, in *Research Report*. 2007, Institut Eurecom: Sophia-Antipolis, France.

75. Jacky C. Chu, Kevin S. Labonte, and Brian N. Levine, *Availability and locality measurements of peer-to-peer file systems*, in *SPIE* 2002.

76. S. Saroiu, P. K. Gummadi, and S. D. Gribble. *A Measurement Study of Peer-to-Peer File Sharing Systems*. in *Multimedia Computing and Networking (MMCN)*. 2002. San Jose, CA.

77. Zhonghong Ou, E. Harjula, and M. Ylianttila, *Effects of Different Churn Models on the Performance of Structured Peer-to-Peer Networks*, in *Personal, Indoor and Mobile Radio Communications*. 2009.

78. Dimakis, A.G., P. B. Godfrey, and Y. Wu. *Network Coding for Distributed Storage Systems*. in *Information Theory*. 2010.

79. K. V. Rashmi, et al., *Optimal Exact-Regeneration Codes for Distributed Storage at the MSR and MBR Points via Product-Matrix Construction.* IEEE Trans. Information Theory, 2010.

80. Kuo, W. and M. J. Zuo, *Thek-out-of-n System Model*, in *Optimal Reliability Modeling: Principles and Applications*. 2003, Wiley. p. 231-280.

81. K. V. Rashmi, et al. *Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage*. in *Proc. Allerton Conf., Urbana-Champaign*. 2009. San Diego.

82. Alexandros G. Dimakis, et al. *A Survey on Network Codes for Distributed Storage*. in *IEEE Surveys*. 2011.

83. Pinheiro, E., W.-D. Weber, and L.A.e. Barroso, *Failure Trends in a Large Disk Drive Population.* USENIX Conference on File and Storage Technologies, 2007.

84. *Hard drive manufacturer specifications*. 2011 [cited; Available from: http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701220.pdf.

85. Lehpamer, H., *Transmission Networks Fundamentals*, in *Microwave Transmission Networks, Second Edition*. 2010, McGrawHill. p. 15.

86. Charles Blake and Rodrigo Rodriguez, *High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two*, in *HotOS IX*. 2003.

87. Hardin, G., *The Tragedy of the Commons*, in *Science*. 1968. p. 1243-1248.

88. *Tragedy of the commons*. 2011 [cited; Available from: http://en.wikipedia.org/wiki/Tragedy_of_the_commons.

89. S. B. Handurukande, et al., *Peer Sharing Behavior in the eDonkey network, and implications for the design of server-less file sharing systems.* SIGOPS Oper. Sys. Rev., 2006. **40**: p. 359-371.

90. Deterding, S. *Meaningful Play: Getting Gamification Right* Google Tech Talk [Webcast] Jan 24, 2011 [cited 2011; Available from: http://www.youtube.com/watch?v=7ZGCPap7GkY&feature=relmfu.

91. *Wikileaks*. [cited; Available from: wikileaks.org.

92. Schwarz, T., et al., *Disk Failure Investigations at the Internet Archive*, in *NASA/IEEE Conference on Mass Storage Systems and Technologies*. 2006.

93. Zghaibeh, M. and K.G. Anagnostakis, *On the Impact of P2P Incentive Mechanisms On User Behavior*, in *NetEcon+IBC*. 2007: San Diego.

94. Mecke, K.R. and D. Stoyan, *A Primer in Perfect Simulation.* Springer Lecture Notes in Physics, 2000: p. 349 - 378.