Building a Reliable P2P System Out of Unreliable P2P Clients: The Case of KAD

Damiano Carra, Ernst W. Biersack
Institut Eurecom
Sophia-Antipolis, France
{carra,erbi}@eurecom.fr

ABSTRACT

Distributed Hash Tables (DHT) provide a framework for managing information in a large distributed network of nodes. One of the main challenges DHT systems must face is node churn, i.e., nodes can arrive and depart at any time. To assure that information published in a DHT remains available despite node churn is equivalent to building a reliable system out of unreliable components.

In this paper we analyze KAD, a widely deployed DHT system. We focus on how to assure that information published in KAD remains available despite churn. We apply reliability theory to understand how the probability of finding an object published in KAD evolves over time. We also evaluate the cost, in terms of message exchanges, associated with the publishing scheme.

Once we have identified the main weaknesses of the existing system, we propose an improved publishing scheme that is able to assure the same reliability but at a dramatically reduced cost. By exploiting knowledge about the lifetime of the nodes used to store the information, it is possible to reduce the publishing cost by one order of magnitude.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms

Algorithms, Design, Performance

Keywords

Peer-to-Peer, Performance Analysis, KAD, Inspection Policy

1. INTRODUCTION

In the last years, Peer-to-Peer (P2P) file sharing has become increasingly popular, with millions of users sharing contents and resources. One of the main problems associated to a distributed environment is to efficiently find information. Distributed Hash Tables (DHTs) organize the overlay in a structured way and map nodes and objects in this structure so that information can be found easily. There are many possible structures and publishing schemes, which depend on the specific application requirements: the content can be made available forever after it is published (independently from the presence of the publishing node) or it can be just announced to the system and maintained by the publishing node. In the first case the responsibility of the object is transferred to the DHT network, in the second case only references are published while the responsibility is kept by the publishing node. KAD, the DHT routing protocol we consider in this paper, adopts the latter scheme.

A major issue in P2P networks is *churn*, i.e. node arrivals and departures that make the system unreliable. Many techniques have been developed to handle churn so that the content (or references) can be made available independently from node dynamics. Nevertheless, only some of them have been deployed, thus it is hard to validate the effectiveness of these solutions.

One exception is KAD, a widely deployed DHT system with millions of users [1]. In KAD, nodes publish at regular intervals multiple copies of the references to the objects they own. This publishing policy generates a high number of messages: recent measurements [2] show that the traffic volume required for publishing is 100 times the traffic for searching objects. The same study reveals that the estimated probability to find the published reference is very high. Despite the success of KAD, we are not aware of any study that provides an exact characterization of the cost and the performance of KAD publishing scheme.

In this paper we analyze the availability of content published on KAD. We propose a model, based on reliability theory, that is able to evaluate the probability of finding an object in any instant t after it is published, along with the cost of publishing. The model considers the particular publishing scheme – two-level scheme – adopted by KAD. With this model, we can assess the solution adopted by KAD for handling churn and identify the weaknesses of its publishing scheme. We then propose an improved publishing scheme that is able to guarantee a given availability, with a decreased cost compared with the basic scheme.

Our analytical model for the reliability is compared with trace-driven simulations, based on *real* traces collected over almost six months by crawling the KAD network [2]. The results show that our model is able to predict the performance of the basic publishing scheme, as well as of the improved scheme. The improved scheme, tested on the same traces, is able to reduce the number of messages by one order of magnitude with respect to the current implementation of KAD.

After providing some background on KAD and reliability theory in Sect. 3, we analyze the reliability offered by the current publishing scheme of KAD in Sect. 4. In Sect. 5 we analyze the shortcomings of the basic scheme and we propose our improved publishing scheme. We show some numerical examples in Sect. 6 and we conclude the paper in Sect. 7.

2. RELATED WORK

KAD represents one of the largest DHT systems with millions of users. Despite its success, very few papers so far have evaluated the performance of the protocol, and none of them has considered the performance of its publishing scheme. In [3], the authors analyze the performance of the DHT lookup phase, focusing on the routing table building process. In [4], the same authors provide a set of measurements on different characteristics of the system, e.g. node availability; the measurements represent the output of their work, while we use measurements as a starting point. Qiao et al. [5] analyze through measurements the query resolution time, i.e., how long it takes to find a key. In this work, we characterize the performance of the protocol in terms of the *probability* to find objects, without considering the delay necessary to retrieve them.

In particular, we focus on the reliability of the publishing system, i.e. the probability that in the system there is a sufficient number of online nodes to assure *content availability*. P2P applications span from file sharing to distributed storage, to streaming systems. Each application has its own requirements for content availability and many solutions have been proposed for the different applications. Here we focus on file sharing applications, which is the target of the KAD.

One of the main problems related to studying the availability of a system is to first determine the availability of the individual nodes. Many works assume an exponential uptime distribution [6][7]. In this paper, instead, we use the results of a six month measurement study of KAD [2], that allows us to characterize more precisely the availability of the nodes, and thus the availability of the content.

Most of the studies on content availability in DHTs under churn mainly use simulation as the primary means of investigation. In [8], authors analyze different techniques (e.g., reactive or periodic recovery), considering the delay of the search as the main performance metric. Bhagwan et al. [9] analyze node availability in Overnet, but without characterizing content availability. Dunn et al. [10] analyze the service availability, instead of simple node availability, but for a general case, without considering the two-level publishing scheme adopted by KAD (see Sect. 3.1 for details).

To the best of our knowledge, no analytical study on content availability over time knowing the node uptime distribution have been done considering KAD DHT system. The closest work to ours is [11], where authors analyze not only the lookup performance, but also the key replication scheme. Given the node uptime distribution, the replication factor, and the republishing rate, they provide a closed form expression for the expected lifetime of keys. In our work, instead, we provide the probability over time to find a key, not only its expectation, thus our results are more general and provide more insights on the dynamical behavior of the system. Moreover, we model in detail the two-level publishing scheme of KAD, while [11] assumes a one-level scheme, without properly modeling a real protocol.

The availability of the content is considered also in different contexts. In [12] authors study the maintenance of soft state between a sender and a receiver through signalling. Duvvuri et al. [13] analyze the optimal lease for cache consistency. Both papers take into account the periodical check of stored information, but they focus on a single instance, while in our work we take into account multiple copies. In [14] authors consider the placement of different copies of web documents, but they consider strategies based on heuristics, without taking into account the availability in terms of probability to find the documents.

3. BACKGROUND

In this section we describe the publishing scheme implemented by KAD system and summarize some basic concepts related to reliability theory[1]; we also introduce the terminology used throughout the paper.

3.1 The Kademlia DHT System

KAD is a DHT protocol that is based on the Kademlia framework [1][2][3]. Like most DHTs, peers and ob-

[1]Here we only give a brief overview of reliability concepts: for more details, the interested reader is referred to [15].

jects in KAD have an unique identifier called *key*, which is 128 bit long, randomly assigned using a hash function. The distance between two entities – peers, objects – is defined through the bitwise XOR of the keys.

The basic operations performed by KAD peers are (i) building a routing table, (ii) performing a lookup and (iii) publishing objects. Routing tables are used by peers to maintain active contacts with a set of neighbors so that operations like searching or publishing can be performed. Lookup is used when a peer wants to look for objects, if it is searching, or to look for other peers, if it is publishing. Objects in KAD are searched sending out multiple requests, each of them iteratively goes closer and closer to the target. The interested reader can find details related to routing in KAD in [3]. The routing table building process and the routing procedures do not impact the analysis we propose in this paper. Instead, we focus on the publishing procedure that represents the primary aim of our study.

Each node in the KAD network is responsible for the objects it owns and it lets the other nodes know these objects by publishing **references**. A reference contains information about the objects and about the node that owns the objects, so that interested nodes can find them. If the node that owns the objects disappears, the objects disappear too, but the references may still exist in the network, in which case they become *stale*. KAD implements a two-level publishing scheme that decreases the number of messages necessary to publish references (for an analysis of the benefits of such a scheme, see [16]). A reference to an object comprises a **source** and W **keywords**:

- The source, whose key is obtained by hashing the content of the object, contains information about the object and the pointer to the publishing node;
- Keywords, whose keys are obtained by hashing the individual keywords of the object name, contain (some) information about the object and the pointer to the source.

Hereinafter, we will refer to source and keywords considering the corresponding keys. We call **publishing node** the node that owns an object and **host nodes** the nodes that have a reference to that object. When a node wants to look for an object, it first searches for the keywords and does a lookup to obtain all the pointers to different sources that contain these keywords. It then selects the source it is interested, looks up that source to obtain the information necessary to reach the publishing node. Figure 1 shows an example of the reference published by a node to an object *Obj* that has 2 keywords, k_1 and k_2, and the pointers each key has.

Since references are stored on nodes that can disappear at any point in time, the publishing node publishes *multiple copies* of each reference – source and keywords.

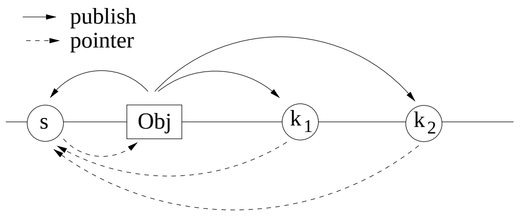


Figure 1: Example of 2-level publish scheme adopted by KAD.

The keys are published on host nodes whose ID agrees in the first 8 bits with the key of the references. This means that the host node is not necessarily the closest node to the key. Experimental results [3] showed that, in general, all the copies are stored in host nodes very close to the key, since routing tables are detailed and the publishing node is able to find many possible host nodes in the zone of interest.

An **expiration time** is associated to each reference, after which the information on the host node are removed. At the same time, the publishing node (if it is still online) republishes the keywords and the source independently from where it published them previously, so the information may be stored in different nodes. For a source the expiration time is 5 hours. For keywords, the expiration time depends on the load of the host node. The **load** is defined as the number of references for that keyword the host node has divided by the maximum number of references it can hold. When the load is below 20%, the expiration time is set to the default value of 24 hours.

3.2 Reliability theory

Reliability theory is a well-established field of research that studies the reliability of systems composed by units, each of them with its own reliability. The primary aim of the reliability theory is to propose methods to analyze and increase the reliability by identifying the units and their role in the system [15].

The **uptime** T of a unit is a random variable, with a corresponding **uptime distribution**, $F(t) = P[T \leq t]$, that denotes the probability that the uptime is less than t or, alternatively, the unit fails within the interval $(0, t]$. From this distribution we can derive the **reliability function** of a unit as $R(t) = 1 - F(t) = P[T \geq t]$, i.e., $R(t)$ represents the probability that the uptime is greater than t.

Another important measure is the probability that the unit will fail in the interval $(t, t + \Delta t]$ given that the unit is still functioning at time t. By dividing this probability by the interval Δt and letting $\Delta t \to 0$, we obtain the **failure rate** (also known as **hazard rate**) $\mu(t) = -\frac{dR(t)}{dt}\frac{1}{R(t)}$.

The failure rate describes the propensity to failures of the unit. An **increasing failure rate** (IFR) with t, called *aging effect*, represents units that deteriorate with time. A **decreasing failure rate** (DFR) is typical for system whose reliability grows over time.

The definition of the reliability for a single unit can be extended to systems composed of different independent units. The two basic structures are **series** and **parallel** of units[2]. The uptime of a series of n units is given by the minimum uptime among all the units; consequently, the reliability of the system, denoted by $R_\wedge(t)$, can be expressed using the reliability of its units $R_i(t)$:

$$\begin{aligned} R_\wedge(t) &= R_1(t)R_2(t)...R_n(t) \\ &= R^n(t) \end{aligned} \tag{1}$$

where the last equality is true for units with identical reliability function, i.e., $R_i(t) = R(t), \forall i$. Conversely, the uptime of a parallel of n units is given by the maximum uptime among all the units, and the reliability of the system, denoted by $R_\vee(t)$, can be expressed as:

$$\begin{aligned} R_\vee(t) &= 1-(1-R_1(t))(1-R_2(t))...(1-R_n(t)) \\ &= 1-(1-R(t))^n \end{aligned} \tag{2}$$

where the last equality is true if $R_i(t) = R(t), \forall i$. More complex systems can be described as a compositions of series and parallel structures.

Up to this point we have implicitly assumed that units cannot be replaced, i.e. when they fail the system deteriorates. We can consider repairable systems where maintenance or replacement is carried out. There are different ways to repair/substitute units of a system: preventive maintenance [18], maintenance upon failure [19], or periodic testing [20], to cite some of them.

What concerns our work, we consider **periodic replacement** (PR), where *all* units are regularly replaced with new units; the replacement intervals τ_i are simply regular intervals, i.e., $\tau_i = i\tau$, with τ constant and $i = 1, 2, \ldots$

From the reliability viewpoint, it is simple to show that at the beginning of each period we restore the initial condition, so $R(t)$ is equal to the non-repairable system in the interval $(0, \tau]$, and then it repeats periodically.

4. RELIABILITY ANALYSIS OF KAD

4.1 General Assumptions

The publishing procedure on KAD comprises different phases: the publishing node first has to retrieve a list of possible host nodes whose key agrees in the first 8 bits with the key of the references, then it tries to publish on that host nodes. Looking for host nodes has a cost, both in term of routing messages and in term of

[2]Another important structure is *k out of n*, which we do not consider as it is not needed to describe KAD.

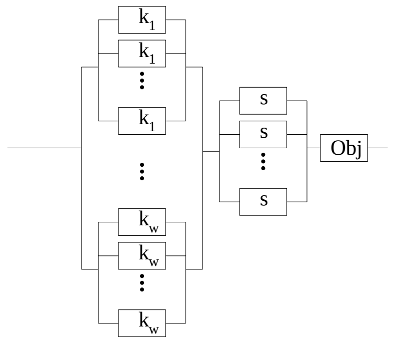


Figure 2: System model

delay. We assume that the delay is negligible with respect to the republishing interval, so that all the replicas are available starting from the same instant.

From viewpoint of the searching node, there are mainly two issues that give origin to cost. First, the searching node has to find the host nodes with the replicas. Host nodes are not necessarily the closest nodes to the key, not only because of the selection policy of the publishing node, but also because, as the time goes by, new nodes with an ID closer to the key than existing host nodes may arrive. This result in an increased delay if the searching node wants all the possible answers. Second, when the searching node asks for a reference (e.g., a keyword) to a host node, if the host node has multiple entries for that reference, it replies with a (random) subset of entries (up to 300). Even if the searching node is able to find a host node, it may not receive the exact reference it is looking for.

In this paper we focus on rare objects and we assume that the main performance index is the probability to find the objects, without taking into account the delay needed for searching them. This means that the searching node knwos all the keywords associated to an object and is always able to find all the available replicas (e.g. through multiple searches in the zone where the references are). We also assume that the load of the host nodes is always below 20%, so that the republishing delay for keywords is 24 hours. The results we obtain can then be considered as performance bounds for an ideal system. We leave for future work the exact characterization of the impact of routing and delay on the performance.

4.2 Model Description

We consider the publishing system as composed by units represented by nodes. The object is found if at least a subset of the units are working, i.e., host node are online so that they can provide the information previously published. For each key of the two-level publishing scheme – source and keywords – KAD creates multiple copies. In order to reach an object, a node must be able to successfully reach (retrieve) at least

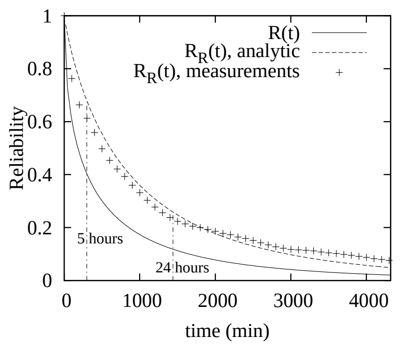
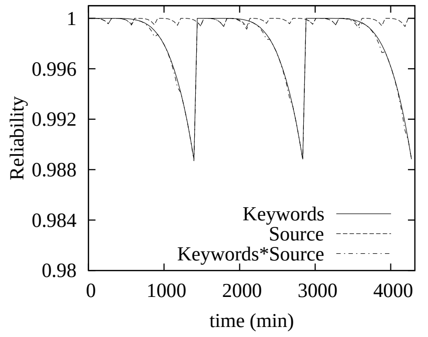
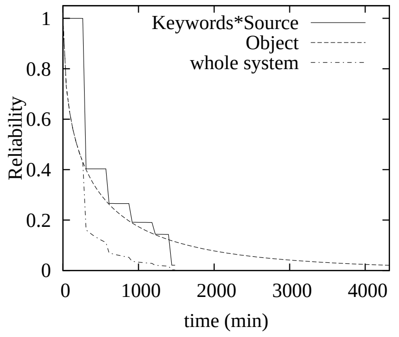


Figure 3: Reliability function for Weibull distributed uptime.



(a) Keywords and source



(b) Whole system

Figure 4: Reliability analysis of KAD (2 keywords, 1 source, $C_s = C_k = 10$, $1/f_s = 5$h, $1/f_k = 24$h).

one copy of:

- one of the keywords;
- the source;
- the object.

The whole system can be modeled as shown in Fig. 2. This system is composed by a series of two parallel structures that represent the keywords and the source and a single unit that represents the publishing node (the owner of the object).

Given the uptime distribution of the nodes (units of the system), we are able to compute the reliability function over time for an object, i.e., the probability that a node, looking for an object, is able to find it. Extensive measurements on the KAD system [2] revealed that the uptime distribution follows a Weibull law, with scale parameter and shape parameter approximately equal to 357.7 and 0.545 respectively. The shape parameter is less than 1, which means that the corresponding reliability function has Decreasing Failure Rate (DFR) property. It is important to note that, when a node publishes a reference, it stores keywords and the source on host nodes that are *online*: this means that we need to consider the *residual* uptime of the host nodes. When the arrival process is stationary, the **residual node uptime** $F_R(t)$ is given by Smith's formula [17]:

$$F_R(t) = \frac{1}{\mathrm{E}[T]} \int_0^t (1 - F(s))ds \tag{3}$$

where $\mathrm{E}[T]$ is the mean uptime and $F(t)$ is the uptime distribution. The residual reliability function $R_R(t)$ is then given by

$$R_R(t) = 1 - \frac{1}{\mathrm{E}[T]} \int_0^t R(s)ds \tag{4}$$

When $F(t)$ is Weibull distributed, the integral can only be solved numerically and the two distributions, $R(t)$ and $R_R(t)$ are shown in Fig. 3. We see that $R_R(t) > R(t)$, i.e. that residual uptime is increased: this phenomenon, observed in many previous measurement studies, is due to the high probability to find online host nodes with uptime higher than average uptime of nodes. It is possible to show that $R_R(t)$ maintains the DFR property [21]. Figure 3 also shows the measurements of the residual lifetime of nodes on the KAD system from [2]. The slight deviation with respect to the reliability function predicted by (4) is due to the fact that in KAD the node sampling process is periodical, and this introduces a correlation in the selected nodes. In any case, the measurements confirm the increased reliability when the residual uptime is taken into account.

4.3 Reliability of the Publishing System

The system model shown in Fig. 2 is composed of a series of three subsystems – keywords, source and owner of the object– so the reliability function is obtained as the product of reliability functions of these three subsystems. The first two subsystems are parallel units, and each unit (with different rates for keywords and sources) is restored periodically, provided that the publishing node is still online.

Let W be the number of keywords of the object, C_s and C_k be the number of copies (replicas) for keywords and sources respectively, and f_s and f_k be the corresponding publishing frequency (per unit of time, e.g. per day). KAD clients use a default value of 10 replicas for both C_s and C_k and republish sources and keywords every 5 and 24 hours respectively.

Figure 4(a) shows the reliability function for the source, the keywords, and for the composition of the two subsystems, obtained from the model. We used $W = 2$ keywords and default values for the number of replicas and the republishing frequencies.

The reliability function of the source is obtained starting from the residual uptime distribution $R_R(t)$ reported in (4). Using Eq. (2), we compute the reliability function $R_{R\vee}(t)$ considering the C_s replicas; then, we periodically repeat $R_{R\vee}(t)$ with a period $1/f_s$. The reliability function of the keywords is similarly obtained considering WC_k replicas and a period $1/f_k$. The periodic reliability functions for source and keywords are then combined according to Eq. (1).

In our model, we assume that the host node, when it goes offline, deletes all the references. KAD instead keeps memory of the references, so that when a node comes back online it checks if the previously stored references are still valid and keeps them. Thus, the results of our reliability analysis can be considered as a lower bound. We leave for future work the exact characterization or the reliability considering multiple session of the same node within the expiration time of its references. In any case, the cost associated to publications remains the same, even if the reliability may increase. This holds also for the improved scheme we propose in Sect. 5.

Note that the global reliability is dominated by the reliability of the keywords, so it may happen that one copy of the source is present but no keyword points to it. In Fig. 4(a) the references are always restored periodically, but this process can be done only if the publishing node is online. In order to take into account this fact, each interval should be multiplied by the probability that at the beginning of the interval the node is still online. This result is shown in Fig. 4(b) (note the different scale of the Y axis), along with the reliability of the publishing node and the final reliability of the system. It is clear that the final reliability depends strongly on the reliability of the publishing node. Since the aim of our study is to understand the impact of design parameters – such as the number of copies or the republishing frequency – hereinafter we will consider a publishing node always online (since it hosts the actual content it publishes), i.e. we will consider only the reliability of the keywords and sources, as shown in Figure 4(a).

From the analysis of the reliability, it is possible to see that the publishing scheme of KAD offers an high probability of finding objects, despite the dynamics of the system. However, as we will see, this good performance comes at the cost of a high traffic load generated by the publishing procedure.

4.4 Cost analysis

KAD implements a periodic replacement (PR) policy, so the cost of publishing can be computed very easily. Let S_s and S_k be the size of the source message and the keyword message respectively. We consider only the publishing messages, without taking into account the *route messages* needed to find the host nodes. Assuming a mean number of keywords W for the file name, the traffic generated using PR policy, B_{PR}, is

$$B_{\mathrm{PR}} = S_s C_s f_s + W S_k C_k f_k \tag{5}$$

This represents the cost for a node that is always online. In case of a node that disappears and reappears, the cost should be multiplied by the mean uptime per unit of time. In order to be able to compare the cost with other publishing schemes, we assume an always online node.

Recent measurements [2], analyzing 1/256 of the key space for 12 hours, observed approximately 5.5 million publishing messages, for a total volume of almost 1 GByte of traffic, for a mean number of 6500 nodes. The same study showed the traffic load generated by search messages is 10 MByte. It is clear that the cost of publishing represents a major issue for KAD system.

5. IMPROVING PUBLISHING SCHEME (AT AFFORDABLE COST)

Analyzing the publishing procedure using reliability theory, it is possible to identify the weaknesses of the scheme. For instance, the publishing node republishes the references at regular intervals independently from the fact that the previous host nodes could still be online. A straightforward improvement could be adopting, instead of periodic replacement, a *periodic inspection* with replacement upon failure (PI) policy: in this case we decrease the traffic since the cost of inspection should be lower than republishing. The analysis of PI policy is simple and we do not include it in this paper. Instead we introduce a more advanced inspection technique called Quantile Based Inspection (QBI) that makes use of the information about the past of the host nodes to predict their reliability.

5.1 Quantile Based Inspection

Instead of *periodically* republishing data, nodes can keep track of the host nodes where they previously published and simply refresh the expiration time for the references. The history of the host nodes (seen by the publishing node when it refreshes the references) can be exploited to *dynamically adapt* the expiration time: since the uptime distribution has DFR property[3], the longer the node stays online, the smaller the probability that it will leave, the higher its reliability. This means that, given a target reliability, we can change the inspection rate if the references are published on stable hosts.

In reliability theory, this policy is known as *quantile-based inspection* (QBI(α), where α is the target reliability [19]). With QBI, the information about the history

[3]Studies on other P2P systems found an uptime distribution according to the Pareto distribution: it is interesting to note that Pareto distribution has DFR property too.

is included in the **conditional reliability** $R_{R|\tau}(t)$:

$$R_{R|\tau}(t) = P[T > t+\tau | T > \tau] = \frac{P[T > t+\tau]}{P[T > \tau]} = \frac{R_R(t+\tau)}{R_R(\tau)} \quad (6)$$

where $R_R(t)$ is the residual reliability function of the unit. The inspection intervals are given by:

$$\begin{cases} \tau_1(\alpha) = \sup\{t > 0 : R_R(t) \geq \alpha\} \\ \tau_n(\alpha) = \sup\{t > 0 : R_{R|\tau_{n-1}}(t) \geq \alpha\} \end{cases} \quad (7)$$

In case of DFR, it is easy to show that $R_{R|\tau_n}(t) > R_{R|\tau_{n-1}}(t), \forall n$. This implies that $\tau_n > \tau_{n-1}$ for a given α, so the inspection interval increases, decreasing the cost per unit of time of the inspection policy. Table 1 shows the sequence of the first five inspection intervals τ_i for different target reliability α using the residual reliability function (4). The sequence of inspection intervals for a given α is uniquely determined and it can be precomputed. Nevertheless, as we will see, the performance can be further increased introducing a bit of randomness in the exact values of τ_i: in this case the values shown in Table 1 represent mean interval lengths.

Table 1: First five inspection intervals τ_i for QBI policy computed with (4) (min.)

α	τ_1	τ_2	τ_3	τ_4	τ_5
0.9	40	90	155	230	310
0.6	270	735	1695	2965	4880
0.3	1000	4205	8685	14395	19590

The general formula to find the reliability over time of a unit using QBI policy is given by (see [20])

$$R_{QBI}(t,\alpha) = R_R(t) + \quad (8)$$
$$+ \sum_{k=1}^{\log R_R(t)/\log\alpha} R_R\left(t - R_R^{-1}(\alpha^k)\right)\left[\alpha^{k-1} - \alpha^k\right]$$

where $R_R^{-1}(t)$ is the inverse of the residual reliability function. It is important to note that, independently from the shape of the reliability function, in any point of time the minimum possible value is α.

There could be the case that, as the inspection time increases, the host node goes offline and then comes back online before the next inspection. In order to avoid a wrong estimation of the host node uptime we impose that stored keywords and source must be deleted if the node goes offline.

The QBI policy can also help in avoiding the problem of *stale references*. As shown in Table 1, the inspection intervals are shorter at the beginning. If the publishing node leaves the system, the references are not refreshed and stale pointers can be deleted faster than in basic KAD publishing scheme. For instance, considering the Weibull-distributed uptime shown in Fig. 3, we can see that a node disconnects after three hours with probability 0.5. This means that, under the basic KAD PR policy, half of the source and keyword replicas are stale for two hours and 22 hours respectively. With the QBI policy, the references are refreshed more frequently at the beginning and the probability to find stale references decreases.

5.2 Desynchronization

In Fig. 4(a) the periodic replacement policy results in a periodic reliability function. For each single unit, the reliability is a decreasing function of time; for a system composed by multiple units, publishing the references at the same time maximizes the reliability at the beginning of the interval, but synchronizes also the minimum values. Thus, introducing a desynchronization mechanism can avoid that all the minimum values happen at the same instant.

The observation of synchronized sources may seem very intuitive, but we were not able to find, from a reliability theory point of view, any work that characterizes the improvement that can be obtained in desynchronizing the inspections. This fact can be explained observing that in engineering systems once the inspection starts, multiple units are inspected at a time, so the synchronization is intrinsic in the inspection policy. In other words, the cost of inspection is composed by a fixed part (setting up the inspection) and a variable part (due to the number of units to be inspected), but the fixed part represents the predominant portion of the cost. In our case, the cost of inspection is linear with the number of units, i.e., each unit must be inspected separately by sending a message. Thus, desynchronization has no impact on the total cost of inspection.

We start computing the improvement that can be obtained in case of PR policy and then we analyze the QBI scheme.

5.2.1 PR policy

Let α be the target reliability of the system composed by source and keywords, α_s and α_k be the target reliability of the subsystems represented by the replicas of the source and the keywords respectively, with $\alpha = \alpha_s \alpha_k$. Moreover, let α_s^c and α_k^c be the target reliability of the single unit c – source and keywords respectively.

Considering the source, for a given replica c, basic inspection interval[4] $\tau = 1/f_s$ and total number of replicas C_s, the inspection intervals with desynchronization, τ_i^c,

[4]For notation simplicity we omit to specify τ_s and τ_k for source and keywords, since it is clear from the context which basic interval we are using.

are given by:

$$\begin{cases} \tau_1^c = c\tau/C_s & c = 1..C_s \\ \tau_i^c = \tau_1^c + (i-1)\tau & c = 1..C_s, i > 1 \end{cases} \tag{9}$$

The first inspection interval is different for each replica, while all the other intervals are constant. The minimum value of the reliability function for the source, α_s, can be found using Eq. (2) and evaluating the reliability function at different instants

$$\alpha_s = 1 - \prod_{c=1}^{C_s} \left(1 - R\left(c\frac{\tau}{C_s}\right)\right) \tag{10}$$

For instance, using reliability function (4), $\tau = 5$ hours and $C_s = 4$ we obtain for the synchronized case a minimum reliability of 96.79%, while in the desynchronized case we have 99.24%. Inspection intervals for keywords and the corresponding minimum value of the reliability function are computed following the same procedure.

5.2.2 QBI policy

In case of QBI policy, we need to solve two problems:

- the inspection intervals $\tau_i^c(\alpha_s^c)$ and $\tau_i^c(\alpha_k^c)$ depend on the target reliability, so even if we obtain an equation similar to Eq. (10) we are not able to find an explicit expression for α_s^c and α_k^c;
- the inspection intervals depend on the history of the host nodes, thus deterministic desynchronization among replicas is not sufficient to assure a permanent desynchronization as in the PR policy.

Since it is useless to consider sources more reliable than keywords, we assume $\alpha_s = \alpha_k = \sqrt{\alpha}$. In this section we consider only the source, since the same results can be found similarly for keywords.

Given α_s and the number of replicas C_s, from Eq. (2) we can obtain a first value for the target reliability for a single replica $\hat{\alpha}_s^c$ inverting the following equation:

$$\alpha_s = 1 - (1 - \hat{\alpha}_s^c)^{C_s} \tag{11}$$

Using $\hat{\alpha}_s^c$ in Eq. (7) we obtain the first inspection interval $\tau_1(\hat{\alpha}_s^c)$. Similarly to the case of PR policy, we can find the desynchronized value of the target reliability, using τ_1:

$$\alpha_s' = 1 - \prod_{c=1}^{C_s} \left(1 - R\left(c\frac{\tau_1}{C_s}\right)\right) \tag{12}$$

In general, α_s' is greater than α_s, thus we iteratively increase τ_1 until we reach a value τ_1^* such that $\alpha_s' = \alpha_s$. The final target reliability for a single unit is then

$$\alpha_s^c = R_R^{-1}(\tau_1^*)$$

The value of α_s^c is found considering only the first inspection interval. We will show in Sect. 6 that this does not represent a limitation and the actual final reliability of the system is always above the target reliability α.

The second problem, assuring desynchronization among replicas, can be solved introducing a *random desynchronization* when the next inspection interval is computed: in this case the probability that two replicas, whenever they are refreshed at the same time and they have the same history, will have the same next inspection interval is negligible. The inspection intervals τ_i^c are given by:

$$\begin{cases} \tau_1^c(\alpha_s^c) = (c/C_s) \cdot \sup\{t > 0 : R_R(t) \geq \alpha_s^c\} \\ \tau_i^c(\alpha_s^c) = \tau_{i-1}^c + \gamma\Delta_i \end{cases} \tag{13}$$

where Δ_i is the $i-$th inter-inspection interval computed as

$$\Delta_i = \sup\{t > 0 : R_{R|\tau_{i-1}}(t) \geq \alpha\} - \tau_{i-1}$$

and γ is a random variable uniformly distributed between 0.7 and 1.3.

5.3 Putting the Pieces Together

In this Section we propose a new publishing scheme for KAD that adopts a **Desynchronized Quantile Based Inspection** (DQBI) policy. In Algorithm 1 we give a high level view of the basic procedure. The scheme assumes that it is possible to specify for each reference its validity and the host node must hold the references until its validity expires; if the host node goes offline it has to delete all the references.

Given the target reliability for the whole system, α, the number of keywords W and the number of replicas for each reference, C_s and C_k, it is possible to compute the target reliability for each replica of the source and the keywords, α_s^c and α_k^c, as described in Sect. 5.2.2.

After publishing for the first time the references, when the different timers expire, the publishing node pings the host nodes. If a host node is found to be up, then the publishing node updates the reliability function (including the information about the past) for that host node and it computes the next expiration of the timer. Then the reference is refreshed with the new expiration time. If the publishing node finds the host node offline, it publishes the reference in a new host node, setting the timer to the basic value.

With the proposed scheme a (stable) node tends to refresh references stored on stable host nodes, i.e. a sort of clustering arises. The number of references on stable nodes could increase, saturating the node. To alleviate this problem, a possible solution is to restart the publishing process from time to time (bootstrapping), i.e. the publishing node may let the keyword and source expire and change the host nodes after some times.

5.4 Cost Analysis

The gain of the new publishing scheme is due mainly to two factors: (i) the cost of refreshing instead of al-

Algorithm 1: DQBI Publishing Scheme

```
Input: Num. of source and keyword replicas ($C_k$, $C_s$),
       target reliability ($\alpha$).
/* auxiliary function                                      */
NextExpirationTime {uptime, $\alpha_n^c$}
    // compute $\tau_i^c$
    compute the next interval (exact value + random
    interval) when this unit will be checked again given its
    previous uptime and the target reliability
endfunction
begin
    for $i \leftarrow 1$ to $C_k$ do
        uptime$_k^i \leftarrow 0$
        $T_k^i \leftarrow$ NextExpirationTime(uptime$_k^i$, $\alpha_k^c$)
        publish keyword replica $i$
    end
    for $j \leftarrow 1$ to $C_s$ do
        uptime$_s^j \leftarrow 0$
        $T_s^j \leftarrow$ NextExpirationTime(uptime$_s^j$, $\alpha_s^c$)
        publish source replica $j$
    end
    When timer $T_n^w = \min_{i,j} \{T_k^i, T_s^j\}$ expires do
        Ping host node that stores the reference
        if host node is up then
            uptime$_n^w$+ = $\Delta T_n^w$ // $\Delta_n^w$ is the
                inter-inspection interval
            $T_n^w \leftarrow$ NextExpirationTime(uptime$_n^w$, $\alpha_n^c$)
            refresh reference and set the new expiration
            time
        else
            uptime$_n^w \leftarrow 0$
            $T_n^w \leftarrow$ NextExpirationTime(uptime$_n^w$, $\alpha_n^c$)
            publish keyword replica $i$
        end
    end
end
```

ways republishing and (ii) the increasing refreshing interval. We assume, as we did for the basic publishing scheme, that the publishing node remains always online, so that we can compare the publishing traffic with Eq.(5). We also assume that there is no maximum expiration time, so that it can increases with no limitation.

Under these assumptions, given a target reliability α_s^c (we consider the source, but the same can be found for the keywords) it is possible to obtain the *average inspection rate* for a single reference (see [19], Sect. 4.2):

$$\beta_s = \frac{1}{(1-\alpha_s^c)^2 \sum_{m=1}^{\infty} (\alpha_s^c)^{m-1} R_R^{-1}\left((\alpha_s^c)^m\right)} \tag{14}$$

At each inspection, with probability α_s^c the host node is still online (so we need only to refresh the reference) and with probability $1-\alpha_s^c$ it is offline, so we need to republish the reference. Assuming that the cost of refreshing is negligible with respect to the cost of republishing, we obtain

$$B_{\text{DQBI}} = (1-\alpha_s^c) S_s C_s \beta_s + W(1-\alpha_k^c) S_k C_k \beta_k \tag{15}$$

where β_s and β_k are the inspection rates for the source and the keywords.

Compared to Eq.(5), in case of $\beta_s = f_s$, $\beta_k = f_k$ and $\alpha_s^c = \alpha_s^c = \alpha^c$, Eq.(15) gives a cost that is $(1-\alpha^c)$% less than PR policy. We should now investigate the impact of the inspection rates β_s and β_k (hereinafter we consider a single reference, so we omit the indication for source and keywords).

Determining the inspection rate means compare in some way the two publishing schemes, PR and DQBI. Given a target reliability for the system, there are two possible comparisons: (i) long run reliability, and (ii) minimum reliability.

Long run reliability is a measure mainly for systems that need to work for a long time and so the aim of the maintenance is to assure an *average* reliability equal to the target reliability. This means that, with the PR scheme, there could be instants when the reliability can drop below the target reliability. For DQBI scheme this cannot happen since the scheme imposes an inspection time such that the reliability is always above the target reliability.

Minimum reliability implies that in every instant the reliability of the system is above the target reliability. This definition is more strict than long run reliability and it should be taken into consideration when considering publishing nodes with relatively small uptime.

6. NUMERICAL RESULTS

In this section we validate our model of the basic and improved scheme using trace driven simulations. The traces [22], collected over six months of crawling the KAD network, report the uptime of approximately half a million of different nodes (identified by their KAD-ID). During these six months, the crawler has taken a snapshot of the node presence every 5 minutes. We define the interval between two snapshots as *time step.*

6.1 Simulator Description and Settings

The basic operations performed by the simulator can be summarized as follows: selection of the host nodes and verification of object reachability.

When a node wants to publish references, it looks for host nodes with ID close to the reference. Since nodes are distributed on the space ID through the hash function, we can assume that there is no correlation between two adjacent nodes, or, in general, there is no correlation among nodes that are close one another. This means that the selection process is equivalent to randomly selecting nodes among all the available nodes (nodes that are online when the publishing process starts). For each reference and for each replica the simulator picks randomly a new host node. When not otherwise stated, we use a default number of replicas $C_s = C_k = 10$ and a file with $W = 2$ keywords.

Once the publishing process has been completed, the simulator starts to verify the presence of the host nodes for each time step between the current time and the next republishing timer expiration (or the maximum observation time). If it is possible to find at least one

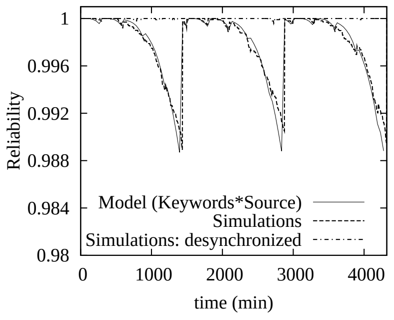


Figure 5: PR scheme: comparison between simulations and analytic results (see Fig. 4(a)).

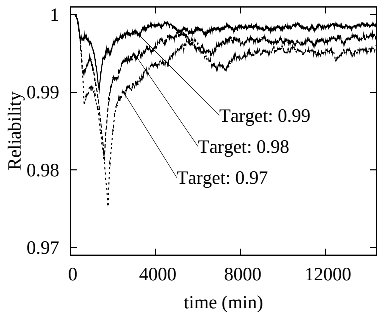


Figure 6: DQBI scheme with different target reliability ($C_s = C_k = 10$).

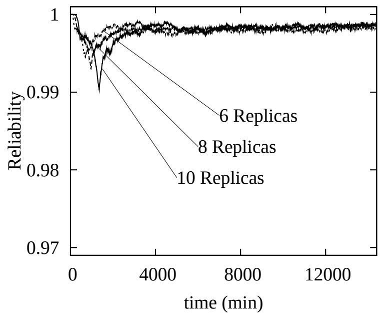


Figure 7: Improved scheme with different number of replicas ($\alpha = 0.99$).

copy of one keyword and one copy of the source, then the reachability is set to 1 for that time step, otherwise is set to 0. In this way, the simulator builds a realization of the publishing process.

By performing multiple realizations, it is possible to obtain, for each time step, multiple samples and, with standard statistical techniques, it is possible to evaluate the availability with a desired confidence level and confidence interval. For each time step we took $5 \cdot 10^5$ samples and the final confidence intervals, for a confidence level of 99% is less than 1% of the point estimate, thus we will not show them in the results.

With the improved scheme, the simulator follows Algorithm (1). Each reference has a timer and it is inspected when the timer expires. The only additional parameter with respect to the basic scheme is the target availability used in the DQBI policy.

6.2 Basic KAD scheme: PR

Using the simulator we want to validate the results obtained from the analysis presented in Sect. 4. We consider an always on publishing node so that we can compare the long term availability without considering the effect of the uptime of the publishing node on the content availability. We use the default values for the republishing interval, i.e., $1/f_s = 5$ hours and $1/f_k = 24$ hours.

In Fig. 5 we compare the simulative results with the analytical ones obtained in Sect. 4 and shown in Fig. 4(a). In the model, we used for the reliability function of each unit the residual uptime distribution obtained from Eq. (4). Our analytical model is able to predict the performance of the publishing scheme of KAD with good accuracy.

Another interesting result is maintaining the periodic replacement, but desynchronizing the republish operation. Figure 5 also shows reliability without synchronization. The reliability becomes almost 1 for every instant. This gain is obtained at a slightly increased cost, due to the fact that the first interval is shorter than the synchronized case.

6.3 Improved Scheme: DQBI

The basic PR scheme adopted by KAD is able to offer a minimum reliability of approximately 0.99. In this section, we analyze the reliability over time of the DQBI scheme for different values of the target reliability, starting from 0.99. It is important to note that, from the analytical point of view, once the target reliability is fixed, the DQBI scheme always assures a reliability at least equal to the target reliability. In this section we investigate how reliability evolves over time.

Using the procedure explained in Sect. 5.2.2, the simulator computes the single unit target reliability, α_s^c and α_k^c, and Algorithm (1) is used to publish. Figure 6 shows the results of the simulations for different values of the target reliability.

The minimum value, approximately equal to the target reliability, is reached only once, while in the other instants the difference between the actual reliability and the minimum reliability is always high. This gain is mainly due to the desynchronization.

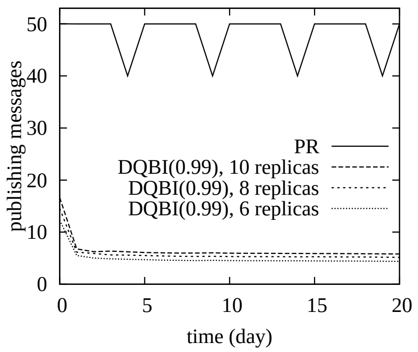
On the one hand, using smaller target reliability α allows us to decrease the cost of publishing; on the other hand the inspection intervals increase. In case of publishing nodes with short uptime, larger inspection intervals increase the probability of having stale references. Instead of decreasing α, another way to reduce the cost is to decrease the number of replicas. In Fig. 7 we show the reliability over time with different number of replicas for each reference, with $C_s = C_k = C$. In order to maintain the same target reliability, the reliability of each unit must increase, and consequently inspection intervals (at least at the beginning) decrease. As we will see, the cost of refreshing and republishing with higher frequency is lower than the cost of maintaining more replicas. In any case, even with few replicas (with high single unit target reliability), the system is able to guarantee the target global reliability.

6.4 Publishing Cost

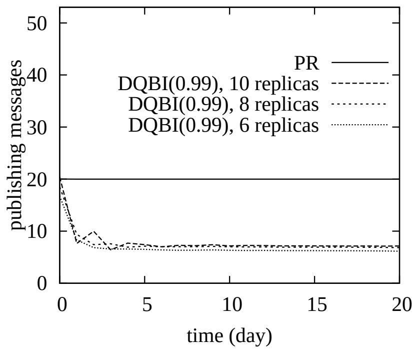
The improved scheme we proposed is able to offer high reliability. In this section we present the results about the cost associated to the different schemes. We measure the cost as the number of publish messages sent for publishing source and keywords during a day. Considering the cost defined in (5) and (15), we rearrange so that we can focus on the total number of *publishing messages*, obtaining

$$\begin{cases} B_{\text{PR}} & = S_s C_s f_s + W S_k C_k f_k \\ & = S_s M_s^{\text{PR}} + S_k M_k^{\text{PR}} \\ B_{\text{DQBI}} & = (1-\alpha_s) S_s C_s \beta_s + W(1-\alpha_k) S_k C_k \beta_k \\ & = S_s M_s^{\text{DQBI}} + S_k M_k^{\text{DQBI}} \end{cases} \tag{16}$$

where M_s^{PR} and M_k^{PR} are the number of publishing messages sent with the PR scheme for source and keywords respectively, and M_s^{DQBI} and M_k^{DQBI} are the corresponding values with the DQBI scheme. In this way, we can compare schemes that use different number of replicas.



(a) Source



(b) Keywords

Figure 8: Cost per day for different schemes.

Figures 8(a) and 8(b) show the cost per day for source and keywords respectively; we compare the cost of the PR scheme with the cost of DQBI scheme with different number of replicas C. For a source, the improvement is evident from the first day[5]. In fact, while in KAD the reliability of the subsystem represented by replicas of the source is higher than the subsystem of the keywords, with our improved scheme the reliability of the two subsystem are set equal, obtaining a gain for the source. For keywords, the cost in the first day is equal for all schemes (all the references must be published), then for DQBI scheme the cost decreases since reference are refreshed.

Table 2: Publishing messages for different schemes.

Scheme	α	C	Source		Keywords	
			Th.	Sim.	Th.	Sim.
PR	-	10	48	48	20	20
DQBI	0.99	10	4.75	5.63	5.38	6.98
DQBI	0.98	10	4.43	5.24	4.85	6.35
DQBI	0.97	10	4.21	4.99	4.51	5.95
DQBI	0.99	8	4.43	4.99	5.23	6.71
DQBI	0.99	6	3.91	4.21	5.11	5.99

Table 2 shows the the cost per day *in the long run.* The table reports both theoretical values and measured with simulations. Theoretical values and simulations match pretty well: the differences are due to the fact that we have traces of six months, and we use them to evaluate cost in the long run. For keywords, the reliability of a single replica is low, so the inspection interval is greater than the inspection interval for the source. Thus, the error with simulation is bigger. With respect to the basic scheme, we can see that with DQBI we have 90% less traffic for the sources and up to 70% less traffic for the the keywords.

7. CONCLUSIONS

The Periodic Replacement publishing scheme is able to offer a high reliability, i.e. the system is robust to node churn. However, this high reliability comes at a high cost: the number of publishing messages is ten times higher than the number of search messages [2].

In this paper we proposed a model of the KAD publishing scheme based on reliability theory. Starting from the weaknesses identified by the model, we proposed an improved publishing scheme, *Desynchronized Quantile Based Inspection* (DQBI), that is able to offer the same reliability with a dramatic reduction in cost.

The results show that it is possible to decrease the number of replicas for each reference, yet maintaining high reliability. In DQBI scheme inspection intervals

[5]With PR scheme, source replicas are republished every 5 hours; in Fig. 8(a) we plot the cost per day, so sources are published 5 times per day, except for the forth day, where the last republish happen at the beginning of the fifth day

for a given reference are small when publishing on a new host node and then increase if the host node (and the publishing node) remains online: this may help to decrease the probability of having stale references.

There are a lot of interesting possible extensions that we are investigating, mainly related to the presence of popular objects. For instance, the two-level publishing scheme, where keywords contain pointers to the source, can be exploited to decrease the republishing overhead associated to keywords of popular objects.

Rare objects with a name that contains popular keywords – or with keywords whose key is close to a popular keyword – also represent a challenge: the reliability of these rare objects may be compromised since the references can fall on a *hot spot node* (node with high load) and can be lost due to the limited number of references host node can hold.

Finally, the analysis can be extended to other P2P system in order to analyze the reliability of other schemes.

8. ACKNOWLEDGMENTS

We would like to thank M. Steiner and T. En-Najjary for many useful discussions and for providing the KAD traces. This work was partially supported by the project CASCADAS (IST-027807) funded by the FET Program of the Eurepean Commission.

9. REFERENCES

[1] M. Steiner, E.W. Biersack, and T. En-Najjary, "Actively Monitoring Peers in Kad," in *Proc. of IPTPS*, Bellevue, WA, USA, Feb. 2007.

[2] M. Steiner, T. Ennajjary, and E.W. Biersack, "A Global View of KAD," in *Proc. of IMC*, San Diego, CA, USA, Oct. 2007.

[3] D. Stutzbach, and R. Rejaie, "Improving Lookup Performance over a Widely-Deployed DHT," in *Proc. of INFOCOM*, Barcelona, Spain, April 2006.

[4] D. Stutzbach, and R. Rejaie, "Understanding Churn in Peer-to-Peer Networks," in *Proc. of IMC*, Rio de Janeiro, Brazil, Oct. 2006.

[5] Y. Qiao, and F.E. Bustamante, "Structured and Unstructured Overlays Under the Microscope - A Measurement-based View of Two P2P Systems That People Use," in *Proc. of the USENIX Annual Technical Conference*, June 2006.

[6] J. Li, J. Stribling, R. Morris, M.F. Kaashoek, and T.M. Gil, "A performance vs. cost framework for evaluating DHT design tradeoffs under churn," in *Proc. of INFOCOM*, Miami, FL, USA, Mar. 2005.

[7] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the Evolution of Peer-to-Peer Systems," in *Proc. of PODC*, Monterey, CA, USA, July 2002.

[8] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *Proc. of the USENIX Annual Technical Conference*, June 2004

[9] R. Bhagwan, S. Savage, and G.M. Voelker, "Understanding Availability," in *Proc. of IPTPS*, February 2003.

[10] R.J. Dunn, J. Zahorjan, S.D. Gribble, H.M. Levy, "Presence-Based Availability and P2P Systems," in *Proc. of IEEE P2P 2005*, Konstanz, Germany, September 2005.

[11] D. Wu, Y. Tian, and K.W. Ng, "Analytical Study on Improving DHT Lookup Performance under Churn," in *Proc. of IEEE P2P 2006*, Cambridge, UK, Sep. 2006.

[12] P. Ji, Z. Ge, J. Kurose, and D. Towsley, "A Comparison of Hard-state and Soft-state Signaling Protocols" IEEE/ACM Transactions on Networking, Vol. 15, No. 2, pp. 281 294, Apr. 2007.

[13] V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web," IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 5, pp. 1266-1276, Sept. 2003.

[14] G. Pierre, M. van Steen, and A.S. Tanenbaum, "Dynamically Selecting Optimal Distribution Strategies for Web Documents," IEEE Transactions on Computers, Vol. 51, No. 6, pp. 637-651, June 2002.

[15] M. Rausand, and A. Hoyland, "System Reliability Theory: Models, Statistical Methods, and Applications," John Wiley & Sons, 1994.

[16] A.T. Gai, and L. Viennot, "Optimizing and Balancing Load in Fully Distributed P2P File Sharing Systems," in *Proc. of ICIW*, Feb 2006.

[17] N. Prabhu, "Stochastic processes: basic theory and its applications," New York: Macmillan, 1965

[18] O. G. Okogbaa, and X. Peng, "Time Series Intervention Analysis for Preventive/Predictive Maintenance Management of Multiunit Systems," in *Proc. IEEE SMC*, San Diego, CA, USA, Oct. 1998.

[19] Y. Yang, and G.-A. Klutke, "Improved Inspection Schemes for Deteriorating Equipment," Probability in the Engineering and Informational Sciences , Vol. 14, pp. 445-460, 2000.

[20] L. Cui, M. Xie, and H.T. Loh, "Inspection schemes for general systems," IIE TRANSACTIONS, vol. 36, n. 9, pages 817-826, 2004.

[21] J. Gurland, and J. Sethuraman, "How Pooling Failure Data May Reverse Increasing Failure Rates," Journal of the American Statistical Association, Vol. 90, No. 432 (Dec., 1995), pp. 1416-1423.

[22] *KAD traces,* http://www.eurecom.fr/∼btroup/kadtraces/, Oct. 2007.