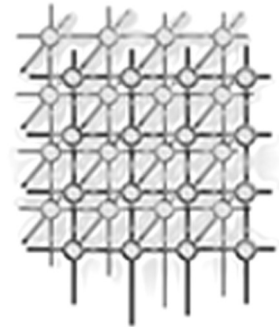


# An analytical study on optimizing the lookup performance of distributed hash table systems under churn



Di Wu<sup>\*,†</sup>, Ye Tian and Kam Wing Ng

*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong*

---

## SUMMARY

The phenomenon of system churn degrades the lookup performance of distributed hash table (DHT) systems greatly. To handle the churn, a number of approaches have been proposed to date. However, there is a lack of theoretical analysis to direct how to make design choices under different churn rates and how to configure their parameters optimally. In this paper, we analytically study three important aspects on optimizing DHT lookup performance under churn, i.e. lookup strategy, lookup parallelism and lookup key replication. Our objective is to build a theoretical basis for designers to make better design choices in the future. We first compare the performance of two representative lookup strategies—recursive routing and iterative routing—and explore the existence of better alternatives. Then we study the effectiveness of lookup parallelism in systems with different churn rates and show how to select the optimal degree of parallelism. Owing to the importance of key replication on lookup performance, we also analyze the reliability of the replicated key under two different replication policies, and show how to perform proper configuration. Besides the analytical study, our results are also validated by simulation, and Kad is taken as a case to show the meaningfulness of our analysis. Copyright © 2007 John Wiley & Sons, Ltd.

*Received 20 June 2006; Revised 26 October 2006; Accepted 18 November 2006*

KEY WORDS: distributed hash table (DHT) system; lookup performance; churn

## 1. INTRODUCTION

In recent years, the distributed hash table (DHT) (e.g., [1–4]) has emerged as a promising approach to build a simple and yet efficient infrastructure for large-scale distributed applications, such as

---

\*Correspondence to: Di Wu, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong.

†E-mail: dwu@cse.cuhk.edu.hk

Contract/grant sponsor: Research Grants Council of the HKSAR; contract/grant number: CUHK4220/04E

---



Grid information services [5,6], P2P file-sharing systems [7], domain naming services [8], storage systems [9,10], content distribution networks [11], etc. However, for the practical deployment of DHT systems in real environments (e.g., desktop Grid, peer-to-peer networks (P2P), etc.), designers have to tackle many new challenges. Generally, in the above environments, there exist a large number of participating nodes, which may join and leave continuously. Such a phenomenon of node dynamics is called ‘churn’. For a DHT system, churn may cause the staleness of routing contacts and the loss of  $\langle \text{key}, \text{data} \rangle$  pairs, and thus deteriorate its lookup performance greatly. As the high-level distributed applications are built on top of the DHT lookup service, their performance will be impacted accordingly.

To improve the DHT performance under churn, researchers have proposed various kinds of approaches (e.g., [9,12–16]) in the aspects of lookup strategy, neighbor selection, data redundancy, parallel lookup, failure recovery, etc. However, the effectiveness of these approaches is mostly verified via simulation or empirical study, and there are few analytical studies to direct how to make design choices under different scenarios and how to configure the parameters optimally. This motivates us to perform analysis to gain a deeper insight into the proposed design solutions, to explore their optimal parameter configuration and to investigate the existence of better alternative approaches.

In this paper, we focus on the following three important aspects of improving the DHT lookup performance under churn.

1. *Lookup strategy.* Among existing DHTs, there are two representative lookup strategies—*recursive routing (RR)* and *iterative routing (IR)*. It is desirable to know, under a given churn rate, which lookup strategy the designer should select and how good it is. It is also useful to know the impacts of parameters and whether we can design better alternatives.
2. *Lookup parallelism.* Although lookup parallelism is already adopted in many DHT implementations, it is unclear how effective it is under different scenarios. As more message overhead will be incurred by lookup parallelism, the designer should know how to select an optimal parallelism degree considering the tradeoff between latency reduction and overhead.
3. *Lookup key replication.* The lookup performance can be improved by increasing the reliability of  $\langle \text{key}, \text{data} \rangle$  pairs through replication. For a given churn rate, it is desirable to know whether simple replication is enough to guarantee the key reliability (i.e. whether it is necessary to install a repair mechanism). In the case when a repair mechanism is necessary, it is also useful to know how the designer should select the replication factor and the repair rate properly to satisfy the requirements of DHT applications.

It is costly and even hard to answer the above questions simply by simulation or empirical study, while theoretical analysis can help to find the answers. We perform analytical studies on the above three aspects, and our main results can be summarized as follows.

- We derive the performance metrics of *RR* and *IR* in closed form, with which the designers can know exactly which strategy is better and how good it is for a given setting. It is observed that both *RR* and *IR* can only perform well under a certain range of churn rates. To make the lookup strategy perform the best under the whole range of churn rates, two enhanced alternatives are considered. We find that a simple *two-phase routing strategy (RR + IR)* can exploit well the benefits of both *RR* and *IR*. Moreover, we also propose another lookup strategy, in which *RR* is enhanced with an *ACK* mechanism. It further improves the lookup performance, and behaves better than *RR*, *IR* and *RR + IR*.



- We apply lookup parallelism to different lookup strategies and compare their performance analytically. It is found that, under a high-churn environment, lookup parallelism can reduce the lookup latency effectively, and, among the considered lookup strategies, IR performs the best after being parallelized; however, under a low-churn environment, the effectiveness of lookup parallelism is only obvious for RR, and there is not much improvement for other lookup strategies. Considering the tradeoff between the lookup latency and the message overhead, our theorems can help designers determine the optimal parallelism degree directly.
- We study two typical policies of key replication on improving lookup performance under churn: (i) replication without a repair mechanism; and (ii) replication with a repair mechanism. We formally derive the expected lifetime of a replicated key under the above two replication policies. It is found that whether a repair mechanism is necessary depends largely on the node lifetime distribution, instead of the system churn rate. Under an exponential lifetime distribution, a repair mechanism is required no matter whether the churn rate is high or low; however, under a Pareto lifetime distribution, if there exists a very heavy tail, simple replication is enough to guarantee a rather long lifetime of the replicated key. This is true even under a high-churn environment. As to the selection of the replication factor and the repair rate, it is application specific. We discuss some practical considerations (e.g., data type, data lifetime, etc.) and show how to configure the parameters properly.

Our theoretical results make it possible for designers to know how to make the optimal design choices under a given churn rate precisely, instead of based on intuition. In addition to the analytical study, we also perform simulations to validate our results, and one real DHT system, Kad [7], is taken as an example to demonstrate the meaningfulness of our analysis for the DHT design.

The remainder of this paper is structured as follows. We first introduce the related work in Section 2, then we present the system model in Section 3. Based on the system model, we perform analytical studies on lookup strategy, lookup parallelism and lookup key replication in Sections 4, 5 and 6, respectively. In Section 7, we use Kad as an example to show how to perform proper configuration based on our analysis. Finally, Section 8 summarizes the whole paper.

## 2. RELATED WORK

DHT systems are a new class of distributed systems which support scalable information storage and lookup. A DHT system organizes participating nodes into an overlay network, and partitions the global key space among them. All  $\langle key, data \rangle$  pairs whose keys fall into a node's key space are stored at that node. To retrieve the data associated with a given key, a lookup operation is invoked to route the query message to the key owner. Since 2001, a large number of DHT designs have been introduced, such as Chord [1], CAN [2], Pastry [3], Tapestry [4] and Kademlia [17]. They differ in the routing strategy and key space partitioning strategy.

To deploy DHTs in dynamic large-scale distributed environments (e.g., desktop Grid, P2P networks), system churn should be properly handled. As the first step, quite a few measurements have been conducted in order to understand well the characteristics of churn. In recent years, Sariou [18] and Stutzbach and Rejaie [19] have performed detailed measurements on real P2P file-sharing systems, including Napster [20], Gnutella [21], BitTorrent [22] and Kad [7]. They have found similar phenomena



of system churn across these systems, and the node lifetime follows a heavy-tailed distribution. Stribling and Yoshikawa [23] monitored the node availability in the PlanetLab system [24]. Based on their all-pairs ping dataset [23], it has been found that the PlanetLab system is also dynamic and the node lifetime follows approximately an exponential distribution.

To mitigate the impacts of churn, a number of solutions have been proposed. In [12], Rhea *et al.* proposed to handle churn by reactive or periodic failure recovery, accurate calculation of lookup timeout and proximity neighbor selection. Later, Lam and Liu [13] further introduced the concept of  $K$ -consistency and designed join and recovery protocols for hypercube-like DHTs. By simulation, they showed that their protocols can handle various churn rates. In [9], Dabek *et al.* explored a range of solutions, such as iterative or recursive routing, proximity neighbor selection, replication and erasure coding, in their implementation of the DHash system. In [25], Li *et al.* provided a PVC (performance versus cost) model to evaluate different DHT design tradeoffs under churn by their self-developed simulator—p2pSim [26]. In addition to performing improvements on existing DHTs, some new DHTs have been designed, particularly for churn-intensive environments, such as Accordion [15] and EpiChord [16]. Different from the above simulation-based study, Stutzbach and Rejaie [27] have developed measurement tools to study the impacts of churn in the real Kad network [7].

There also exist some theoretical works about DHT performance under churn. However, most of them mainly focus on one special kind of DHT. In [28], Liben-Nowell *et al.* provided asymptotic performance bounds of Chord under churn. The concept of half-life is introduced to measure the rate of membership changes, and it is shown that  $\Omega(\log n)$  per-node maintenance bandwidth is required per half-life to maintain the connectivity of Chord. In [29,30], Krishnamurthy *et al.* and El-Ansary *et al.* analyzed the performance of Chord under churn by using a master-equation-based approach. It accurately derived the fraction of failed or incorrect routing pointers, which can be used to predict the Chord lookup performance under churn. Our analysis differs in that we consider the common issues in DHT designs, and our objective is not just to analyze the impact of churn, but to help system designers make wise decisions to better handle churn. The only similar work to the best of our knowledge is by Godfrey *et al.* [31], but they consider the problem of how to minimize churn by node selection, which can be seen as a complementary work to ours.

### 3. SYSTEM MODEL

To make our analysis tractable, some necessary assumptions are made as follows. First, we assume that the underlying physical links are ideal, and there is no packet loss during transmission. Therefore, the routing failure is only caused by stale contacts in the routing table due to system churn. Second, to model the system churn, each arriving node is assumed to be associated with a lifetime  $L$ , which is a random variable following a certain distribution. In this paper, two typical kinds of node lifetime distribution are considered.

- An *exponential lifetime distribution*, which is widely used in reliability theory [32] to model the time to failure of components, and has been observed in the PlanetLab system. Its cumulative distribution function (CDF) is given by  $F(x) = 1 - e^{-\lambda x}$  ( $\lambda > 0$ ).
- A *Pareto lifetime distribution*, which is observed in real P2P file-sharing systems [18], with the CDF given by  $F(x) = 1 - (1 + (x/\beta))^{-\alpha}$ , where  $\alpha$  represents the heavy-tailed degree and  $\beta$  is



a scale parameter. Under a Pareto lifetime distribution, most nodes have a short lifetime, while a few nodes have a much longer lifetime.

Supposing that a node is alive at a particular time point  $t_0$ , then the period from  $t_0$  to the time point that the node leaves the system is defined as the residual life of that node. Denote  $R$  to be the residual lifetime of a node since  $t_0$ . When the system enters a steady state, the distribution of the residual lifetime  $R$  is independent of  $t_0$  and its CDF  $F_R(x)$  is given, according to [33], as follows:

$$F_R(x) = P(R < x) = \frac{1}{E[L]} \int_0^x (1 - F(z)) dz \quad (1)$$

where  $F(x)$  is the CDF of node lifetime  $L$  and  $E[L]$  is the expectation of node lifetime  $L$ .

When a node  $v$  joins the system, it will select (or be assigned) a set of existing nodes as its neighbors. Here, the neighbor selection is assumed to be independent of the node lifetime. Under the steady system state, in the case when  $t_0$  refers to the time point that node  $v$  joins the system, the residual lifetime of its neighbors follows the distribution given in Equation (1).

A neighbor leaves the system when its residual lifetime is exhausted. In that case it takes some time  $S$  for the node  $v$  to detect the failure and replace it with a live node. During that period, in the routing table of node  $v$ , the contact of the departed neighbor is in the 'stale' state. If the node  $v$  forwards a lookup to a 'stale' contact, the lookup will fail and cause a timeout. The contacts that are not in the 'stale' state are referred to as 'fresh' contacts. Under the steady state, the probability that a routing contact is in the 'fresh' state is given by

$$p = \frac{E[R]}{E[R] + E[S]} \quad (2)$$

where  $E[R]$  is the expected residual lifetime of a neighbor ( $E[R] = \int_0^\infty (1 - F_R(x)) dx$ ) and  $E[S]$  stands for the expected time for neighbor replacement.

In the following sections we will perform analysis on lookup strategy, lookup parallelism and lookup key replication based on the above system model.

#### 4. LOOKUP STRATEGY

The lookup strategy defines how a query is routed towards the destination. Among existing DHT systems, the lookup strategies can be generally categorized into two types: RR and IR.

Most DHTs (e.g., Tapestry [4], Pastry [3], etc.) adopt RR in their design, in which intermediate nodes on the routing path will directly forward the query to the node in the next hop, without making an acknowledgement to the originator (as shown in Figure 1(a)). Thus, the query message can be routed as quickly as possible, but the originator has no control over the lookup process. If the lookup returns no response, the originator has to wait until its timeout, as there is no knowledge about the real cause.

In contrast to RR, IR enables the originator to control the whole lookup process. In each step, instead of letting the intermediate node forward the query on its behalf, the originator will ask it to return the address of the next hop. With the returned address, the originator initiates the query to each successive node on its own, until the destination is reached (as shown in Figure 1(b)). Owing to its manageable behavior, Kademlia [17] and its variant Kad use IR in their design.

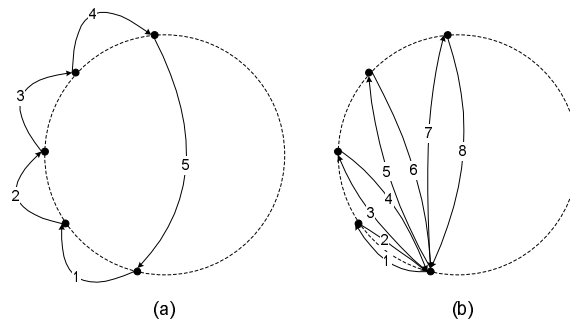


Figure 1. Lookup strategies: (a) RR; (b) IR.

Intuitively, RR is more efficient than IR. However, considering the factor of system churn, which will be a better choice? What can IR bring to us? And, if possible, can we design a more effective lookup strategy?

In this section, we will compare the performance of RR and IR under churn analytically, and explore whether there exist better approaches.

#### 4.1. Analysis

Without loss of generality, we use a variable  $l$  to denote the length of the routing path between the source and the destination, which is also the number of routing hops from the source to the destination (e.g.,  $l = 4$  in Figure 1). The value of  $l$  is different in various DHTs, for example.  $O(\log N)$  in Chord [1],  $O((d/4)n^{1/d})$  in CAN [2],  $O(1)$  in Onehop Overlay [34], etc. We denote the average one-hop routing latency by  $\Delta$ , and the average round-trip time by  $RTT = 2\Delta$ .

In both RR and IR, when the lookup responds with no reply, a new lookup will be launched after a timeout. However, they differ in their timeout settings. In RR, the timeout is set as  $T_l$ , which should be no less than the time to complete the entire lookup, i.e.  $T_l \geq (l + 1)\Delta$ ; in IR, its timeout  $T_h$  needs to be no less than the round-trip time, i.e.  $T_h \geq RTT = 2\Delta$ .

To measure the lookup performance, two metrics are adopted: (1) the lookup latency  $W$ , which is the latency to complete the entire lookup; and (2) the message overhead  $C$ , which is the number of messages incurred during the lookup process. For convenience, the notation used in this paper is summarized in Table I.

By applying the theory of stochastic processes, we have the following results for RR.

**Theorem 4.1.** For RR, the expected lookup latency is given by

$$E[W_{RR}] = (l + 1)\Delta + \frac{1 - p^l}{p^l} T_l$$



Table I. Notation.

Symbol	Definition
$L$	The lifetime of a node
$R$	The residual lifetime of a node
$S$	The time required for a neighbor replacement
$p$	The probability that a contact is 'fresh' under the steady state, $p = E[R]/(E[R] + E[S])$
$l$	The length of the routing path between the source and the destination
$\Delta$	The average one-hop routing latency
$\sigma$	The standard deviation of one-hop routing latency
$RTT$	The average round-trip time, $RTT = 2\Delta$
$T_l$	The timeout of the entire lookup, $T_l \geq (l + 1)\Delta$
$T_h$	The timeout of the round-trip routing, $T_h \geq RTT = 2\Delta$
$W$	The latency to complete the entire lookup
$C$	The message overhead (i.e. the number of messages) incurred during a lookup
$k$	The parallelism degree of a parallel lookup
$r$	The replication factor (i.e. the number of replicas)
$T$	The lifetime of a replicated key
$T_{rp}$	The period between two consecutive repairs
$\mu$	The repair rate for a replicated key, $\mu = 1/T_{rp}$

and the expected message overhead is given by

$$E[C_{RR}] = (l + 1) + \frac{(1 - p^l)}{p^l} \times \frac{[1 - (l + 1)p^l + lp^{l+1}]}{(1 - p)^2}$$

*Proof.* In RR, let  $N$  be the number of timeouts experienced before arriving at the destination.  $N$  is a random variable satisfying  $P(N = n) = (1 - p^l)^n p^l$  ( $n = 0, 1, \dots$ ). The lookup latency  $W$  can then be represented by  $W = (l + 1)\Delta + NT_l$  and its expected value is calculated by

$$E[W_{RR}] = \sum_{n=0}^{\infty} [(l + 1)\Delta + nT_l]P(N = n) = (l + 1)\Delta + \frac{1 - p^l}{p^l} T_l$$

To calculate the message overhead, we first define  $M$  to be the number of messages sent during one timeout.  $M$  is a random variable satisfying the geometric distribution with  $P(M = i) = (1 - p)p^{i-1}$  ( $i = 1, 2, \dots, l$ ). Therefore, the expected value of  $M$  is given by  $E[M] = \sum_{i=1}^l i(1 - p)p^{i-1}$  and the expectation of the total number of messages can be given by  $E[C_{RR}] = (l + 1) + E[N] \times E[M]$ , where  $E[N] = (1 - p^l)/p^l$ . The theorem can be proved after reduction.  $\square$

Similarly, we have the results about the performance of IR as follows.

**Theorem 4.2.** For IR, the expected lookup latency is given by

$$E[W_{IR}] = 2l\Delta + \frac{1 - p}{p} lT_h$$

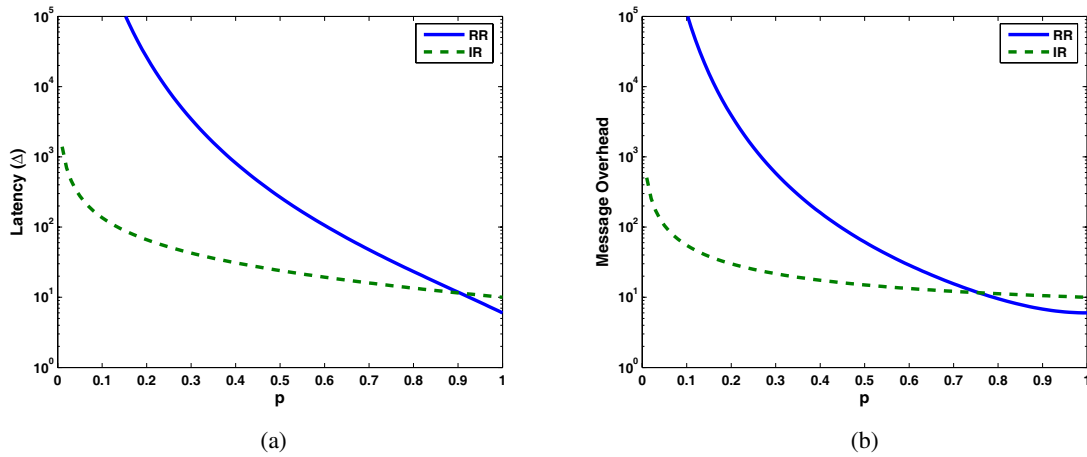


Figure 2. Comparison between RR and IR: (a) expected lookup latency; (b) expected message overhead.

and the expected message overhead is given by

$$E[C_{IR}] = \left(\frac{1}{p} + 1\right)l$$

*Proof.* In IR, the entire lookup process can be divided into  $l$  independent one-hop lookup processes. Denote by  $W'$  the time required for the originator to receive the reply including the addresses of the next hop and by  $N$  the number of timeouts that occurred within the duration. The *probability mass function (PMF)* of  $N$  is given by  $P(N = n) = p(1 - p)^n$  ( $n = 0, 1, \dots$ ), and the expected latency to reach the next hop is given by

$$E[W'] = \sum_{n=0}^{\infty} (RTT + nT_h)P(N = n) = RTT + \frac{1-p}{p}T_h = 2\Delta + \frac{1-p}{p}T_h$$

By  $E[W_{IR}] = lE[W']$ , we can deduce the result.

The message overhead can be computed in a similar way. Denote  $C'$  as the number of messages sent in one-hop routing, then we have  $E[C'] = \sum_{n=0}^{\infty} (n + 2)P(N = n) = (1/p) + 1$ , with which the average message overhead can be given by  $E[C_{IR}] = lE[C'] = ((1/p) + 1)l$ .  $\square$

For a better understanding of Theorems 4.1 and 4.2, we plot the expected latency and overhead of RR and IR under different  $p$  in Figure 2. As  $p = E[R]/(E[R] + E[S])$ , given that  $E[S]$  is fixed, the probability  $p$  reflects the rate of system churn. When  $p$  is large, the churn rate is low (i.e.  $E[R]$  has a large value), but when  $p$  is small, the churn rate is high (i.e.  $E[R]$  has a small value). In fact, in most DHT systems, the time required for neighbor replacement  $S$  is determined by the stabilization period of the routing table. Under the steady state, its expectation  $E[S]$  will approach a fixed value. Therefore, it is proper for us to use  $p$  to represent the system churn rate. Moreover, the range of  $p$  corresponds to the whole range of churn rates. For the timeout setting, similar to the TCP-style timeout setting [35],

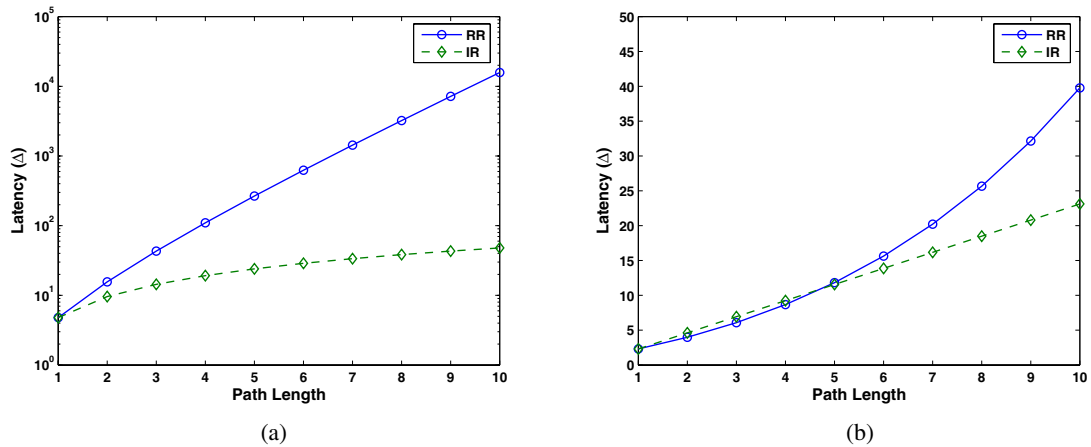


Figure 3. Impact of path length on lookup latency: (a) under a high-churn environment ( $p = 0.5$ ); (b) under a low-churn environment ( $p = 0.9$ ).

we define  $T_l = (l + 1)(\Delta + 4\sigma)$  and  $T_h = 2(\Delta + 4\sigma)$ , where  $\sigma$  is the standard deviation of one-hop routing latency, which is assumed to be  $0.1\Delta$  here.

From Figures 2(a) and (b), it is found that neither RR nor IR can outperform the other in the whole range of churn rates. When the churn rate is low (e.g.,  $p > 0.9$ ), RR performs better than IR, and has low latency and low overhead, but its performance deteriorates greatly when the churn rate is high (e.g.,  $p < 0.5$ ). In contrast, IR is rather robust to the system churn, and performs better than RR under intensive churn.

In the above study, the length of the routing path  $l$  is assumed to be fixed ( $l = 5$ ). However, based on the theorems,  $l$  will also have an impact on our comparison results. Figures 3(a) and (b) show how the path length has an impact on the routing latency in systems with a high churn rate ( $p = 0.5$ ) and a low churn rate ( $p = 0.9$ ), respectively. In Figure 3(a), we observe that, when the churn rate is high, the latency of RR increases almost exponentially with the increase of path length, while IR just has a smooth increase. The lookup performance of RR is always less efficient than IR when the path length  $l \geq 2$ . Figure 3(b) shows that, when the churn rate is low, in the case when the routing path length is short ( $l \leq 5$ ), RR can perform better than IR, but for a longer routing path RR becomes worse again.

The above observation can be directly explained by Theorems 4.1 and 4.2. From the theorems, we can see that, for RR,  $E[W_{RR}] \propto 1/p^l$  and  $E[C_{RR}] \propto 1/p^l$ , while, for IR,  $E[W_{IR}] \propto (1/p) \times l$  and  $E[C_{IR}] \propto (1/p) \times l$ . Thus, with the increase of either the churn rate (i.e.  $p$  becomes smaller) or the routing path length  $l$ , the performance of RR deteriorates much more quickly than that of IR.

The meaningfulness of our theorems is that, for a given setting (e.g., node lifetime distribution, neighbor replacement time, routing path length, timeout setting, etc.), we can know exactly which lookup strategy is better and how good it is. It is useful because we can select the most suitable lookup strategy for a specific environment.



## 4.2. Enhancements

From the above observations, we can find that both RR and IR can only function well within a certain range of churn rates. Although we can select a proper lookup strategy for a given system churn rate, it is possible that the system churn rate may vary with time. Therefore, it will be more ideal if we can devise better approaches that exploit the benefits of both and perform the best under the whole range of churn rates.

Let us first consider a simple *two-phase routing strategy (RR + IR)*: in the first phase, RR is used for fast routing; once the RR fails, then we switch to IR. The routing process includes two phases and its performance is given by Theorem 4.3.

**Theorem 4.3.** For two-phase routing (RR + IR), the expected lookup latency is given by

$$E[W_{RR+IR}] = (l + 1)\Delta p^l + \left[ T_l + 2l\Delta + \frac{1-p}{p}lT_h \right] (1 - p^l)$$

and the expected message overhead is given by

$$E[C_{RR+IR}] = (l + 1)p^l + \left[ \frac{lp^{l+1} - (l + 1)p^l + 1}{(1 - p)^2} + \left( \frac{1}{p} + 1 \right) l \right] (1 - p^l)$$

*Proof.* In two-phase routing, let  $X$  be the number of timeouts for RR before initiating IR,  $X$  can only have two values, 0 and 1, and  $P(X = 0) = p^l$ ,  $P(X = 1) = 1 - p^l$ . The expected latency is given by  $E[W_{RR+IR}] = (l + 1)\Delta P(X = 0) + (T_l + E[W_{IR}])P(X = 1) = (l + 1)\Delta p^l + (T_l + 2l\Delta + ((1 - p)/p)lT_h)(1 - p^l)$ .

Its average message overhead is given by  $E[C_{RR+IR}] = (l + 1)P(X = 0) + (E[M] + E[C_{IR}])P(X = 1) = (l + 1)p^l + (E[M] + E[C_{IR}])(1 - p^l)$ . Here,  $M$  is the number of messages sent before the failure of RR, and

$$E[M] = \sum_{i=1}^l i(1 - p)p^{i-1} = \frac{lp^{l+1} - (l + 1)p^l + 1}{(1 - p)^2}$$

$E[C_{IR}]$  is the expected number of messages sent during IR in the case of failure of RR. The value of  $E[C_{IR}]$  is given in Theorem 4.2.  $\square$

In addition to RR + IR, we also propose another lookup strategy, which enhances RR with an acknowledgement (ACK) mechanism. It is called the *recursive routing with ACK strategy (RR + ACK)*.

The basic idea of RR + ACK is that, in addition to normal forwarding like RR, an intermediate node also sends an ACK containing the addresses of the next hop to the originator. In the case of failure, the originator can reinitiate another lookup according to the addresses in the latest ACK. It is unnecessary to start the lookup from the very beginning.

As to the timeout setting, it is a bit different in RR + ACK. For the routing in the first hop or immediately after a failure, the originator determines whether the routing fails or not based on the arrival of the first ACK. So the timeout should be no less than the round-trip time. However, for the routing in the following hops, the originator can detect the routing failure based on the inter-arrival time between ACKs, and the timeout only needs to be no less than the one-hop routing latency. Here, for implementation simplicity, we define only one timeout for both situations and its value is set as  $T_h$ , which is the timeout of the round-trip routing.

For RR + ACK, its performance is given in Theorem 4.4.



**Theorem 4.4.** For RR with ACK (RR + ACK), the expected lookup latency is given by

$$E[W_{RR+ACK}] = (l + 1)\Delta + \frac{1 - p}{p}lT_h$$

and the expected message overhead is given by

$$E[C_{RR+ACK}] = \left(p + \frac{1}{p}\right)l$$

*Proof.* In RR with ACK, let  $X$  be the number of RR failures during the lookup process,  $0 \leq X \leq l$ . The PMF of  $X$  is given by  $P(X = i) = \binom{l}{i} p^{l-i} (1 - p)^i$  ( $i = 0, 1, \dots, l$ ). In the case of  $i$  failures, the expected latency  $E[W(i)] = (l - i + 1)\Delta + i(T_h + W^*)$ , where  $W^*$  is the expected latency for the originator to reach the next hop using the addresses in the ACK. Denote by  $N$  the number of timeouts that occurred before reaching the next hop, so

$$W^* = \sum_{n=0}^{\infty} (\Delta + nT_h)P(N = n) = \sum_{n=0}^{\infty} (\Delta + nT_h)p(1 - p)^n = \Delta + \frac{1 - p}{p}T_h$$

Then we can deduce the expected latency to reach the destination,

$$E[W] = \sum_{i=0}^l E[W(i)]P(X = i) = (l + 1)\Delta + \frac{1 - p}{p}lT_h$$

As to the expected overhead, we can derive it in a similar way. In the case of  $i$  RR failures in the whole lookup process, the expected overhead  $E[C(i)] = (l - i) \times 2 + iC^*$ , where  $C^*$  is the expected overhead for the originator to reach the next hop in the case of forwarding failure. We have

$$E[C^*] = \sum_{n=0}^{\infty} (n + 2)P(N = n) = 1 + \frac{1}{p}$$

and

$$E[C] = \sum_{i=0}^l E[C(i)]P(X = i) = \left(p + \frac{1}{p}\right)l \quad \square$$

The performance comparison of RR, IR, RR + IR and RR + ACK is plotted in Figure 4. From Figures 4(a) and (b), we find that the simple two-phase lookup strategy (RR + IR) combines RR and IR rather well. In low-churn environments (e.g., when  $p \geq 0.9$ ), RR + IR has a comparable lookup latency to RR, which is better than IR; when the churn becomes intensive (e.g., when  $p < 0.9$ ), it is only slightly worse than IR, but much better than RR. However, the best is the recursive routing with ACK (RR + ACK) strategy. It has the lowest lookup latency among the four in the whole range of churn rates, and comparable message overhead to IR.

In Figures 5(a) and (b), we also study the impact of path length on the lookup latency of four lookup strategies under a high-churn environment and a low-churn environment separately. Under both environments, IR, RR + IR and RR + ACK are robust to the variation of path length, and only RR deteriorates greatly. RR + ACK has the best performance in all the cases. In fact, RR + IR and

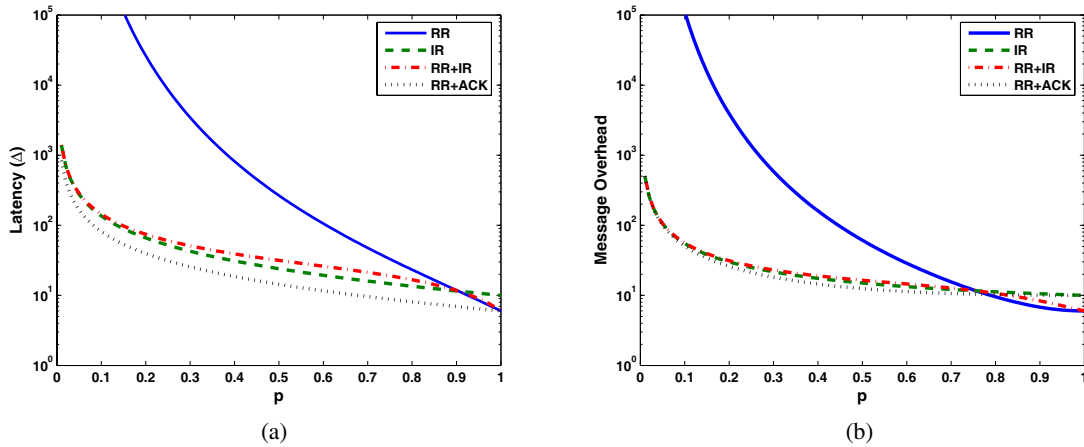


Figure 4. Performance comparison among RR, IR, RR + IR and RR + ACK: (a) expected lookup latency under different churn rates ( $0 \leq p \leq 1$ ); (b) expected message overhead under different churn rates ( $0 \leq p \leq 1$ ).

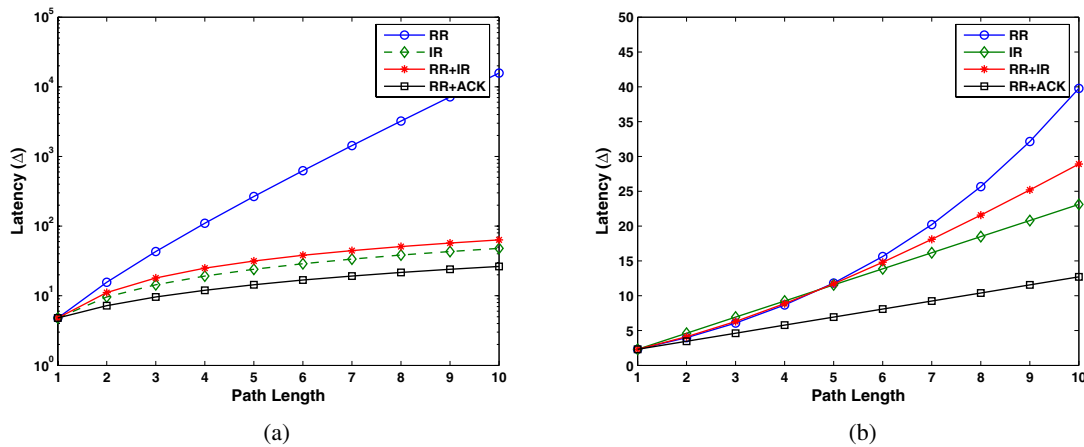


Figure 5. Impact of path length on RR, IR, RR + IR and RR + ACK: (a) under a high-churn environment ( $p = 0.5$ ); (b) under a low-churn environment ( $p = 0.9$ ).

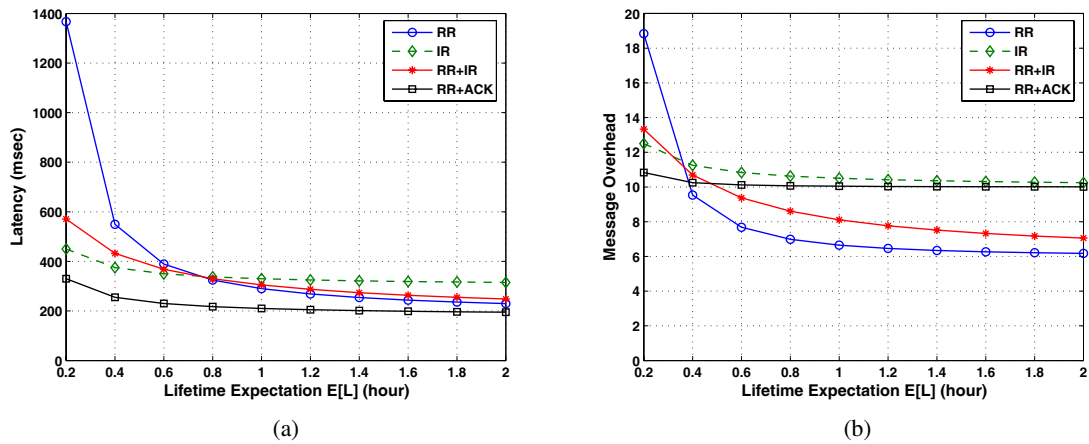


Figure 6. Simulation-based comparison among RR, IR, RR + IR, RR + ACK: (a) average lookup latency; (b) average message overhead.

RR + ACK can be seen as the variants of IR. Therefore, they have similar robustness to the variation of the churn rate and routing path length.

Besides numerical study, we also validate our results with a discrete event simulator of a Kademlia-like DHT system. Modifications are made to support different lookup strategies. The node lifetime satisfies a Pareto distribution with  $\alpha = 3$ .  $\beta$  is adjusted to change the lifetime expectation  $E[L]$  and simulate different levels of churn. In our settings, the time for neighbor replacement is random in the range [4, 20] minutes and the system consists of 5000 nodes.

Figure 6 shows the simulation results, which correlate well with our analysis results in the previous part. IR, RR + IR and RR + ACK are all robust to the system churn, and RR + ACK performs the best to reduce the lookup latency. RR performs rather well under low-churn environments, and has low latency and low overhead, but its performance deteriorates greatly when the system becomes highly dynamic. RR + IR achieves the benefits of RR and IR, and has a comparable performance to the better one of the two in the whole range of churn. Both RR + IR and RR + ACK are good design candidates that can adapt to different levels of system churn.

## 5. LOOKUP PARALLELISM

Lookup parallelism can be used to improve the DHT lookup performance under churn. In a parallel lookup, the originator initiates multiple lookups simultaneously. Even if some of them meet stale contacts, the whole lookup process can continue to progress without being blocked.

What we want to know is, which lookup strategy is more suitable to be parallelized? And how much efficiency can we gain from lookup parallelism? As the performance improvement of parallel lookup is at the cost of more message overhead, another straightforward question arises that, given a certain level of system churn, what is the optimal degree of parallelism?



In the following, we assume that there exists redundancy in the routing table to support lookup parallelism. Moreover, the number of concurrent lookups of a parallel lookup is referred to as the ‘parallelism degree’, denoted by  $k$ .

Let us first consider applying lookup parallelism to RR. Initially, the originator launches  $k$  recursive lookups concurrently. Each recursive lookup is executed independently, and the entire lookup is successful in the case when one recursive lookup reaches the destination and returns a reply to the originator. If there is no reply returned within the timeout period  $T_l$ , the originator initiates another  $k$  recursive lookups. The process will be repeated until the reply from the destination is received. By analysis, we have the following results for the *parallel recursive routing (pRR)*.

**Theorem 5.1.** For the pRR, the expected lookup latency is given by

$$E[W_{\text{pRR}}] = (l + 1)\Delta + \frac{(1 - p^l)^k}{1 - (1 - p^l)^k} T_l$$

and the expected message overhead is given by

$$E[C_{\text{pRR}}] = \frac{kp^l(l + 1)}{1 - (1 - p^l)^k} + k \frac{(1 - p^l)}{1 - (1 - p^l)^k} \times \frac{1 - (l + 1)p^l + lp^{l+1}}{(1 - p)^2}$$

*Proof.* The proof is similar to Theorem 4.1. Let  $N$  be the number of timeouts experienced before arriving at the destination, and let  $q$  be the probability that all  $k$  recursive lookups fail to reach the destination. We have  $q = (1 - p^l)^k$  and  $P(N = n) = q^n(1 - q)$  ( $n = 0, 1, \dots$ ). The expected number of timeouts is given by  $E[N] = q/(1 - q)$ . Then, we can give the expected lookup latency by

$$E[W_{\text{pRR}}] = (l + 1)\Delta + E[N] \times T_l = (l + 1)\Delta + \frac{q}{1 - q} T_l = (l + 1)\Delta + \frac{(1 - p^l)^k}{1 - (1 - p^l)^k} T_l$$

Let  $M$  be the number of messages incurred by a single recursive lookup before its failure, so  $E[M] = \sum_{i=1}^l i(1 - p)p^i$  ( $i = 1, \dots, l$ ). The expected message overhead of pRR includes two parts: (1) the expected number of messages incurred in the successful parallel lookup; (2) the expected number of messages incurred during the failed parallel lookups. It can be represented by  $E[C_{\text{pRR}}] = C_1 + E[N] \times kE[M]$ , where  $C_1$  refers to the first part, and  $E[N] \times kE[M]$  refers to the second part.  $E[N]$  and  $E[M]$  have been given in the above already, so we just need to calculate  $C_1$ . Let  $X$  be the number of successful replies from the destination,  $P(X = i) = \binom{k}{i} (p^l)^i (1 - p^l)^{k-i}$ . Then

$$C_1 = (l + 1)E[X | X \geq 1] + (k - E[X | X \geq 1]) \times E[M]$$

and

$$E[X | X \geq 1] = \sum_{i=1}^k iP(X = i | X \geq 1) = \sum_{i=1}^k i \frac{P(X = i)}{P(X \geq 1)} = \sum_{i=1}^k i \frac{P(X = i)}{1 - P(X = 0)} = \frac{kp^l}{1 - q}$$

After reduction, we can deduce  $E[C_{\text{pRR}}]$ . □

For the *parallel iterative routing (pIR)*, in each routing hop except the last hop,  $k$  one-hop lookups are initiated concurrently. Every successful lookup will return a list of contacts in the next hop, and here the number of returned contacts by one lookup is assumed to be bigger than the parallelism degree  $k$ . After receiving the reply, the originator can initiate another  $k$  lookups for the next hop. The process will be repeated until the destination is reached. In the last hop, as there exists one destination only, a single lookup is performed.

Using the same metrics as in Section 4, our results for the performance of pIR are as follows.



**Theorem 5.2.** For the pIR, the expected lookup latency is given by

$$E[W_{pIR}] = 2l\Delta + \left[ \frac{(1-p)^k}{1-(1-p)^k}(l-1) + \frac{1-p}{p} \right] T_h$$

and the expected message overhead is given by

$$E[C_{pIR}] = kp(l-1) + \frac{k(l-1)}{1-(1-p)^k} + \frac{1}{p} + 1$$

*Proof.* In each hop, let  $q$  be the probability that all  $k$  lookups fail simultaneously. As the probability of forwarding to a live contact is given by  $p$ , we can directly represent  $q = (1-p)^k$ . Similar to the deduction in Theorem 4.2, the entire process can also be divided into  $l$  phases owing to their independence. In the first  $(l-1)$  hops,  $k$  lookup is performed, while, in the final hop, a single lookup is performed. Let  $W''$  be the time required for  $k$  lookup to return the neighbor contacts in the next hop, and  $N$  be the number of timeouts within this period,  $P(N = n) = q^n(1-q)$  ( $n = 0, 1, \dots$ ). The expected value of  $W''$  is given by

$$E[W''] = \sum_{n=0}^{\infty} (RTT + nT_h)P(N = n) = RTT + \frac{q}{1-q}T_h = 2\Delta + \frac{(1-p)^k}{1-(1-p)^k}T_h$$

The expected latency of the whole lookup is

$$E[W_{pIR}] = (l-1)E[W''] + E[W'] = 2l\Delta + \left[ \frac{(1-p)^k}{1-(1-p)^k}(l-1) + \frac{1-p}{p} \right] T_h$$

where  $E[W'] = 2\Delta + (1-p)/pT_h$ . The deduction of  $E[W']$  is given in the proof of Theorem 4.2.

Define  $C''$  to be the number of incurred messages in one-hop parallel lookup, which includes lookup messages and reply messages. In the case of  $n$  timeouts,  $C'' = (n+1)k + kp$ . Therefore, the expected value of  $C''$  is given by

$$\begin{aligned} E[C''] &= \sum_{n=0}^{\infty} [(n+1)k + kp]P(N = n) = kp + \sum_{n=0}^{\infty} (n+1)k(1-q)q^n = kp + \frac{k}{1-q} \\ &= kp + \frac{k}{1-(1-p)^k} \end{aligned}$$

Thus, we can give the expected number of messages incurred in the lookup process by

$$E[C_{pIR}] = (l-1)E[C''] + E[C'] = kp(l-1) + \frac{k(l-1)}{1-(1-p)^k} + \frac{1}{p} + 1$$

where  $E[C'] = (1/p) + 1$ . The deduction of  $E[C']$  is given in the proof of Theorem 4.2. □

It is also possible to apply lookup parallelism to the two-phase routing (RR + IR) and RR with ACK (RR + ACK) proposed in the previous section.

For the *two-phase routing*, we consider the case that parallelism is used in the second phase, together with IR. For the first phase, we perform RR without parallelism.

As to recursive routing with ACK, it can be parallelized similarly to RR. However, the difference is that in the case when one recursive lookup fails, in RR + ACK the originator can initiate another one



to replace it after a short timeout  $T_h$ , but in pRR, the originator does nothing for the failed lookup and reinitiates all of them when there is no reply for a timeout period of  $T_l$ .

The results about the performance of the *parallel two-phase routing* ( $RR + pIR$ ) and *parallel recursive routing with ACK* ( $pRR + ACK$ ) are shown in Theorems 5.3 and 5.4.

**Theorem 5.3.** For  $RR + pIR$ , the expected lookup latency is given by

$$E[W_{RR+pIR}] = (l + 1)\Delta p^l + (T_l + E[W_{pIR}](1 - p^l))$$

and the expected message overhead is given by

$$E[C_{RR+pIR}] = (l + 1)p^l + \left( \frac{lp^{l+1} - (l + 1)p^l + 1}{(1 - p)^2} + E[C_{pIR}] \right) (1 - p^l)$$

where  $E[W_{pIR}]$  and  $E[C_{pIR}]$  are given in Theorem 5.2.

*Proof.* The proof is similar to Theorem 4.3, except that the second phase is replaced by pIR. Therefore,  $E[W_{RR+pIR}] = (l + 1)\Delta P(X = 0) + (T_l + E[W_{pIR}])P(X = 1)$  and  $E[C_{RR+pIR}] = (l + 1)P(X = 0) + (E[M] + E[C_{pIR}])P(X = 1)$ , where  $X$  is the number of timeouts of RR before initiating IR.  $X$  can only be 0 and 1, and  $P(X = 0) = p^l$ ,  $P(X = 1) = 1 - p^l$ .  $\square$

**Theorem 5.4.** For  $pRR + ACK$ , the expected lookup latency is given by

$$E[W_{pRR+ACK}] = (l + 1)\Delta + \frac{T_h}{p} \sum_{i=0}^l i[g(i)^k - g(i + 1)^k]$$

and the upper bound of the expected message overhead is given by

$$E[C_{pRR+ACK}] \leq 2kl + k \frac{1 - p}{p} \sum_{i=0}^l i[g(i)^k - g(i + 1)^k] + (k - 1)(l + 1)$$

where  $g(i) = \sum_{j=i}^l \binom{l}{j} p^{l-j} (1 - p)^j$ .

*Proof.* For the  $m$ th recursive lookup among  $k$  concurrent lookups ( $m = 1, \dots, k$ ), we denote  $X_m$  as the number of RR failures experienced by it,  $0 \leq X_m \leq l$ . The PMF of  $X_m$  is given by  $P(X_m = i) = \binom{l}{i} p^{l-i} (1 - p)^i$  ( $i = 0, 1, \dots, l$ ).  $X_1, X_2, \dots, X_k$  are assumed to be independent of each other. Define  $Y = \min\{X_1, X_2, \dots, X_k\}$ , and thus

$$\begin{aligned} P(Y = i) &= P(Y \geq i) - P(Y \geq i + 1) = \prod_{m=1}^k P(X_m \geq i) - \prod_{m=1}^k P(X_m \geq i + 1) \\ &= [P(X_m \geq i)]^k - [P(X_m \geq i + 1)]^k \end{aligned}$$

Denoting  $g(i) = P(X_m \geq i)$ , we have  $P(Y = i) = g(i)^k - g(i + 1)^k$  and  $g(i) = \sum_{j=i}^l P(X_m = j) = \sum_{j=i}^l \binom{l}{j} p^{l-j} (1 - p)^j$ . The left proof is similar to Theorem 4.4. The expected lookup latency is given by

$$E[W_{pRR+ACK}] = \sum_{i=0}^l E[W(i)]P(Y = i) = [(l - i + 1)\Delta + i(T_h + W^*)]P(Y = i)$$



$W^*$  is the expected latency for the originator to reach the next hop in the case of failure, and  $W^* = \Delta + ((1 - p)/p)T_h$  (given in the proof of Theorem 4.4). After reduction, we have

$$E[W_{pRR+ACK}] = (l + 1)\Delta + \frac{T_h}{p} \sum_{i=0}^l i P(Y = i) = (l + 1)\Delta + \frac{T_h}{p} \sum_{i=0}^l i [g(i)^k - g(i + 1)^k]$$

As to the message overhead, it is complex to give the closed-form solution. However, an upper bound can be provided. When the first lookup arrives at the destination, the other lookups are either in progress or in the failure state. In both cases, they generate no more messages than the first arriving lookup. For the lookups in the failure state, the originator will not launch them again; for the lookups in progress, the originator has no control over them and they continue to be forwarded until they fail or reach the destination. In the worst case, each lookup in progress will generate another  $l + 1$  messages. Denote  $C_Y$  as the message overhead caused by the first arriving lookup, so the upper bound for the total message overhead can be given by  $E[C_{pRR+ACK}] \leq kC_Y + (k - 1)(l + 1)$ . Here,  $C_Y$  can be calculated by  $C_Y = \sum_{i=0}^l E[C(i)]P(Y = i)$ , where  $E[C(i)] = (l + 1 - i) \times 2 + iC^*$ .  $C^*$  is the expected overhead for the originator to reach the next hop and its value is given by  $C^* = 1 + 1/p$  (see Theorem 4.4 for the proof). After reduction, we can obtain the result. □

Based on the above theorems, we show the results of applying lookup parallelism to different lookup strategies in Figure 7. In Figure 7(a), it is observed that, under a high-churn environment ( $p = 0.3$ ), lookup parallelism effectively reduces the lookup latency for all the considered lookup strategies. However, even after being parallelized, the latency of RR is still very high. It is also interesting to find that, although IR is not as good as RR + ACK when there is no parallelism ( $k = 1$ ), after being equipped with lookup parallelism, IR can achieve lower latency than RR + ACK, and perform the best among the considered lookup strategies. This indicates that IR is more suitable to being parallelized.

As to the overhead, from Figure 7(b), we find that the overhead of pRR is much higher than the overhead of the other three, but it has a very slow increase when we increase the parallelism degree  $k$ . This is because, in spite of the increase of  $k$ , the number of retrying times by the originator is reduced at the same time due to parallelism (based on Figure 7(a)), and thus the total overhead is only increased smoothly. For pIR, RR + pIR and with pRR + ACK, the message overhead is much lower and increases almost linearly with an increase of the parallelism degree  $k$ . Note that, in Figure 7(a), the curve of pRR + ACK shows the upper bound of the message overhead.

Figure 7(c) plots the expected lookup latency of different parallel lookup strategies under a low-churn environment ( $p = 0.9$ ). The effectiveness of lookup parallelism is not so obvious as in a high-churn environment. Only the latency of pRR has a large reduction, while the latency of the other three has not much reduction. When  $k \geq 4$ , there is almost no latency reduction for all four lookup strategies. This means that lookup parallelism is not of much use under a low churn rate. As to the message overhead, we can observe from Figure 7(d) that the overhead of all four strategies increases linearly with an increase of the parallelism degree.

In summary, under a high-churn environment, lookup parallelism can effectively reduce the lookup latency for all of the considered lookup strategies. However, during design selection, pRR is not a suitable candidate owing to its high latency and overhead. pIR, RR + pIR and pRR + ACK are comparably good candidates, while pIR performs slightly better than RR + pIR and pRR + ACK. Under a low-churn environment, the effectiveness of lookup parallelism is only obvious for pRR.

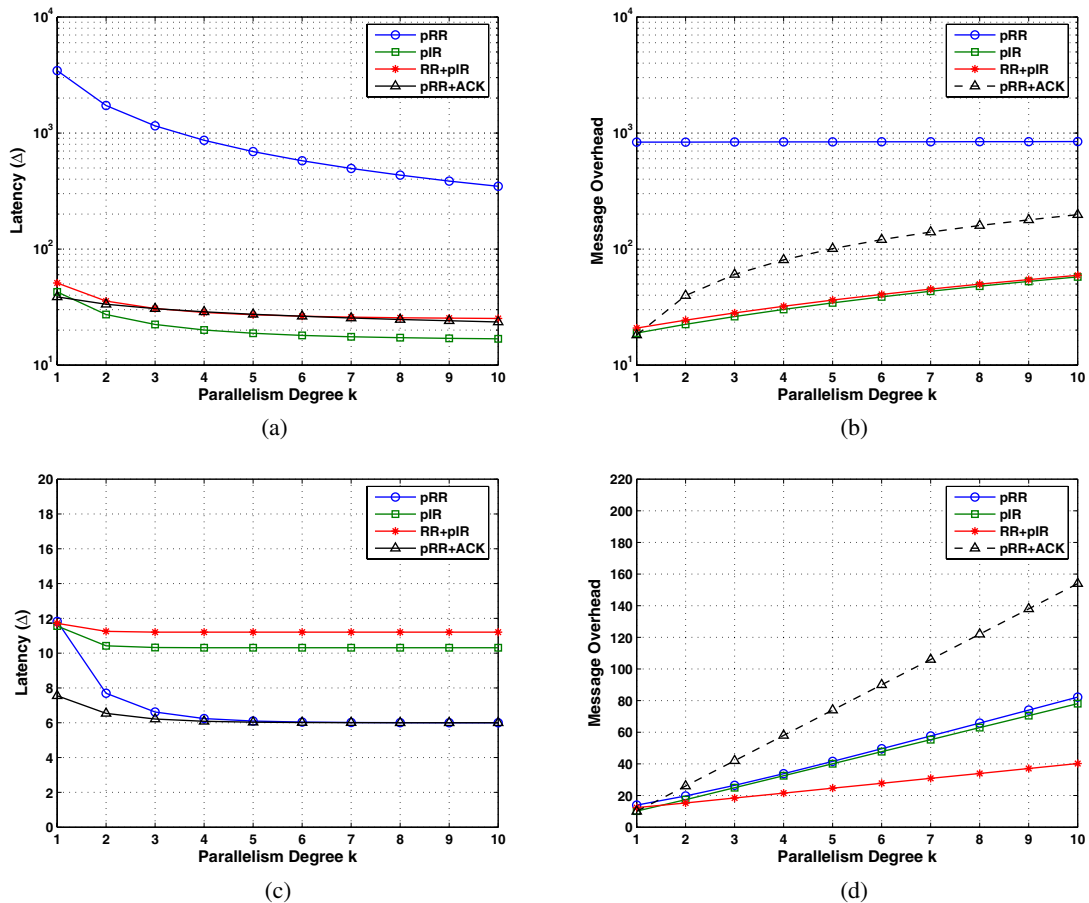


Figure 7. Performance comparison among pRR, pIR, RR + pIR and pRR + ACK: (a) expected lookup latency under a high-churn environment ( $p = 0.3$ ); (b) expected message overhead under a high-churn environment ( $p = 0.3$ ); (c) expected lookup latency under a low-churn environment ( $p = 0.9$ ); (d) expected message overhead under a low-churn environment ( $p = 0.9$ ).

For pIR, RR + pIR and pRR + ACK, the latency reduction is slight. From the perspective of designers, all four strategies can be taken as candidates, while pRR + ACK performs slightly better.

From the observation in Figure 7, we also clearly see that there exists a tradeoff between the latency  $E[W]$  and the overhead  $E[C]$  while lookup parallelism is adopted. The selection of the optimal parallelism depends on the balance between these two factors during design consideration. In Section 7, we will show how to configure the optimal parallelism degree by a specific example.

In addition to the above four parallel lookup strategies, there also exist other ways to parallelize the lookup. For instance, in the original pRR, only the originator can initiate multiple lookups, while



the intermediate nodes are just allowed to forward the lookup to one address in the next hop. It is also possible to change the behavior of intermediate nodes, and allow them to forward the lookup to multiple addresses in the next hop. Another example is that, in the original RR + pIR, lookup parallelism is only used in the second phase with IR. We can also apply the lookup parallelism in the first phase together with RR. Anyway, the performance of these variants can be analyzed similarly. Owing to space limits, we will not include all of them here.

## 6. LOOKUP KEY REPLICATION

Lookup key replication is another important aspect in improving the lookup performance under churn. If each  $\langle key, data \rangle$  pair is only published on exactly one node, it may be problematic under churn. Considering that, if one node leaves the system without migrating its keys to other nodes, the keys stored on that node will be lost permanently. Thus, the following lookups towards the keys on that node will also fail.

To handle the loss of lookup performance due to churn, the most commonly used approach is replication, in which one  $\langle key, data \rangle$  pair is duplicated in multiple nodes. By replication, we can guarantee the lookup to successfully find one replicated key with high probability.

In spite of the fact that replication has already been adopted in practical implementations (e.g., Chord [1], Kademia [17] and DHash [9]), some basic questions still require our consideration, for example, how many replicas are enough to guarantee a certain level of reliability? And do we require repair mechanisms to further improve the robustness to failure? We refer to the above decisions as the ‘*replication policy*’ and the number of replicas as the ‘*replication factor*’.

Two general replication policies are studied in this paper, and they differ in whether a repair mechanism is adopted. In the following, we will analyze them in detail.

### 6.1. Replication without repair mechanism

When publishing a  $\langle key, data \rangle$  pair, the pair is replicated at multiple nodes, each of which keeps only one copy; later, when attempting to retrieve the data associated with a given key, we should lookup these replica places and the lookup can be successful provided that there is at least one copy alive. During implementation, the process of replication and retrieval may be different for different DHTs, for example, we can use the successor nodes to place the replicas; moreover, it is also possible to use hashing to determine replica places.

However, no repair will be performed for the failed replica, and the whole replica system dies when all the replicas fail. In that situation, the following lookups for the replicated key fail accordingly.

Here, we adopt the lifetime of a replicated key  $T$ , instead of the reliability probability (i.e. ‘number of nines’), as the metric to evaluate the key reliability. Given the node lifetime distribution, we can deduce the expected lifetime of the replicated key accordingly.

For a replicated key without repair, we have the following result.

**Theorem 6.1.** *Given the replication factor  $r$ , the expected lifetime of a replicated key without repair is given by*

$$E[T] = \sum_{i=1}^r \binom{r}{i} \frac{(-1)^{i+1}}{\lambda i}$$



for an exponential lifetime distribution and

$$E[T] = \sum_{i=1}^r \binom{r}{i} (-1)^i \frac{\beta}{i(1-\alpha) + 1}$$

for a Pareto lifetime distribution with  $\alpha > 2$ .

*Proof.* Suppose that the key is replicated at  $r$  existing live nodes. Their residual lifetimes  $R_1, R_2, \dots, R_r$  satisfy the distribution  $F_R(x)$  given in Equation (1), and the lifetime of the replicated key  $T$  can be represented by the maximum of  $R_1, R_2, \dots, R_r$ . This is because the key lifetime is determined by the replica with the longest residual lifetime.  $T = \max\{R_1, R_2, \dots, R_r\}$  is also a random variable, and its complementary cumulative distribution function (CCDF) is given by  $F_T^c(x) = P(T > x) = P(\max\{R_1, R_2, \dots, R_r\} > x) = 1 - [F_R(x)]^r$ .

From its CCDF, we can directly obtain the expected value of  $T$ ,

$$E[T] = \int_0^{+\infty} F_T^c(x) dx = \int_0^{+\infty} [1 - [F_R(x)]^r] dx$$

Based on the previous results in Section 3 (for an exponential lifetime,  $F_R(x) = 1 - e^{-\lambda x}$ , and for a Pareto distribution,  $F_R(x) = 1 - (1 + (x/\beta)^{1-\alpha})^{-1}$ ), we can deduce the theorem.  $\square$

## 6.2. Replication with repair mechanism

The replication process is similar to the above, but a repair mechanism is installed. Periodically, some responsible node (maybe the key owner or the replica) checks the liveness of other replicas. In the case when one replica is found to be dead, the responsible node will perform a repair by recopying the *(key, data)* to a new node. By this means, we keep the number of replicas unchanged; however, if the responsible node is dead, a repair will no longer be performed. The repair rate is denoted by  $\mu = 1/T_{rp}$ , where  $T_{rp}$  is the period between two consecutive repairs performed by the responsible node. For a replicated key with repair, our results are as follows.

**Theorem 6.2.** *Given the replication factor  $r$  and the repair rate  $\mu$  (i.e.  $1/T_{rp}$ ), the expected lifetime of a replicated key with repair is given by*

$$E[T] = \frac{1}{\prod_{z=1}^{r-1} z\theta/(z\theta + \mu)} \left[ \frac{1}{r\theta} + \left( 1 - \prod_{z=1}^{r-1} \frac{z\theta}{z\theta + \mu} \right) \frac{1}{\mu} \right]$$

where  $\theta = \lambda$  for an exponential lifetime distribution and  $\theta = (\alpha - 2)/\beta$  for a Pareto lifetime distribution with  $\alpha > 2$ .

*Proof.* Let  $\theta$  be the departure rate of peers under steady state. By applying Little's law [36], we have  $\theta = \lambda$  for an exponential lifetime distribution and  $\theta = (\alpha - 2)/\beta$  for a Pareto lifetime distribution.

The key replica system can be modeled by a continuous time Markov chain (CTMC) with  $r + 1$  states: the system is in state  $i$  when there are  $r + 1 - i$  replicas alive, and the CTMC state transition diagram is shown in Figure 8.

Initially, the system is in state 1, but it will enter state  $r + 1$  eventually. The state  $r + 1$  is the absorbing state, and all other states are transient states. What we care about is the expected time for the system to enter absorbing state  $r + 1$ , which is also the lifetime of the replica system.

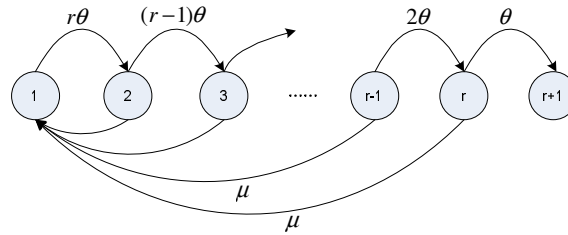


Figure 8. Markov state transition.

The transition matrix of the embedding of CTMC [37] is an  $(r + 1) \times (r + 1)$  matrix, which is given by

$$\mathbf{P} = \left[ \begin{array}{cccc|c}
 0 & 1 & 0 & \dots & 0 \\
 \frac{\mu}{(r-1)\theta + \mu} & 0 & \frac{(r-1)\theta}{(r-1)\theta + \mu} & \dots & 0 \\
 \frac{\mu}{(r-2)\theta + \mu} & 0 & 0 & \dots & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots \\
 \frac{\mu}{\theta + \mu} & 0 & \vdots & 0 & \frac{\theta}{\theta + \mu} \\
 \hline
 0 & 0 & \dots & 0 & 1
 \end{array} \right] = \left[ \begin{array}{c|c}
 \mathbf{Q} & * \\
 \hline
 \mathbf{0} & \mathbf{1}
 \end{array} \right]$$

Here,  $Q$  is an  $(r \times r)$  matrix including all the transient states. Based on  $Q$ , we can derive the fundamental matrix [37] of  $P$ , denoted by  $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$ .

The element  $N_{ij}$  of  $N$  gives the expected number of times that the process is in the transient state  $j$  if it is started in the transient state  $i$ . In our case, the initial system state is 1, therefore we only need to compute  $N_{1i}$ .

Let  $A = I - Q$ , so  $N = A^{-1} = (1/\det(A))(C_{ij})^T$ , where  $\det(A)$  is the determinant of  $A$ ,  $C_{ij}$  is the matrix cofactor of  $A$  and  $N_{ij} = C_{ji}/\det(A)$ . It is easy to derive  $C_{i1}$  and  $\det(A)$  based on  $A$ , and we have

$$C_{11} = 1; \quad C_{21} = 1; \quad C_{i1} = \prod_{z=r+2-i}^{r-1} \frac{z\theta}{z\theta + \mu} \quad (i = 3, 4, \dots, r)$$

and

$$\det(A) = \sum_{i=1}^r A_{i1} \times C_{i1} = 1 + \frac{(-\mu)}{(r-1)\theta + \mu} + \sum_{i=3}^r \frac{(-\mu) \prod_{z=r+2-i}^{r-1} z\theta / (z\theta + \mu)}{(r-i+1)\theta + \mu} = \prod_{z=1}^{r-1} \frac{z\theta}{z\theta + \mu}$$

Define  $t_i$  to be the average duration staying at state  $i$  per time,

$$t_1 = \frac{1}{r\theta}; \quad t_i = \frac{1}{(r-i+1)\theta + \mu} \quad (i = 2, 3, \dots, r)$$



Then the expected elapsed time before entering the absorbing state can be given by

$$\begin{aligned}
 E[T] &= \sum_{i=1}^r t_i \times N_{1i} = \frac{1}{\det(A)} \sum_{i=1}^r t_i \times C_{i1} \\
 &= \frac{1}{\det(A)} \left[ \frac{1}{r\theta} + \frac{1}{(r-1)\theta + \mu} + \sum_{i=3}^r \frac{\prod_{z=r+2-i}^{r-1} z\theta/(z\theta + \mu)}{(r-i+1)\theta + \mu} \right] \\
 &= \frac{1}{\det(A)} \left[ \frac{1}{r\theta} + (1 - \det(A)) \frac{1}{\mu} \right] \\
 &= \frac{1}{\prod_{z=1}^{r-1} z\theta/(z\theta + \mu)} \left[ \frac{1}{r\theta} + \left( 1 - \prod_{z=1}^{r-1} \frac{z\theta}{z\theta + \mu} \right) \frac{1}{\mu} \right]. \quad \square
 \end{aligned}$$

In Figure 9, we compare the two replication policies under an exponential lifetime distribution and a Pareto lifetime distribution respectively. For each distribution, we consider two lifetime expectations, i.e.  $E[L] = 1$  h and  $E[L] = 10$  h. The former has a high churn rate, while the later has a low churn rate. For a given lifetime expectation, the exponential distribution has a fixed  $\lambda$ , but for the Pareto distribution  $\alpha$ ,  $\beta$  are not fixed. In our study, we consider different heavy-tailed degrees by varying the value of  $\alpha$ . The distribution exhibits a more heavy-tailed degree when  $\alpha$  is smaller.

From Figure 9, we can observe that whether a repair mechanism is necessary depends largely on the node lifetime distribution, instead of the system churn rate. Under an exponential lifetime distribution (see Figures 9(a) and (b)), it is hard to achieve a long lifetime simply by replication. Without a repair mechanism, even with 30 replicas, the expected lifetime of the replicated key is still rather short under a low-churn environment (i.e.  $E[L] = 10$  h). However, the situation is a bit different for a Pareto lifetime distribution. In Figures 9(c) and (d), it can be found that the repair mechanism is not indispensable for a Pareto lifetime distribution with a very heavy tail. Only by replication can we achieve a rather long lifetime for the replicated key. This is true even in a high-churn environment. For instance, in the parameter setting (i.e.  $E[L] = 1$  h,  $\alpha = 2.01$ ,  $\beta = 1.01$ ), without a repair mechanism, by using 15 replicas, the expected lifetime of the replicated key is longer than 61 days. However, if we want to increase the lifetime of the replicated key to the level of years, it is better to install a repair mechanism. With the linear increase of repair rate, the lifetime of the replicated key is increased almost exponentially.

In the above, our analysis shows that a repair mechanism is recommended for improving the lifetime of the replicated key under most situations. As to the selection of the replication factor and repair rate, however, it is application-specific. We should consider some practical issues during design, such as data type, data lifetime, repair cost, etc. In different DHT systems, the data stored under the  $\langle key, data \rangle$  pair may be a data fragment (e.g., storage system), a file pointer (e.g., file-sharing system) or a metadata item (e.g., information management system). Also the data itself has a lifetime, and it is not always necessary to keep the key as long as possible. For example, in a file-sharing system, in the case when one node leaves the system, it becomes unnecessary to continue to repair the DHT keys published by that node. What is more, the repair cost should also be taken into consideration. In the case when the repair cost is expensive, it is better to do a lazy repair; however, if the repair cost is cheap, an eager repair is feasible. Based on the design goal, our theoretical results in the above section can help to determine the appropriate replication factor and repair rate. In Section 7, we will use the Kad file-sharing system as an example to show how to perform the configuration.

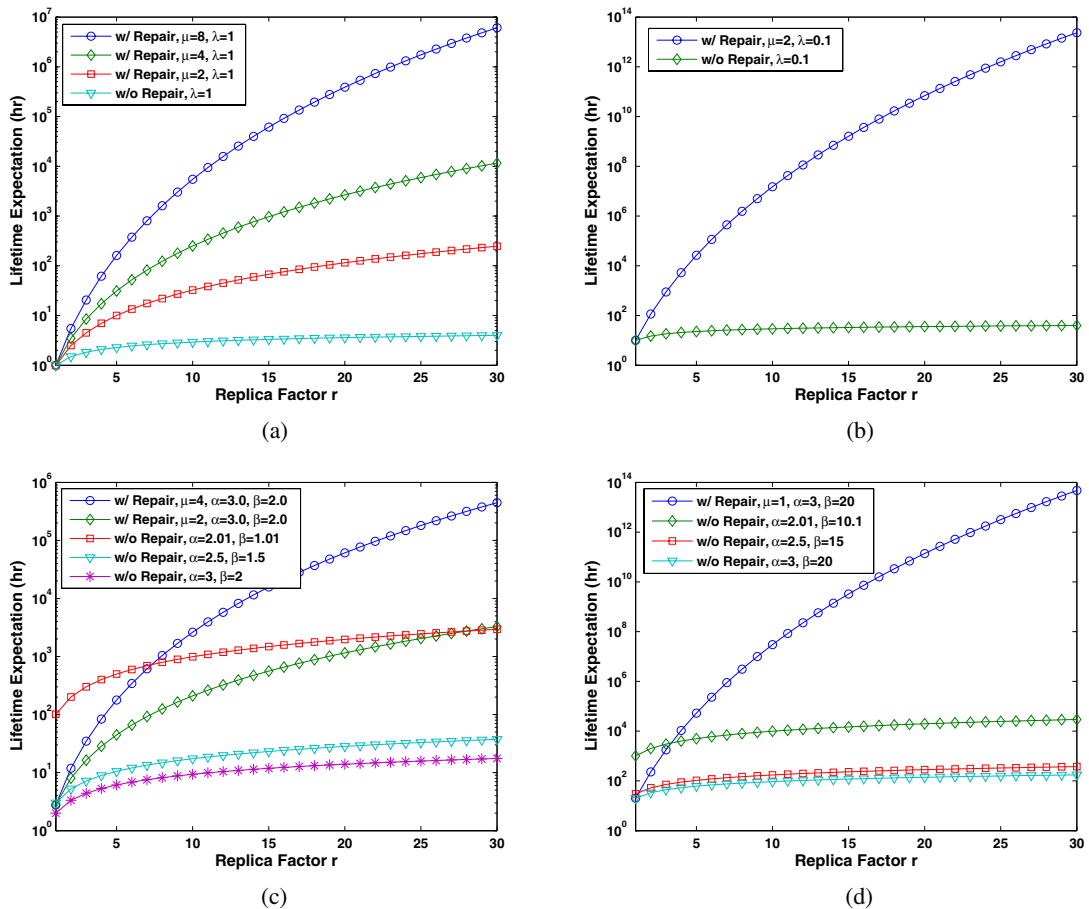


Figure 9. Expected lifetime of a replicated key: (a) exponential lifetime distribution with  $E[L] = 1$  h; (b) exponential lifetime distribution with  $E[L] = 10$  h; (c) Pareto lifetime distribution with  $E[L] = 1$  h; (d) Pareto lifetime distribution with  $E[L] = 10$  h.

### 7. CASE STUDY

In this section, we take one real DHT system, Kad [7], as a case study to show how to apply our analysis to optimize DHT systems. Kad is the first widely deployed, Kademlia-based [17], file-sharing system and adopts the pIR strategy. According to our results in Section 5, pIR performs the best among four lookup strategies under high-churn environments.

In Kad, the node lifetime distribution follows a heavy-tailed distribution with the expectation of 2.71 h (see [19]), which can be approximated by a Pareto distribution with  $\alpha = 2.107$ ,  $\beta = 3$ . The time



Table II. Expected latency and overhead under different parallelism degrees.

Parallelism degree	Lookup latency $E[W_{\text{pIR}}] (\times \Delta)$	Message overhead $E[C_{\text{pIR}}]$
$k = 1$	6.0411	6.0205
$k = 2$	6.0139	9.9798
$k = 3$	6.0137	13.9661
$k = 4$	6.0137	17.9524
$k = 5$	6.0137	21.9388

for neighbor replacement  $S$  is about 4–19 min, and the routing path length  $l$  is about 3 due to the random bits improvement [27].

Next, we first show how to configure the optimal parallelism degree in Kad. Based on the equations in Section 3 and Theorem 5.2, we get the numerical results as shown in Table II.

It can be observed that, when  $k \geq 2$ , the latency reduction is very slight compared with the incurred overhead. For the current Kad system, it is a good choice to set  $k$  as 2 or 3, if the message overhead is not a big issue. To further improve the parallelism degree is almost meaningless. In [27], Stutzbach and Rejaie reached a similar conclusion ( $k = 3$ ) by empirical study of the Kad system. In the above, we have shown how to select a proper parallelism degree for the pIR. In other systems where different parallel lookup strategies are adopted, we can configure their parallelism degree in a similar way based on our theorems.

In the Kad system, key replication is also adopted to defend against the loss of lookup performance caused by churn. The data in the  $\langle \text{key}, \text{data} \rangle$  pair is a pointer to the file holder. In the current implementation, the Kad algorithm produces 19 replicas on average, and adopts a kind of soft-state protocol to maintain the replicated keys. The file holder republishes the key every 5 h for common files, and the key will become expired if no refresh is performed. The use of a soft-state protocol is due to the transient nature of peers. In the case when the file holder leaves the system, it is no longer necessary for the system to maintain the keys published by the file holder.

In fact, the republishing operation can also be regarded as a kind of repair operation, except that there is only one responsible node, which is the file holder itself. In the following, we will show how to select a proper replication policy and its corresponding replication factor. Based on previous theorems in Section 6, we can compute the expected lifetime of a replicated key under two different replication policies as shown in Table III.

From the results, we find that even without repair, Kad can still guarantee the existence of the key about 17.52 days (which is much greater than the node lifetime expectation  $E[L] = 2.71$  h) with 19 replicas. Although the lifetime of the replicated key without repair is not long enough, it is safe for us to perform the repair lazily. When equipped with a lazy republishing mechanism (e.g.,  $T_{\text{rp}} = 5$  h), the lifetime is further improved to over 50 years (19 783.15 days). This means that, if a node  $v$  remains in the system and performs republishing periodically, the lookups towards the keys published by node  $v$  can be successful with a very high probability.



Table III. Expected lifetime of the replicated key under different replication policies.

Replication factor	Replication without repair	Replication with repair		
		$T_{rp} = 1$ h	$T_{rp} = 5$ h	$T_{rp} = 10$ h
$r = 1$	1.16 days	1.16 days	1.16 days	1.16 days
$r = 2$	2.23 days	17.81 days	4.95 days	3.34 days
$r = 3$	3.24 days	183.11 days	14.51 days	6.82 days
$r = 4$	4.23 days	<b>1455.04 days</b>	34.94 days	11.85 days
$r = 5$	5.19 days	9538.83 days	73.99 days	18.67 days
$r = 6$	6.13 days	53 700.32 days	142.70 days	27.50 days
$r = 7$	7.06 days	266 797.03 days	256.13 days	38.59 days
$r = 8$	7.98 days	1 193 175.81 days	434.14 days	52.13 days
$r = 9$	8.88 days	4 875 817.51 days	702.26 days	68.35 days
$r = 10$	9.78 days	18 419 755.75 days	1092.52 days	87.45 days
$r = 11$	10.66 days	64 934 292.60 days	<b>1644.41 days</b>	109.64 days
$r = 12$	11.54 days	215 245 161.67 days	2405.81 days	135.12 days
$r = 13$	12.41 days	675 170 740.71 days	3434.03 days	164.08 days
$r = 14$	13.28 days	2 014 795 284.00 days	4796.82 days	196.72 days
$r = 15$	14.14 days	5 745 897 822.48 days	6573.50 days	233.23 days
$r = 16$	14.99 days	15 721 415 304.90 days	8856.04 days	273.80 days
$r = 17$	15.84 days	41 410 003 510.32 days	11 750.23 days	318.60 days
$r = 18$	16.68 days	105 314 332 666.14 days	15 376.92 days	367.84 days
$r = 19$	<b>17.52 days</b>	259 282 661 905.53 days	<b>19 873.15 days</b>	421.68 days
$r = 20$	18.36 days	619 397 485 609.87 days	25 393.53 days	480.30 days
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$r = 32$	$\vdots$	$\vdots$	$\vdots$	<b>1619.98 days</b>

However, it is somewhat wasteful to create so many replicas. In the current Kad system, there exist few peers whose lifetimes can reach the year level. In fact, the probability that a node's lifetime  $L$  is more than 50 days is very small according to the node lifetime distribution. Therefore, we can choose a smaller replication factor; for example, with 11 replicas and  $T_{rp} = 5$  h, the expected lifetime of the replicated key reaches 1644.41 days ( $\approx 4.5$  years), which is also long enough to guarantee a high probability of successful lookup. By increasing the repair rate, we can further reduce the replication factor while guaranteeing the same level of key reliability. In the case when  $T_{rp} = 1$  h, with four replicas, the expected lifetime of the replicated key can also reach 1455.04 days ( $\approx 4$  years). Given a target level of reliability, there are multiple configurations that can satisfy the reliability requirement.

However, in reality, we should consider other practical issues, such as bandwidth cost, storage cost, etc. For instance, considering the following three configurations: (i)  $T_{rp} = 1$  h,  $r = 4$  replicas; (ii)  $T_{rp} = 5$  h,  $r = 11$  replicas; (iii)  $T_{rp} = 10$  h,  $r = 32$  replicas. They can achieve the comparable long existence of the replicated key based on Table III. The comparison of their storage costs is  $4 : 11 : 32$ , and the comparison of their bandwidth cost incurred by key republishing is  $\frac{4}{1} : \frac{11}{5} : \frac{32}{10} = 4 : 2.2 : 3.2$ .



In the Kad system, as the data stored in the  $\langle key, data \rangle$  pair are a pointer to the file holder, instead of the real file, the storage cost is not a big issue. On the contrary, it is more important to reduce the bandwidth cost of republishing, for there exists a large volume of files in the system to be published periodically. Therefore, among the three, the second configuration is the best, as it can minimize the bandwidth cost. Similarly, we can compare other configurations analytically and select the optimal one.

In the above, what we have described is an example so that designers can get an idea of how to configure the parameters properly. For a specific application, designers should take the application requirements into account to make the decision. Moreover, our Theorems 6.1 and 6.2 can give designers great help in choosing the proper parameters.

## 8. CONCLUSION

In this paper, our objective has been to provide directions for the DHT designers to optimize the lookup performance under churn. We have analytically studied three important aspects in handling churn in DHT systems. For lookup strategy, two typical lookup strategies, RR and IR, have been studied in terms of their latency and overhead. Our theoretical results enable designers to know exactly which strategy is better under a given system setting. To make the lookup strategy perform the best in the whole range of churn rates, we have explored the existence of their enhancements. Both the two-phase routing strategy and the RR with ACK strategy are demonstrated to be better choices. Then, we have studied the performance of parallel lookup and the selection of the optimal degree of parallelism. Later, we have analyzed two key replication policies to handle data key loss due to churn, and have shown whether a repair mechanism is necessary for a given environment. As to the selection of the replication factor and repair rate, it is application-specific and we have taken Kad as an example to show how to perform the proper configuration. In our future work, we plan to investigate other aspects of optimizing the DHT lookup performance under churn. This is meaningful for making DHT systems more widely adopted.

## REFERENCES

1. Stoica I *et al.* Chord: A scalable peer-to-peer lookup service for internet applications. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, San Diego, CA, August 2001. ACM Press: New York, 2001; 149–160.
2. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, San Diego, CA, August 2001. ACM Press: New York, 2001; 161–172.
3. Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proceedings of the IFIPL ACM International Conference on Distributed Systems Platforms (Middleware'01)*, Heidelberg, November 2001. Springer: Berlin, 2001; 329–350.
4. Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD, Kubiatowicz J. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communication* 2004; **22**(1):41–53.
5. Cai M, Frank M, Chen J, Szekely P. Maan: A multi-attribute addressable network for Grid information services. *Proceedings of the 4th International Workshop on Grid Computing (Grid'03)*, Phoenix, AZ, 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003; 184–191.
6. Teo YM, March V, Wang X. A DHT-based Grid resource indexing and discovery. *Proceedings of the Singapore-MIT Alliance Annual Symposium*. NUS: Singapore, 2005.
7. Emule Project Web Site. <http://www.emule-project.net/> [December 2006].



8. Ramasubramanian V, Sirer E. The design and implementation of a next generation name service for the Internet. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2004. ACM Press: New York, 2004; 331–342.
9. Dabek F, Li J, Sit E, Robertson J, Kaashoek MF, Morris R. Designing a DHT for low latency and high throughput. *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association: Berkeley, CA, 2004; 85–98.
10. Kubiawicz J *et al.* Oceanstore: An architecture for global-scale persistent storage. *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems Services (ASPLOS)*, November 2000. ACM Press: New York, 2000; 190–201.
11. Coral Content Distribution Network Web Site. <http://www.coralcdn.org/> [December 2006].
12. Rhea S, Geels D, Roscoe T, Kubiawicz J. Handling churn in a DHT. *Proceedings of the USENIX Annual Technical Conference*, 2004. USENIX Association: Berkeley, CA, 2004.
13. Lam SS, Liu H. Failure recovery for structured P2P networks: Protocol design and performance evaluation. *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, 2004. ACM Press: New York, 2004; 199–210.
14. Kuhn F, Schmid S, Wattenhofer R. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. *Proceedings of the 4th International Workshop on Peer-To-Peer Systems (IPTPS)*, 2005. Springer: Berlin, 2005; 13–23.
15. Li J, Stribling J, Morris R, Kaashoek MF. Bandwidth-efficient management of DHT routing tables. *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, 2005. USENIX Association: Berkeley, CA, 2005.
16. Leong B, Liskov B, Demaine E. Epichord: Parallelizing the Chord lookup algorithm with reactive routing state management. *Proceedings of the IEEE International Conference on Networks (ICON'04)*, 2004. IEEE Press: Piscataway, NJ, 2004.
17. Maymounkov P, Mazières D. Kademia: A peer-to-peer information system based on the XOR metric. *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002. Springer: Berlin, 2002; 53–65.
18. Saroiu S. Measurement and analysis of Internet content delivery systems. *Doctoral Dissertation*, University of Washington, December 2004.
19. Stutzbach D, Rejaie R. Characterizing churn in peer-to-peer networks. *Technical Report CIS-TR-05-03*, University of Oregon, 2005.
20. Napster Web Site. <http://www.napster.com/> [December 2006].
21. Gnutella Project Web Site. <http://rfc-gnutella.sourceforge.net/> [December 2006].
22. Bittorrent Web Site. <http://www.bittorrent.com/> [December 2006].
23. Stribling J, Yoshikawa C. All-pairs ping dataset. <http://ping.ececs.uc.edu/ping/> [December 2006].
24. PlanetLab Project Web Site. <http://www.planet-lab.org/> [December 2006].
25. Li J, Stribling J, Morris R, Kaashoek MF, Gil TM. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. *Proceedings of the IEEE Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2005. IEEE Press: Piscataway, NJ, 2005.
26. P2Psim Project Web Site. <http://pdos.csail.mit.edu/p2psim/> [December 2006].
27. Stutzbach D, Rejaie R. Improving lookup performance over a widely-deployed DHT. *Proceedings of the IEEE Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2006. IEEE Press: Piscataway, NJ, 2006.
28. Liben-Nowell D, Balakrishnan H, Karger D. Analysis of the evolution of peer-to-peer systems. *Proceedings of the ACM Conference on Principles of Distributed Computing (PODC)*, 2002. ACM Press: New York, 2002.
29. Krishnamurthy S, El-Ansary S, Aurell E, Haridi S. A statistical theory of Chord under churn. *Proceedings of the 4th International Workshop on Peer-To-Peer Systems (IPTPS)*, 2005. Springer: Berlin, 2005; 93–103.
30. El-Ansary S, Krishnamurthy S, Aurell E, Haridi S. An analytical study of consistency and performance of DHTs under churn. *SICS Technical Report T2004:12*, Swedish Institute of Computer Science, Stockholm, Sweden, 2004.
31. Godfrey PB, Shenker S, Stoica I. Minimizing churn in distributed systems. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2006. ACM Press: New York, 2006; 147–158.
32. Hoyland A, Rausand M. *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley: New York, 1994.
33. Prabhu N. *Stochastic Processes; Basic Theory and its Applications*. Macmillan: New York, 1965.
34. Tang C, Buco MJ, Chang RN, Dwarkadas S, Luan LZ, So E, Ward C. Low traffic overlay networks with large routing tables. *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, 2004. ACM Press: New York, 2004.
35. Jacobson V, Karels MJ. Congestion avoidance and control. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 1988. ACM Press: New York, 1988; 314–329.
36. Haverkort BR. *Performance of Computer Communication Systems*. Wiley: New York, 1998.
37. Grinstead C, Snell JL. *Introduction to Probability*. American Mathematical Society: Providence, RI, 1997.