Towards a Scalable and Robust DHT

Baruch Awerbuch[*]
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218, USA
baruch@cs.jhu.edu

Christian Scheideler
Department of Computer Science
Technical University of Munich
85748 Garching, Germany
scheideler@in.tum.de

ABSTRACT

The problem of scalable and robust distributed data storage has recently attracted a lot of attention. A common approach in the area of peer-to-peer systems has been to use a distributed hash table (or DHT). DHTs are based on the concept of virtual space. Peers and data items are mapped to points in that space, and local-control rules are used to decide, based on these virtual locations, how to interconnect the peers and how to map the data to the peers.

DHTs are known to be highly scalable and easy to update as peers enter and leave the system. It is relatively easy to extend the DHT concept so that a constant fraction of faulty peers can be handled without any problems, but handling adversarial peers is very challenging. The biggest threats appear to be join-leave attacks (i.e., adaptive join-leave behavior by the adversarial peers) and attacks on the data management level (i.e., adaptive insert and lookup attacks by the adversarial peers) against which no provably robust mechanisms are known so far. Join-leave attacks, for example, may be used to isolate honest peers in the system, and attacks on the data management level may be used to create a high load-imbalance, seriously degrading the correctness and scalability of the system.

We show, on a high level, that both of these threats can be handled in a scalable manner, even if a constant fraction of the peers in the system is adversarial, demonstrating that open systems for scalable distributed data storage that are robust against even massive adversarial behavior are feasible.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and layout*

General Terms

Algorithms, Theory

[*]Supported by NSF-ANIR 0240551, NSF-CCF 0515080, and NSF-CCR 0311795.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SPAA'06, July 30–August 2, 2006, Cambridge, Massachusetts, USA.
Copyright 2006 ACM 1-59593-262-3/06/0007 ...$5.00.

Keywords

overlay networks, peer-to-peer systems, robustness

1. INTRODUCTION

In a distributed storage system, information is distributed among multiple storage devices, simply called *nodes* in the following. To provide a basic lookup service, the following operations have to be implemented:

- Insert(d): this inserts data item d into the system.
- Lookup(name): this returns any data item d with $\mathrm{Name}(d) = \mathrm{name}$, if it exists.

Once a distributed storage system becomes large enough, one also has to deal with nodes leaving and joining the system, simply because storage devices may break down or new devices have to be added in order to maintain a desired service quality. Hence, two more operations are needed:

- Join(v): node v joins the system
- Leave(v): node v leaves the system

How can these four operations be implemented to obtain a robust and scalable distributed storage system? The most prominent approach studied in the research community is to implement a distributed hash table, or DHT. DHTs have been realized in various contexts including server-based systems such as Akamai and peer-to-peer systems such as Chord [32], CAN [23], Pastry [9], and Tapestry [34]. Most of the DHT-based systems are based on two influential papers: a paper by Plaxton, Rajaraman, and Richa on locality-preserving data management in distributed environments [22] and a paper by Karger, Lehman, Leighton et al. on consistent hashing and web caching [14]. The consistent hashing approach is a very simple and elegant approach that is based on the following rule:

Suppose that we have two random functions f and g. The function f maps the nodes randomly to real numbers in $[0,1)$, and the function g maps the data items randomly to real numbers in $[0,1)$. Every data item d is stored at the node v with a minimum distance between $f(v)$ and $g(d)$ (viewing $[0,1)$ here as a ring). It turns out that this rule has several nice features [14]:

- On expectation, every node has the same load,
- when integrating a node into or removing a node from a system of n nodes, only a $1/n$-fraction of the data has to be replaced on expectation, and
- when storing $O(\log n)$ copies of each data item, the system is robust against a constant fraction of faulty nodes.

However, all of these properties only hold if f and g are random and the names selected for the nodes and the data items are independent of f and g. Unfortunately, in a DHT, g must be a fixed hash function because otherwise it would not be possible to compute the location of the data items. Hence, it is easy for the adversary to generate many data items that all need to be stored at the same node, even if it cannot invert g. It just has to try sufficiently many names. In fact, millions of names can be quickly tested with hash functions like SHA-1.

Fortunately, f does not have to be a fixed hash function for the DHT to work. But even a truly random mapping does not protect against adversarial attacks. Suppose, for example, that every new node is mapped uniformly at random to a point in $[0, 1)$. If the adversary wants to overpopulate a certain area of the $[0, 1)$ space, say some interval I, it just needs to execute sufficiently many join and leave operations in which it keeps all nodes that made it into I in the system and removes all others for another join attempt.

DHT constructions are very vulnerable to a node and data imbalance in the virtual space since this can seriously degrade their scalability. Can we design simple protocols for the operations join, leave, insert and lookup that are *provably* robust against these attacks without restricting the openness of the system?

In this paper we show that, on a high level, this is possible. More precisely, we will show that there are scalable join and leave protocols so that for a polynomial number of join and leave requests the nodes will be evenly distributed in the $[0, 1)$ space, with high probability[1], and the honest and adversarial nodes will be well-spread so that quorums of size $O(\log n)$ can be formed to wash out any adversarial behavior violating the protocols by simple majority decision. Moreover, we will show that there are scalable and robust insert and lookup protocols so that for a polynomial number of attempts the adversary will not manage to find data names so that it can create a high request or load imbalance in the system.

1.1 Our contributions

Next we give a detailed description of our contributions. For simplicity, we assume that the number of honest nodes in the system will only change by a constant factor over time. In this way, notions like "in a polynomial number of rounds" and "with high probability" are well-defined. However, using the techniques in [3], our approach can also be extended to systems in which the number of honest nodes in the system may change in an arbitrary way over time, as long as it does not drop too rapidly.

In the following, let n be the maximum number of honest nodes in the system at any time and let ϵn for some $\epsilon < 1$ be the maximum number of nodes that the adversary can have in the system at any time. Thus, the adversary has bounded resources, but apart from that the adversary can do what it likes, such as choosing any names it likes for the data items and the nodes.

Join-leave attacks

First, we focus on making a DHT robust against join-leave attacks. More precisely, we consider the following scenario. There are n blue (or honest) nodes and ϵn red (or adversarial) nodes for some fixed constant $\epsilon < 1$. There is a rejoin operation that, when applied to node v, lets v first leave the system and then join it again from scratch. The leaving is done by simply removing v from the system and the joining is done with the help of a join operation to be specified by the system. We assume that the sequence of rejoin requests is controlled by an adversary, which is a typical assumption in the analysis of online algorithms. The adversary can only issue rejoin requests for the red nodes, but it can do this in an arbitrary adaptive manner. That is, at any time it can inspect the entire system and select whatever red node it likes to rejoin the system. Our goal is to find an *oblivious* join strategy, i.e., a strategy that cannot distinguish between the blue and red nodes, so that for *any* adversarial strategy above the following two conditions can be preserved for every interval $I \subseteq [0, 1)$ of size at least $(c \log n)/n$ for a constant $c > 0$ and any polynomial number of rounds in n:

- *Balancing condition:* I contains $\Theta(|I| \cdot n)$ nodes.
- *Majority condition:* the blue nodes in I are in the majority.

It is not difficult to see that the brute-force strategy of giving every node a new random place whenever a node rejoins will achieve the stated goal, with high probability, but this would be a very expensive strategy. The challenge is to find a join operation that needs as little randomness and as few rearrangements as possible to satisfy the two conditions. Fortunately, there is such a strategy, called the *cuckoo rule*. We first introduce some notation, and then we describe the strategy.

In the following, a *region* is an interval of size $1/2^r$ in $[0, 1)$ for some integer r that starts at an integer multiple of $1/2^r$. Hence, there are exactly 2^r regions of size $1/2^r$. A *k-region* is a region of size (closest from above to) k/n, and for any point $x \in [0, 1)$, the k-region $R_k(x)$ is the unique k-region containing x.

Cuckoo rule: If a new node v wants to join the system, pick a random $x \in [0, 1)$. Place v into x and move all nodes in $R_k(x)$ to points in $[0, 1)$ chosen uniformly and independently at random (without replacing any further nodes).

Our first main result is summarized in the following theorem.

THEOREM 1.1. *For any constants ϵ and k with $\epsilon < 1-1/k$, the cuckoo rule with parameter k satisfies the balancing and majority conditions for a polynomial number of rounds, with high probability, for any adversarial strategy within our model. The inequality $\epsilon < 1 - 1/k$ is sharp as counterexamples can be constructed otherwise.*

Hence, a constant $k > 1$ would be sufficient to prevent *adaptive* join-leave attacks of a *constant fraction* of adversarial peers. Thus, it is remarkably easy to defend an open distributed storage system against even massive join-leave attacks. The cuckoo rule allows us to use local quorum strategies in order to wash out adversarial behavior violating the protocols.

Lookup and insert attacks

Our basic strategy to handle attacks on the data layer is to use $2c - 1 = \Theta(\log n)$ one-way hash functions mapping each data item to $2c - 1$ points in the $[0, 1)$ space. These hash functions are fixed but have certain expansion properties to make it hard for the adversary to create bad sets of insert or lookup requests. In order to achieve an even load balance of the requests and the data items, we use the majority trick of Upfal and Wigderson [33]: for each insert request, store copies of the data item in at least c of the $2c - 1$ locations, and for each lookup request, access at least c locations of the data item. This indeed suffices for the correct implementation of these requests because if the copies are stored in a reliable way, then the lookup operation will always retrieve at least one copy of the data item.

Given this basic scheme, we present a scalable dynamic overlay network and robust protocols for the insert and lookup operations. In the following, U may represent the space of all names or the names that the adversary can sample in a polynomial number of time steps.

[1]By "with high probability" or "w.h.p." we mean a probability of at least $1 - 1/n$ where n is the size of the system.

THEOREM 1.2. *For any collection of lookup requests for data items out of a set U of polynomial size with one request per node, the lookup protocol can serve all of these requests correctly and reliably in polylogarithmic time so that each node is passed by at most $O(\log^5 n)$ requests.*

Notice that the upper bound is *guaranteed* for *any adaptively* chosen set of lookup requests, including data items with multiple lookup requests. The hash functions just need to be selected so that they satisfy certain expansion properties. Also, notice that the adversary cannot modify or delete a request in transit since we use quorum strategies. Certainly, the bound is still too high for practical purposes, but the best bound previously known for adaptively chosen lookup requests in overlay networks is the trivial linear bound. In light of this, our result is an exponential improvement, and an interesting problem for future research will certainly be whether further improvements are possible. For insert requests we obtain a similar result.

THEOREM 1.3. *For any collection of insert requests for data items out of a set U of polynomial size with one request per node, the insert protocol can serve all of these requests in polylogarithmic time so that each node is passed by at most $O(\log^5 n)$ requests. Moreover, the maximum amount of copies to be stored by any node to serve all of the requests is bounded by $O(\log^3 n)$.*

The $O(\log^3 n)$ bound is just an $O(\log n)$ factor away from the optimal number of copies per node since we need to store $\Theta(n \log n)$ copies in $\Theta(n/\log n)$ quorum regions containing $\Theta(\log n)$ nodes each.

Prerequisites

An important prerequisite for our join and leave operations to work correctly is a distributed random number generator that can generate an unbiased random number even under the influence of a constant fraction of adversarial nodes, and an important prerequisite for our insert and lookup operations to work correctly is that one-way hash functions with certain expansion properties are available so that the adversary has no other way then sampling names in order to design malicious collections of names. It is commonly believed that one-way hash functions exist though no formal proof has been found yet. But we will at least prove that random hash functions will have expansion properties, w.h.p., that are good enough for our results to hold. A distributed random number generator sufficient for our purposes can be built on top of existing verifiable-secret-sharing (VSS) protocols. In fact, we have developed a distributed random number generator based on the $\lfloor \frac{n-1}{4} \rfloor$-VSS protocol in [12] that we include in this paper for completeness. This generator may be used by the $\Theta(\log n)$ nodes in any quorum in our DHT to correctly and efficiently generate random numbers under the assumption that the honest nodes are in a sufficient majority in that quorum.

1.2 Previous work

In the area of peer-to-peer systems, work on robustness in the context of overlay network maintenance has mostly focused on how to handle a large fraction of faulty nodes (e.g., [2, 25, 32]) or churn, that is, peers frequently enter and leave the system (e.g., [15, 24]). However, none of these approaches can protect a DHT against the join-leave attacks considered in this paper because just assigning a random or pseudo-random point to each new node (by using some random number generator or cryptographic hash function) does not suffice to preserve the balancing and majority conditions [3]. People in the peer-to-peer community are aware of the danger of these attacks [6, 8] and various solutions have been proposed that may help thwart these attacks in practice [4, 5, 28, 20, 27, 29] but until recently no mechanism was known that can *provably* cope with these attacks without sacrificing the openness of the system.

One such mechanism, that can only cope with a linear number of adversarial join requests, was proposed in [11]. The first mechanism that was shown to preserve randomness in the system under adaptive adversarial behavior for a polynomial number of adversarial join-leave requests uses *random* node IDs and enforces a *limited* lifetime on every node in the system, i.e., every node *has* to reinject itself after a certain amount of time steps [3]. However, this leaves the system in a hyperactive mode that may unnecessarily consume resources that could be better used for other purposes. Ideally, one would like to use *competitive* strategies. That is, the resources consumed by the mixing mechanism should scale with the join-leave activity of the system. Recently, it was shown that for a pebble-shuffling game this can be achieved [26]. In this game, there are n blue pebbles and ϵn red pebbles for some fixed constant $\epsilon < 1$. The pebbles are laid out on a ring and the red pebbles can join and leave the ring in an adaptive adversarial fashion. It was shown in [26] that a simple protocol called k-rotation exists that can preserve the majority condition with high probability. That is, for any sequence of $\Theta(\log n)$ pebbles along the ring, a majority of them is blue. However, the result in [26] cannot be taken over to a virtual space setting as adversarial strategies exist for which the k-rotation rule cannot satisfy the balancing condition. Therefore, we had to design a new strategy, which we called the cuckoo rule.

Also attacks on the data management layer have been considered in the past. Most of the work considers the flash crowd scenario in which many peers want to access the same information at the same time. When using a pure DHT design, this can lead to severe bottlenecks. To remove these bottlenecks, various caching strategies have been proposed. Among them are CoopNet [21], Backslash [30], PROOFS [31] in the systems community and [19] in the theory community. However, being able to handle flash crowds is not sufficient to survive the attacks considered in this paper because much worse than having many requests to the *same* data item is to have many requests to *different* data items residing at the *same* peer. Standard combining or caching strategies do not work here, so new strategies are needed. It turns out that, interestingly, work on deterministic simulations of CRCW PRAMs comes to the rescue here. This was pioneered by Mehlhorn and Vishkin [18] and further developed in a series of papers [1, 13, 16, 33]. The basic ideas behind our insert and lookup protocols are based on these results though adaptations of the proof techniques were necessary here because our strategies are based on a dynamic well-structured overlay network whereas the PRAM results above have only considered static complete networks or networks with expander-like properties.

1.3 Organization of the paper

In Section 2 we will prove Theorem 1.1 and in Section 3 we will prove Theorems 1.2 and 1.3. The paper ends with a conclusion.

2. ANALYSIS OF THE CUCKOO RULE

Recall that a *region* is an interval of size $1/2^r$ in $[0, 1)$ for some positive integer r that starts at an integer multiple of $1/2^r$. Let $\hat{R}$ be any fixed region of size $(c \log n) \cdot k/n$, for some constant c, for which we want to check the balancing and majority conditions over polynomial in n many steps. Thus, $\hat{R}$ contains exactly $c \log n$ many k-regions. The *age* of a k-region is the difference between the current round and the last round when a new node was placed into it (and all old nodes got evicted), and the age of $\hat{R}$ is defined as the sum of the ages of its k-regions. A node in $\hat{R}$ is called *new* if it was placed in $\hat{R}$ when it joined the system, and otherwise it is called *old*. Before we start with our analysis, we state some technical lemmas. The bounds in the first are also known as Chernoff bounds.

LEMMA 2.1 ([17]). *Suppose that $X_1, \ldots, X_n$ are independent binary random variables. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Then it holds for all $\epsilon \geq 0$ that $\Pr[X \geq (1+\epsilon)\mu] \leq e^{-\epsilon^2\mu/(2(1+\epsilon))}$ and for all $0 \leq \epsilon \leq 1$ that $\Pr[X \leq (1-\epsilon)\mu] \leq e^{-\epsilon^2\mu/2}$.*

LEMMA 2.2. *Suppose that $X_1, \ldots, X_n$ are independent random variables with $\Pr[X_i = t] = p(1-p)^{t-1}$ for every $t \in \mathbb{N}$ for some fixed $0 < p < 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Then it holds for every $\epsilon > 0$ that $\Pr[X \geq (1+\epsilon)\mu] \leq e^{-\epsilon^2 n/2(1+\epsilon)}$ and for all $0 \leq \epsilon \leq 1$ that $\Pr[X \leq (1-\epsilon)\mu] \leq e^{-\epsilon^2 n/2}$.*

PROOF. Consider transforming every $X_i = t$ into a binary string $B_t = (000...01)$ with $(t-1)$ zeroes. Then the event $X_1 = t_1$, $X_2 = t_2, X_3 = t_3, \ldots$ can be represented by a string B of the form $B_{t_1} \circ B_{t_2} \circ B_{t_3} \ldots = 00...1\,00...1\,00...1....$ Notice that B contains n many 1's and the total number of positions (0 or 1) in B is $T = \sum_i t_i$.

Now, consider instead an infinite set of binary random variables $Y_1, Y_2, Y_3, \ldots$ with $\Pr[Y_i = 1] = p$. Viewing Y_i as representing the ith position in B, it is not difficult to check that

$$\Pr\left[\sum_{i=1}^{n} X_i \geq T\right] = \Pr\left[\sum_{j=1}^{T} Y_j \leq n\right]$$

and

$$\Pr\left[\sum_{i=1}^{n} X_i \leq T\right] = \Pr\left[\sum_{j=1}^{T} Y_j \geq n\right]$$

Setting $T = (1 \pm \epsilon)\mu$ and applying the Chernoff bounds to $\sum_{j=1}^{T} Y_j$ gives the lemma. $\square$

THEOREM 2.3 ([10]). *Let $X_1, \ldots, X_n$ be an arbitrary set of random variables and let f be a function satisfying the property that for each $i \in \{1, \ldots, n\}$ there is a non-negative c_i such that $|\mathrm{E}[f \mid X_1, \ldots, X_i] - \mathrm{E}[f \mid X_1, \ldots, X_{i-1}]| \leq c_i$. Then*

$$\Pr[f \geq \mathrm{E}[f] + t] \leq e^{-t^2/(2\sum_{i=1}^{n} c_i^2)}$$

and

$$\Pr[f \leq \mathrm{E}[f] - t] \leq e^{-t^2/(2\sum_{i=1}^{n} c_i^2)} .$$

Note that the X_i's in this theorem are *not* required to be independent. Now we can start with the proof of Theorem 1.1. We assume that before the adversary starts with its rejoin operations, only the n blue nodes were in the system, and sufficiently many rejoin operations have been executed on the blue nodes so that every k-region has been entered by a new node at least once. Afterwards, the adversary enters with its ϵn red nodes one by one, using the cuckoo rule in each round, and then it starts executing rejoin operations on the red nodes as it likes. The assumption of acting on a sufficiently old system significantly simplifies the proofs.

The next lemma follows directly from the cuckoo rule because every k-region can have at most one new node at any time.

LEMMA 2.4. *At any time, $\hat{R}$ contains at most $c \log n$ new nodes.*

In order to bound the number of old nodes in $\hat{R}$, we first have to bound the age of $\hat{R}$ (Lemma 2.5). Then we bound the maximum number of nodes in a k-region (Lemma 2.6) and use this to bound the number of evicted blue and red nodes in a certain time interval (Lemma 2.9). After that, we can combine all lemmas to bound the number of old blue and red nodes in $\hat{R}$ (Lemma 2.10).

LEMMA 2.5. *At any time, $\hat{R}$ has an age within $(1 \pm \delta)(c \log n)$ n/k, with high probability, where $\delta > 0$ is a constant that can be made arbitrarily small depending on the constant c.*

PROOF. Let $R_1, \ldots, R_C$ be the k-regions of $\hat{R}$, where $C = c \log n$. For every k-region R_i, let the random variable X_i denote the age of R_i at the beginning of the given round, and let $X = \sum_{i=1}^{C} X_i$. For all i and $t \geq 1$ it holds that $\Pr[X_i = t] = (k/n)(1-(k/n))^{t-1}$. Hence, X_i is geometrically distributed with probability $p = k/n$. Thus, $\mathrm{E}[X_i] = 1/p = n/k$, and therefore, $\mathrm{E}[X] = \sum_{i=1}^{C} \mathrm{E}[X_i] = (n/k) \cdot C$. It remains to show that X is concentrated around $\mathrm{E}[X]$.

Unfortunately, the ages of the k-regions are not independent as two k-regions cannot have the same age. However, there is an easy solution to this problem. Let $Y_1, \ldots, Y_C$ be independent random variables with the same probability distributions as $X_1, \ldots, X_C$ and let $Y = \sum_{i=1}^{C} Y_i$. Then it holds for all ages $0 < t_1 < t_2 \ldots < t_C$ that

$$\begin{aligned}
&\Pr[X_1 = t_1 \ \wedge \ldots \wedge \ X_C = t_C] \\
&= \prod_{i=1}^{C} \Pr[X_i = t_i \mid X_1 = t_1 \ \wedge \ldots \wedge \ X_{i-1} = t_{i-1}] \\
&= \prod_{i=1}^{C} \left(1 - \frac{k}{n}\right)^{-(i-1)} \Pr[Y_i = t_i] \\
&= \left(1 - \frac{k}{n}\right)^{-\binom{C}{2}} \Pr[Y_1 = t_1 \ \wedge \ldots \wedge \ Y_C = t_C]
\end{aligned}$$

Hence, because all X_i's have the same probability distribution, it holds for any $T \geq 1$ that

$$\begin{aligned}
&\Pr[X \geq T] \\
&= \sum_{\{t_1,\ldots,t_C\} \subseteq \mathbb{N}, \sum_i t_i \geq T} \Pr[X_1 = t_1 \ \wedge \ldots \wedge \ X_C = t_C] \\
&= \sum_{\{t_1,\ldots,t_C\} \subseteq \mathbb{N}, \sum_i t_i \geq T} \Pr[Y_1 = t_1 \ \wedge \ldots \wedge \ Y_C = t_C] \left(1 - \frac{k}{n}\right)^{-\binom{C}{2}} \\
&\leq \left(1 - \frac{k}{n}\right)^{-\binom{C}{2}} \Pr[Y \geq T] \ \leq \ e^{C^2 k/n} \Pr[Y \geq T] .
\end{aligned}$$

Combining this with Lemma 2.2, it follows that $\Pr[X \geq (1+\delta)C \cdot n/k]$ is polynomially small for any constant $\delta > 0$ depending on the constant in C.

For a lower bound on the age of $\hat{R}$, we use the mapping $f : \mathbb{N}^C \to \mathbb{N}^C$ with

$$f(t_1, t_2, \ldots, t_C) = (t_1 + d_1, t_2 + d_2, \ldots, t_C + d_C)$$

where

$$d_j = |\{k \in \{1, \ldots, C\} \mid t_k < t_j \ \vee \ (t_k = t_j \ \wedge k < j)\}|$$

for all $j \in \{1, \ldots, C\}$. It is easy to check that this mapping is injective. Furthermore, for all $(t_1, \ldots, t_C) \in \mathbb{N}$ it holds for $(t'_1, \ldots, t'_C) = f(t_1, \ldots, t_C)$ that $t'_1, \ldots, t'_C$ are pairwise disjoint and

$$\Pr[Y_1 = t_1 \ \wedge \ldots \wedge \ Y_C = t_C] = \Pr[X_1 = t'_1 \ \wedge \ldots \wedge \ X_C = t'_C]$$

Hence, for any $T \geq 0$,

$$\begin{aligned}
&\Pr[Y \geq T] \\
&= \sum_{(t_1,\ldots,t_C)\in\mathbb{N}^C, \sum_i t_i \geq T} \Pr[(Y_1,\ldots,Y_C) = (t_1,\ldots,t_C)] \\
&= \sum_{(t_1,\ldots,t_C)\in\mathbb{N}^C, \sum_i t_i \geq T} \Pr[(X_1,\ldots,X_C) = f(t_1,\ldots,t_C)] \\
&\leq \sum_{\{t'_1,\ldots,t'_C\}\subseteq\mathbb{N}, \sum_i t'_i \geq T} \Pr[(X_1,\ldots,X_C) = (t'_1,\ldots,t'_C)] \\
&= \Pr[X \geq T]
\end{aligned}$$

and therefore $\Pr[X \leq T] \leq \Pr[Y \leq T]$. Combining this with Lemma 2.2, it follows that $\Pr[X \leq (1-\delta)C\cdot n/k]$ is polynomially small for any constant $\delta > 0$ depending on the constant in C. □

LEMMA 2.6. *For any k-region R in $\hat{R}$ it holds at any time that R has at most $O(k \log n)$ nodes, with high probability.*

PROOF. We need two claims to prove the lemma. Let $T = \gamma(n/k)\ln n$, where γ is a sufficiently large constant.

CLAIM 2.7. *For any (red or blue) node v, v gets replaced at most $(1+\delta)\gamma \ln n$ times within T rounds, w.h.p., where $\delta > 0$ can be made arbitrarily small depending on γ.*

PROOF. For any $t \in \{1,\ldots,T\}$ let the binary random variable X_t be 1 if and only if v gets replaced in the tth round. Let $X = \sum_{t=1}^{T} X_t$. Because a k-region is chosen uniformly at random for eviction, $\Pr[X_t = 1] = k/n$ for every t. Hence, $\mathrm{E}[X] = (k/n)\cdot T = \gamma \ln n$. Since the X_t's are independent, the bound on X follows from the Chernoff bounds. □

CLAIM 2.8. *For any k-region R in $\hat{R}$ it holds at any time that R has an age of at most T, w.h.p.*

PROOF. The probability that R is evicted in some round is k/n, and this probability is independent of other rounds. Hence, the probability that R has an age of at least T is equal to $(1-\frac{k}{n})^T \leq e^{(k/n)\cdot T} = n^{-\gamma}$. □

The two claims imply that there are at most $(1+\epsilon)n\cdot(1+\delta)\gamma\ln n$ node replacements during the lifetime of a k-region, w.h.p. Hence, the expected number of nodes in a k-region can be at most

$$\frac{k}{n}\cdot((1+\epsilon)n\cdot(1+\delta)\gamma\ln n + 1) = (1+\epsilon)(1+\delta)k\cdot\gamma\ln n + 1\ .$$

Since the locations of the node replacements are independent of each other, it follows from Lemma 2.1 that the number of nodes in a k-region is at most $O(k\log n)$ at any time, w.h.p. □

Next we bound the number of blue and red nodes that are evicted in a certain time interval.

LEMMA 2.9. *For any time interval I of size $T = (\gamma/\epsilon)\log^3 n$, the number of blue nodes that are evicted in I is within $(1\pm\delta)T\cdot k$, with high probability, and the number of red nodes that are evicted in I is within $(1\pm\delta)T\cdot\epsilon k$, with high probability, where $\delta > 0$ can be made arbitrarily small depending on γ.*

PROOF. We start with the proof for the blue nodes. Consider any time interval I consisting of T rounds. For every $t \in \{1,\ldots,T\}$ let the random variable X_t denote the number of blue nodes evicted in the tth round of I, and let $X = \sum_{t=1}^{T} X_t$. Since there are n blue nodes in the system, it follows that $\mathrm{E}[X_t] = k$ for every t, no matter how the blue nodes are distributed, and therefore $\mathrm{E}[X] = T\cdot k$. From Lemma 2.6 we know that every X_t is at most $4\log n$, w.h.p. Also, for any given $X_1,\ldots,X_{t-1}$, $\mathrm{E}[X_t \mid X_1,\ldots,X_{t-1}] = \mathrm{E}[X_t] = k$. Hence, it holds for the function $f(X_1,\ldots,X_n) = \sum_{i=1}^{n} X_i$ that

$$\begin{aligned}
&|\mathrm{E}[f \mid X_1,\ldots,X_i] - \mathrm{E}[f \mid X_1,\ldots,X_{i-1}]| \\
&= \left|\left(\mathrm{E}\left[\sum_{j=i+1}^{n}(X_j \mid X_1,\ldots,X_i)\right] + \sum_{j=1}^{i} X_j\right) - \right. \\
&\quad \left.\left(\mathrm{E}\left[\sum_{j=i}^{n}(X_j \mid X_1,\ldots,X_{i-1})\right] + \sum_{j=1}^{i-1} X_j\right)\right| \\
&= |X_i - \mathrm{E}[X_i]| \leq 4\log n
\end{aligned}$$

Thus, the method of bounded martingale differences (Lemma 2.3) implies that, for any constant $\delta \geq 0$,

$$\Pr[X \geq (1+\delta)T\cdot k] \leq e^{-\delta^2(T\cdot k)^2/(2\sum_{i=1}^{T}(4\log n)^2)}$$

which is polynomially small in n if the constant in T is sufficiently large. The same holds for $\Pr[X \leq (1-\delta)T\cdot k]$, which proves the lemma for the blue nodes. The proof for the red nodes is the same. □

Combining Lemmas 2.5 to 2.9, we obtain the following lemma.

LEMMA 2.10. *At any time, $\hat{R}$ has within $(1\pm\delta)(c\log n)\cdot k$ old blue nodes and within $(1\pm\delta)(c\log n)\cdot\epsilon k$ old red nodes, with high probability, if none of these has rejoined.*

PROOF. Consider any age distribution $t_1,\ldots,t_C$ for the k-regions $R_1,\ldots,R_C$ of $\hat{R}$, where $C = c\log n$. Let $T = (\gamma/\epsilon)\log^3 n$ be selected as in Lemma 2.9. Then it follows from Lemma 2.9 and the Chernoff bounds that $\hat{R}$ hat at least

$$\frac{(1+\delta)T\cdot k}{n/k}\sum_{i=1}^{C}\lfloor t_i/T\rfloor \geq \frac{(1+\delta)k^2}{n}\left(\sum_{i=1}^{C} t_i - C\cdot T\right)$$

blue nodes and at most

$$\frac{(1-\delta)T\cdot k}{n/k}\sum_{i=1}^{C}\lceil t_i/T\rceil \leq \frac{(1-\delta)k^2}{n}\left(\sum_{i=1}^{C} t_i + C\cdot T\right)$$

blue nodes, w.h.p. Since $\sum_{i=1}^{C} t_i$ is within $(1\pm\delta')Cn/k$ according to Lemma 2.5, Lemma 2.10 follows for the blue nodes.

The same calculations (with an additional ϵ factor) apply to the red nodes. □

Combining Lemmas 2.4 and 2.10, we can now prove when the balancing and majority conditions are satisfied.

- **Balancing condition:** From Lemmas 2.4 and 2.10 it follows that every region R of size $(c\log n)k/n$ has at least $(1-\delta)(c\log n)\cdot k$ and at most $(1+\delta)(c\log n + (c\log n)k + (c\log n)\epsilon k) = (1+\delta)(c\log n)(1+(1+\epsilon)k)$ nodes, where the constant $\delta > 0$ can be made arbitrarily small. Hence, the regions are balanced within a factor of close to $(1+\epsilon+1/k)$.

- **Majority condition:** From Lemmas 2.4 and 2.10 it also follows that every region of size $(c\log n)k/n$ has at least $(1-\delta)(c\log n)\cdot k$ blue nodes and at most $(1+\delta)(c\log n + (c\log n)\cdot\epsilon k)$ red nodes, w.h.p., where the constant $\delta > 0$ can be made arbitrarily small. These bounds are also tight in the worst case, which happens if the adversary focuses on a specific region R of size $(c \log n)k/n$ and continuously rejoins with any red node outside of R. Hence, the adversary is not able to obtain the majority in any region of size $(c \log n)k/n$ as long as $(c \log n)(\epsilon k + 1) < (c \log n) \cdot k$ which is true if and only if $\epsilon < 1 - 1/k$.

Hence, for $\epsilon < 1 - 1/k$ the balancing and majority conditions are satisfied, w.h.p., and this is sharp, which proves Theorem 1.1.

3. INSERT AND LOOKUP PROTOCOLS

In this section we present our robust insert and lookup protocols. These protocols are based on a dynamic de Bruijn graph and $2c - 1$ one-way hash functions with certain expansion properties. Also other dynamic graphs may be used, but the dynamic de Bruijn graph turns out to be the most useful for our purposes. We first describe the dynamic de Bruijn graph and how to store data and route messages in it, and then we specify what kind of expansion properties the hash functions need to satisfy. Afterwards, we present and analyze the insert and lookup protocols.

For simplicity, we assume that the number of honest nodes in the system only deviates by a constant factor over time and that n is the maximum number of nodes in the system at any time. (We just need this so that we can focus on a fixed region size. Local-control update mechanisms for the region size such as the ones in [3, 11] may be used if n significantly changes over time.

3.1 The dynamic de Bruijn graph

In the classical d-dimensional de Bruijn graph, $\{0,1\}^d$ represents the set of nodes and two nodes $x, y \in \{0,1\}^d$ are connected by an edge if and only if there is a $b \in \{0,1\}$ so that $x = (x_1 \ldots x_d)$ and $y = (bx_1 \ldots x_{d-1})$ (i.e., y is the result of a right shift of the bits in x with the highest bit position taken by b) or $y = (x_2 \ldots x_d b)$. When viewing every node $x \in \{0,1\}^d$ as a point $\sum_{i=1}^d x_i/2^i \in [0,1)$ and letting $d \to \infty$, then the node set of the de Bruijn graph is equal to $[0,1)$ and two points $x, y \in [0,1)$ are connected by an edge if and only if $x = y/2$, $x = (1+y)/2$, $x = 2y \bmod 1$, or $x = (2y-1) \bmod 1$. This motivates the following dynamic variant of the de Bruijn graph (e.g., [19]):

Recall the definition of a region. We identify the peers by their points in $[0,1)$. Given a peer $v \in [0,1)$, we define its *quorum region* R_v as the unique region of size closest to $(\gamma \log n)/n$ from above that contains v, where $\gamma > 1$ is a sufficiently large but fixed constant. For any set of peers $V \subset [0,1)$, we require that every peer $v \in V$ maintains connections to all peers whose quorum regions contain a point in $\{v, v/2, (1+v)/2, 2v \bmod 1, (2v-1) \bmod 1\}$. Let us call the resulting graph $DB(V)$. The following lemma easily follows from the fact that the d-dimensional de Bruijn graph has a constant degree and a diameter of $O(d)$.

LEMMA 3.1. *For any node set $V \subset [0,1)$ that satisfies the balancing condition, every quorum region forms a clique of $\Theta(\log n)$ nodes, which implies that $DB(V)$ has a diameter of $O(\log n)$ and a degree of $O(\log n)$.*

Besides having nice topological properties, the dynamic de Bruijn graph is easy to update. Whenever a peer v enters of leaves the system, only the quorum regions containing a point in $\{v, v/2, (1+v)/2, 2v \bmod 1, (2v-1) \bmod 1\}$ are affected, which only sum up to $O(\log n)$ nodes when using the cuckoo rule, w.h.p.

3.2 Reliable storage and routing

If also the majority condition holds, then the honest nodes are in the majority in each quorum region. This allows the honest nodes to wash out adversarial behavior violating the protocols by simple majority decision. In order to reliably store data, we therefore demand that copy i of data item x be stored in all nodes of the unique quorum region containing $h_i(x)$.

In order to route a message from a node $v \in [0,1)$ to a point $w \in [0,1)$ (representing a node or location of a copy) in a reliable way, we execute the following Route(v, w) protocol:

Focus only on the first $\log n$ bits of the binary representation of v, denoted by $(v_1 v_2 \ldots v_{\log n})$, and forward the message from R_v along the quorum regions containing the points $(v_2 v_3 \ldots v_{\log n} w_1)$, $(v_3 v_4 \ldots v_{\log n}\, w_1 w_2)$, and so on, until the quorum region containing the point $(w_1 w_2 \ldots w_{\log n})$ is reached, which also contains w. It is easy to check that this routing strategy can be performed along adjacent regions in $DB(V)$ and that the following lemma holds.

LEMMA 3.2. *If the balancing and majority conditions are satisfied, then Route(v, w) combined with majority decision reliably routes a message in at most $\log n$ communication rounds from the quorum region containing v to the quorum region containing w.*

The only problem is the congestion caused by routing multiple messages. As a prerequisite for this we need suitable hash functions.

3.3 Robust hash functions

Next we specify two properties the $2c-1$ hash functions $h_1, \ldots, h_{2c-1}$ have to satisfy for our protocols to work. The first property, given in Lemma 3.3, will be crucial to show that many requests can avoid congested nodes during the routing, and the second property, given in Lemma 3.4, will be crucial to show that many requests can avoid destination nodes with a high contention, no matter which collection of data items is selected for the requests.

In order to investigate the congestion caused by the routing in the dynamic de Bruijn graph, we introduce the shuffle graph. For any $d \geq 1$, the d-dimensional shuffle graph $SH(d)$ consists of $d+1$ levels numbered from 0 to d. The node set of level i is given as $V_i = \{0,1\}^d$, and for every $0 \leq i < d$, every pair of nodes $v \in V_i$ and $w \in V_{i+1}$ is connected if and only if their binary representations satisfy $v = (v_1 v_2 \ldots v_d)$ and $w = (v_2 v_3 \ldots v_d b)$ for some bit $b \in \{0,1\}$.

The shuffle graph is related to the well-known Omega network. It is a leveled form of the de Bruijn graph and contains for every source $s \in V_0$ and destination $t \in V_d$ a unique path of length d from s down to t. In fact, these paths represent the paths the packets will move along through $DB(V)$ when using the reliable routing strategy above. So instead of focusing on routing problems between nodes in $DB(V)$ we will focus on routing problems from source nodes in V_0 to destination nodes in V_d in $SH(d)$. This makes it easier to investigate congestion issues in $DB(V)$. Notice that for every $1 \leq \ell \leq d$, $SH(d)$ contains $2^{d-\ell}$ disjoint graphs $SH(\ell)$ from level 0 to ℓ. Let $\mathcal{SH}(\ell)$ denote their set.

Now, let U be the universe of data items, $C = U \times \{1, \ldots, 2c-1\}$ be the set of copies, $V = \bigcup_{i=1}^d V_i$ be the set of nodes in $SH(d)$ and $\mathcal{H} = \{h_1, \ldots, h_{2c-1}\}$ be the set of hash functions used to assign the $2c-1$ copies of each data item to points in $[0,1)$. The mapping of copies to points in Section 3.2 implies that, given $\mathcal{H}$, the destination node of the ith copy of data item $u \in U$ in $SH(d)$ is the node $v \in V_d$ representing the highest d bits in the binary representation of $h_i(u)$. To simplify our presentation, when we talk in the following about the node $h_i(u)$, we mean the corresponding node $v \in V_d$.

In order to witness a bad congestion, we will make use of so-called k-bundles $F \subseteq C \times V$, where every edge $(c, v) \in F$ represents an event that a request for copy c passes through a congested node $v \in V$. Let $U(F) = \{u \in U \mid \exists v \in V : ((u,i),v) \in F$ for some $i\}$ and $V(F) = \{v \in V \mid \exists c \in C : (c,v) \in F\}$. Given a set $\mathcal{H}$, we call F a *k-bundle* if

1. $|F| = k|U(F)|$ and
2. there is an injective mapping $f : U(F) \to V_0$ of data items to source nodes so that for every edge $e = ((u,i),v) \in F$ the path from $f(u) \in V_0$ to $h_i(u) \in V_d$ in $SH(d)$ passes through v.

F is also called a k-bundle of $U(F)$, and we define $\Gamma_F(U(F)) = V(F)$. A k-bundle is called *σ-sparse* if for any subset $V'_\ell \subseteq V_\ell$ representing all nodes in level ℓ of some graph $SH(\ell)$ in $\mathcal{SH}(\ell)$ it holds that $|V(F) \cap V'_\ell| \le \sigma |V'_\ell|$. $\mathcal{H}$ is called a *(σ,k)-expander* if for any $S \subseteq U$ with $|S| \le n$ and any σ-sparse k-bundle F of S, $|\Gamma_F(S)| \ge |S|$. In the following, let $|U| = m$.

LEMMA 3.3. *If $c \ge 6\log m$ and $m \ge n^3$, $\sigma \le 1/(8e\log n)$ and the functions $h_1,\ldots,h_{2c-1}$ are chosen uniformly and independently at random, then $\mathcal{H}$ is a $(\sigma, c/2)$-expander with high probability.*

PROOF. Let $d = \log n$. Suppose that, for randomly chosen functions $h_1,\ldots,h_{2c-1}$, $\mathcal{H}$ is not a $(\sigma,c/2)$-expander. Then there exists a set $S \subset U$ with $|S| \le n$ and a σ-sparse $c/2$-bundle F of S with $|\Gamma_F(S)| < |S|$. We claim that the probability p_s that such a set S of size s exists is at most

$$\binom{m}{s}\binom{n}{s}\binom{(2c-1)s}{cs/2} d^{cs/2} \binom{dn}{s} \cdot \sigma^{cs/2}$$

This holds because there are $\binom{m}{s}$ ways of choosing a subset $S \subset U$, $\binom{n}{s}$ ways of assigning the nodes in S to nodes in V_0, $\binom{(2c-1)s}{cs/2}$ ways of selecting $cs/2$ edges for F and $d^{cs/2}$ ways of specifying a level for them. Furthermore, there are at most $\binom{dn}{s}$ ways of choosing a σ-sparse $W \subseteq V$ witnessing a bad expansion of F. Since W is σ-sparse, the probability that any particular edge selected for F indeed points to a node in W is at most σ, and since the hash functions are chosen uniformly and independently at random, we obtain a total probability of at most $\sigma^{cs/2}$.

Next we simplify p_s. Using the conditions on c and σ in the lemma it holds that

$$\begin{aligned}
&\binom{m}{s}\binom{n}{s}\binom{(2c-1)s}{cs/2} d^{cs/2} \binom{dn}{s} \cdot \sigma^{cs/2} \\
\le\ & \left(\frac{em}{s}\right)^s \left(\frac{en}{s}\right)^s \left(\frac{e \cdot 2cs}{cs/2}\right)^{cs/2} d^{cs/2} \left(\frac{edn}{s}\right)^s \cdot \sigma^{cs/2} \\
=\ & \left(\frac{em}{s} \cdot \frac{en}{s} \cdot \frac{edn}{s} \cdot (4ed\sigma)^{c/2}\right)^s \\
\le\ & \left(\frac{m^2}{s^3} \cdot \left(\frac{1}{2}\right)^{3\log m}\right)^s \le \frac{1}{m^s}
\end{aligned}$$

Hence, summing up over all possible values of s, we obtain a probability of having a bad σ-sparse $c/2$-bundle of at most $2/m$, which proves the lemma. □

Although it seems that we only proved an expansion property for a static graph, notice that as long as n is the maximum number of nodes in the system at any time, we just need to consider all $SH(d)$ with $d \le \log n$ to capture all possible routing strategies for the nodes in $DB(V)$ at any time. Since the probability bound in Lemma 3.3 is polynomially small, the hash functions can also satisfy all of these $SH(d)$ together with high probability.

Next we prove another expansion result that is needed for an even load balancing of the data. We call $\mathcal{H}$ a *(λ,k,σ)-expander* if for any $S \subseteq U$ with $|S| \le \sigma n/c$ and any k-bundle F of S with $V(F) \subseteq V_d$ it holds that $|\Gamma_F(S)| \ge \lambda k|S|$. We just focus on the nodes in V_d here since we are only interested in the contention at the destinations of the requests for the data items in $U(F)$.

LEMMA 3.4. *Let $0 < \lambda < 1$ be any constant. Then it holds that for any $c \ge 6\log m$ and $\sigma \le 1/(\lambda(4e)^{(1+\lambda)/(1-\lambda)})$ that if the functions $h_1,\ldots,h_{2c-1}$ are chosen uniformly and independently at random, then $\mathcal{H}$ is a $(\lambda, c/2, \sigma)$-expander with high probability.*

PROOF. The proof is similar to the proof of Theorem 1 in [13]. Suppose that, for randomly chosen functions $h_1,\ldots,h_{2c-1}$, $\mathcal{H}$ is not a $(\lambda,c/2,\sigma)$-expander. Then there exists a set $S \subset U$ with $|S| \le \sigma n/c$ and a $c/2$-bundle F of S with $V(F) \subseteq V_d$ and $|\Gamma_F(S)| < \lambda(c/2)|S|$. We claim that the probability p_s that such a set S of size s exists is at most

$$\binom{m}{s}\binom{(2c-1)s}{cs/2}\binom{n}{\lceil \lambda(c/2)s - 1\rceil} \cdot \left(\frac{\lceil \lambda(c/2)s - 1\rceil}{n}\right)^{cs/2}$$

This holds because there are $\binom{m}{s}$ ways of choosing a subset $S \subset U$. Furthermore, there are $\binom{(2c-1)s}{cs/2}$ ways of a choosing $cs/2$ edges for F and at most $\binom{n}{\lceil\lambda(c/2)s-1\rceil}$ ways of choosing a set $W \subseteq V_d$ witnessing a bad expansion of the edges in F. The fraction of collections $\mathcal{H}$ for which the selected edges indeed point to nodes in W is at most $(\frac{\lceil\lambda(c/2)s-1\rceil}{n})^{cs}$ because the hash functions $h_1,\ldots,h_{2c-1}$ are chosen independently and uniformly at random.

Next we simplify p_s. Using the conditions on c and σ in the lemma it holds that

$$\begin{aligned}
&\binom{m}{s}\binom{(2c-1)s}{cs/2}\binom{n}{\lceil \lambda(c/2)s - 1\rceil} \cdot \left(\frac{\lceil \lambda(c/2)s - 1\rceil}{n}\right)^{cs/2} \\
\le\ & \left(\frac{em}{s}\right)^s (4e)^{cs/2} \left(\frac{2en}{\lambda cs}\right)^{\lambda cs/2} \left(\frac{\lambda cs}{2n}\right)^{cs/2} \\
=\ & \left[\frac{em}{s} \cdot \left((2e)^{1+\lambda}\lambda^{1-\lambda} \cdot \left(\frac{cs}{n}\right)^{1-\lambda}\right)^{c/2}\right]^s \\
\le\ & \left[m \cdot \left((2e)^{1+\lambda} \cdot \lambda^{1-\lambda} \cdot \sigma^{1-\lambda}\right)^{c/2}\right]^s \\
\le\ & \left[m \cdot \left(\frac{1}{2}\right)^{c/2}\right]^s \le \frac{1}{m^s}
\end{aligned}$$

Hence, summing up over all possible values of s, we obtain a probability of having a bad $c/2$-bundle of at most $2/m$, which proves the lemma. □

We are now ready to describe and analyze our lookup and update protocols. For both protocols we assume that $\mathcal{H}$ is appropriately chosen, i.e., it satisfies the expansion properties in Lemmas 3.3 and 3.4.

3.4 The lookup protocol

Suppose that we have n adversarially chosen lookup requests, one per peer. A naive lookup protocol for the requests would be as follows:

Sent out $2c-1$ packets per request, one for each hash function. Forward the packets level by level. For every quorum region R and every level ℓ, combine all lookup packets to the same copy into a single lookup packet. Once the packets have reached the destinations, answers are sent back level by level. In each level, the answers are split so that for all packets that were previously merged in that level an answer is sent back.

Under arbitrary adversarial behavior, this strategy does not work well because the adversary can generate many packets for different copies to the same destination. Moreover, even if the packets have a low congestion at the destinations, according to the Borodin-Hopcroft lower bound it can still happen that their unique routing paths through the network create a high congestion at intermediate levels. The latter congestion problem could be handled with the help of Valiant's trick of sending the packets to random intermediate destinations, but randomness in an adversarial environment is a costly resource that should avoided if possible. Thus, we decided to focus on a deterministic lookup protocol.

Our lookup protocol allows several attempts for each lookup request and uses a simple threshold mechanism to control the congestion in each attempt. More precisely, we repeat the following procedure for sufficiently many rounds:

For each of the remaining lookup requests, $2c-1$ packets are sent out, one for each of its $2c-1$ destinations. The packets are routed level by level, and packets to the same copy are combined in each region. If the number packets left in a region is more than $\gamma c \log^2 n$ for some level ℓ, then all of them are discarded. Otherwise, all of the packets are forwarded to the next level. For all packets that have reached their destinations, answers are sent back by reversing the routing of the lookup packets, splitting the answers whenever their lookup packets were combined. A lookup request is successful in that round if at least c answers are received for it.

Each of these attempts will take at most $O(c \log^3 n)$ communication rounds (under the assumption that one packet can be forwarded by each region in a communication round), but the question is, how many attempts are needed until all of the lookup requests are successful, no matter which data items have been selected for them. The following theorem gives an answer to this, which implies Theorem 1.2.

THEOREM 3.5. *For any set of n lookup requests out of a set U of polynomial size with one request per node, the lookup protocol can serve all requests in a guaranteed number of $O(\log n)$ attempts and needs at most $O(c \log^4 n)$ communication rounds altogether.*

PROOF. Let us consider any fixed attempt and let s be the number of remaining lookup requests. For simplicity, let us view the routing of packets in the dynamic de Bruijn graph as routing them along the d-dimensional shuffle graph $SH(d)$ with $d = \log n$.

Let the threshold set by the lookup protocol be $2c/\sigma$ with $\sigma = 1/(8e \log n)$ so that Lemma 3.3 can be applied. A node in the shuffle graph is called *congested* if packets for more than $2c/\sigma$ different copies pass it, and let W be the set of all congested nodes. Note that W is σ-sparse because in every subgraph $SH(\ell)$ in $\mathcal{SH}(\ell)$ there can be a total of at most $(2c-1)2^\ell$ packets and therefore at most $(2c-1)2^\ell/(2c/\sigma) \leq \sigma 2^\ell$ nodes at level ℓ with at least $2c/\sigma$ packets. Furthermore, since the total number of packets is $(2c-1)s$, $|W| \leq d \cdot (2c-1)s/(2c/\sigma) = \sigma ds$.

Since $\mathcal{H}$ is a $(\sigma, c/2)$-expander, there can be at most $|W|$ lookup requests with at least $c/2$ packets passing through nodes in W because otherwise we would have a σ-sparse $c/2$-bundle F with $|\Gamma_F(U(F))| < |U(F)|$. Hence, at most $|W| = \sigma ds = s/8e$ of the lookup requests fail. Therefore, at most $O(\log n)$ attempts are necessary until all requests have been served.

Going back from $SH(d)$ to region routing in the de Bruijn graph, we have to multiply the congestion threshold by a factor of $O(\log n)$ as there are $O(\log n)$ nodes in each quorum region. □

3.5 The insert protocol

The insert protocol is similar to the lookup protocol, with the difference that we play a collision game at the destinations (first used in [7] in the context of PRAM simulations) to keep the number of copies each node has to store as small as possible. The protocol proceeds in rounds. In each round, every remaining insert request generates a packet for each of its $2c-1$ destinations. The packets are routed level by level, combining packets wherever possible. If a region has more than $\gamma c \log^2 n$ packets in a level, all of them are discarded, and otherwise all of them are forwarded. In the destination level, we use a slightly more restrictive bound. If a region in the destination level has more than $\gamma' c \log n$ packets, for a constant γ', then all of them discarded, and otherwise acknowledgements are sent out for all of them. The acknowledgements will be sent back level-wise and split appropriately in order to inform all relevant source nodes. An insert request is successful in that round if at least c acknowledgements are received for it.

This insert protocol has the following performance, which implies Theorem 1.3.

THEOREM 3.6. *For any set of n insert requests out of a set U of polynomial size with one request per node, the insert protocol can serve all requests in a guaranteed number of $O(\log n)$ attempts and needs at most $O(c \log^4 n)$ communication rounds altogether. Furthermore, every node has to store at most $O(c \log^2 n)$ copies.*

PROOF. Consider any fixed attempt and let s be the number of remaining insert requests. For simplicity, we again view the routing of packets in the dynamic de Bruijn graph as routing them along the d-dimensional shuffle graph $SH(d)$ with $d = \log n$.

Let the threshold for the intermediate levels be $2c/\sigma$ with $\sigma = 1/(8e \log n)$ and the threshold for the final level be $2\gamma c$. A node in an intermediate level is *congested* if packets for more than $2c/\sigma$ different copies pass it, and a node in the final level is *congested* if packets for more than $2\gamma c$ different copies reach it. Let W_1 be the set of all intermediate congested nodes and W_2 be the set of all final congested nodes. From the previous proof we know that W_1 is σ-sparse and $|W_1| \leq \sigma ds$, and it is easy to see that $|W_2| \leq s/\gamma$.

Let S be the set of failed lookup requests. Then either S and W_1 form a σ-sparse $c/2$-bundle or S and W_2 form a $c/2$-bundle. In the first case, we know from the previous proof that $|S| \leq s/8e$. In the second case, $|S| \leq |W_2|/(\lambda c)$ for some constant $0 < \lambda < 1$ because $\mathcal{H}$ is a $(\lambda, c/2, 1/\gamma)$-expander if γ is sufficiently large. In both cases, $|S| \leq s/8e$. Hence, at most $O(\log n)$ attempts are necessary until all requests have been served.

Since each node only starts to accept packets once the total number of different copies in it with packets is below $2\gamma c$, every node only has to store $O(c)$ copies at the end. Moving from nodes to regions, this means that every quorum region has to store at most $O(c \log n)$ copies. □

The total number of stored copies per node could be reduced to $O(c \log n)$ if in each attempt we could count the total number of copies for which packets are sent to a quorum region and reject all packets based on this total count and the threshold $\gamma' c \log n$. Doing this deterministically is possible but requires sophisticated techniques based on sorting. So we did not consider this approach in this paper.

4. ROBUST RANDOM ID GENERATION

In order to generate a random ID for a node, we use a verifiable secret sharing (VSS) scheme. In VSS, a dealer D tries to store a secret s in n nodes so that it can be reliably recovered. More precisely, a protocol on n nodes is called a (n, k)-VSS scheme if, for any adversary owning k nodes, the following requirements hold:

- **Privacy**: If D is honest, then the adversary's view during the sharing phase reveals no information about s.
- **Correctness:** If D is honest, then the reconstructed value is *always* equal to the secret s.
- **Commitment:** Even if D is dishonest, any successful execution of the sharing phase determines a unique value s^* which will be reconstructed at the reconstruction phase.

A protocol fulfilling all these properties is, for example, the $\lfloor \frac{n-1}{4} \rfloor$-VSS in [12]. For completeness, we present it here:

- Sharing phase:
    1. D chooses a random bivariate polynomial $F \in K[x, y]$ of degree (at most) k in each variable s.t. $F(0, 0) = s$. It sends to each player P_i the (univariate) polynomials $f_i(x) = F(x, i)$ and $g_i(y) = F(i, y)$.
    2. Player P_i sends to each player P_j the value $g_i(j)$.
    3. Player P_i broadcasts a list L_i of players P_j for whom it holds that $f_i(j) \neq g_j(i)$.
- Local computation (by each player):
    1. Add edge (i, j) to the consistency graph G on n nodes if P_i is not in L_j and P_j is not in L_i.
    2. Find a maximal matching in $\bar{G}$.
    3. Define a vertex set C to include all vertices not in the matching. (C is a clique in G.)
    4. Define ADD to be the set of vertices i s.t. $i \notin C$ and there exist $2k + 1$ nodes $j \in C$ such that $(i, j) \in G$.
    5. If $|C| + |ADD| \geq 3k + 1$ then accept the sharing; otherwise, disqualify the dealer.
- Reconstruction phase: Each $P_i \in C \cup ADD$ provides $f_i(0)$. Use error correction on $\{f_i(0)\}_{i \in C \cup ADD}$ to recover the polynomial $g_0(y) = F(0, y)$. Compute $g_0(0)$.

The problem with applying it to our setting is that it uses a broadcast operation to disseminate information and it assumes the set of nodes to be static. However, it can be adapted so that it can be used to generate a random ID in a dynamic, asynchronous environment without a broadcast channel. This works as follows (the number of adversarial nodes is assumed to be at most $k/6$):

Suppose that node u wants to generate a new ID and let G be the group of nodes u knows in its quorum region R_u (which includes all honest node but may not include all adversarial nodes). Then u asks all nodes $v \in G$ to execute the VSS scheme above on G for a secret s_v picked at random from $[0, 1)$ by v. If v knows more than $4k$ nodes in G, then v executes step 1 of the sharing phase on s_v and attaches G to its messages; otherwise, it aborts. Every node $w \in G$ that receives a message from v with the same G it received from u executes step 2 of the sharing phase. Every node $w' \in G$ that receives step 2-messages from at least $|G| - k/6$ nodes $w \in G$ for some v, computes $L^v_{w'}$ and sends $L^v_{w'}$ to u. Once u has received from at least $|G| - k/6$ nodes $w \in G$ L^v_w's for *all* nodes v in some set $S \subseteq G$ with $|S| = |G| - k/2$, u starts the local computation phase to determine the set P of all v with $|C_v| + |ADD_v| \geq 3k+1$. If $|P| \geq |G| - k$, then u sends to each $v \in G$ an ID reconstruction request together with P and $\{(C_v, ADD_v)\}_{v \in P}$. Each node $w \in G$ receiving an ID reconstruction request from u forwards this request to all other nodes in G. Each node $w \in G$ receiving the same ID reconstruction request from at least $|G| - k/6$ nodes in G waits until it has sent L^p_w to u for all $p \in P$ and afterwards initiates the reconstruction phase by sending $\{(f^p_w(0), L^p_w)\}_{p \in P}$ to all $v \in G$. Each node $w \in G$ receiving at least $|G| - k/3$ reconstruction messages for all $p \in P$ first checks if there is a $p \in P$ s.t. no change of L^p_i for $\leq k/6$ messages it received together with suitable L^p_is for the $\leq k/3$ missing messages would fulfill the conditions on C_p and ADD_p. If so, it aborts. Otherwise, it recovers $g^p_0(y)$ for every p, computes $x = \bigoplus_{p \in P} g_0(y)$, and sends x to u.

We show the correctness with two lemmata, using the assumption that all honest nodes in G know each other and at most $k/6$ nodes in G are adversarial.

LEMMA 4.1. *In any case in which there is an honest node v that computes some ID x in the ID generation scheme initiated by some (honest or adversarial) node u, x must be random and no honest node computes a value different from x.*

PROOF. First of all, an honest node v only participates in the ID generation stage if $|G| > 4k$, i.e. G is sufficiently large for the VSS-protocol in [12] with threshold k to work. Second, notice that every honest node $v \in G$ will only reveal any of its private information about keys x_w if it received at least $|G| - k/6$ votes concerning the ID reconstruction message from u matching the reconstruction message it got from u. Thus, P and $(C_p, ADD_p)_{p \in P}$ are fixed for v at that stage. Furthermore, due to at most $k/6$ adversarial nodes, no honest node can have a different view of P and $(C_p, ADD_p)_{p \in P}$ when revealing its private information. Hence, at that point where the first honest node reveals private information about some keys, no adversarial node can influence P or $(C_p, ADD_p)_{p \in P}$ any more for the honest nodes.

Also, at that point where the first honest node v reveals private information, it must have sent L^p_v's for all $p \in P$ to u. Thus, it must have received $g^p_i(j)$'s from at least $|G| - k/6$ nodes in G for all $p \in P$. This, in turn, means that for each $p \in P$ at least $|G| - k/3$ honest nodes in G must have received a pair $(f^p_i(x), g^p_i(y))$ from p before any private information is revealed by any honest node. Let us call these nodes *p-safe*.

Now, any honest node v that is convinced that the sharing for some $p \in P$ is successful must have used at least $|G| - 5k/6$ $f^p_i(0)$'s from p-safe honest nodes, because among the at least $|G| - k/3$ $f^p_i(0)$'s it receives from nodes in G, at most $k/6$ can come from adversarial nodes and at most $k/3$ can come from non-p-safe honest nodes. If v is convinced of the correct sharing for p, then because it may have changed L^p_i's from at most $k/6$ of the p-safe honest nodes to justify this. Hence, at least $|G| - k$ $f^p_i(0)$'s from p-safe nodes will be considered when revealing p's secret which, according to [12], will recover a unique, unbiased value x_p.

Because $|P| \geq |G| - k$ and $|G| > 4k$ in order for an honest node v to participate, at least one x_p must have been generated by an honest node. This x_p is random and unknown to the adversarial nodes until $(C_p, ADD_p)_{p \in P}$ is fixed. Hence, when v is convinced that the recovery phase succeeded, it computes a random ID x. □

LEMMA 4.2. *Any honest node u initiating the random ID generation scheme will get the same value x back from at least $|G| - k/3$ nodes in G.*

PROOF. If u is honest, then it will wait until it has at least $|G| - k/6$ nodes $w \in G$ that sent L^v_w's to u for all nodes v of some set $S \subseteq G$ with $|S| \geq |G| - k/2$. In this case, at least $|G| - k/3$ honest nodes w must have sent L^v_w's for all nodes in S, and therefore at least $|G| - k/3$ honest nodes will initiate the reconstruction phase. This makes sure that every honest node in G receives reconstruction messages from at least $|G| - k/3$ honest nodes in G, which allows them to recover the keys x_p. Notice that no honest node will abort, because if u makes sure that the conditions for ADD_p and C_p are fulfilled for every $p \in P$, then every honest node can find corrections for the at most $k/6$ L_i^p's it received from adversarial nodes and can come up with suitable L_i^p's for the at most $k/3$ missing nodes so that the conditions on ADD_p and C_p are met. If these conditions can be met, it follows from [12] that unique keys can be recovered from the received parts, and from the lemma above it follows that these keys must be unbiased, and at least one of them must be random. Hence, all honest nodes in G that participate in the recovery, which are at least $|G| - k/3$, will agree on the same, random value for x. $\square$

5. CONCLUSIONS

In this paper we showed that, on a high level, a scalable DHT can be designed that is provably robust against adaptive adversarial join-leave attacks as well as insert and lookup attacks. Certainly, low-level protocols still have to be designed for our operations that work well and correctly in an asynchronous environment. We believe that designing such protocols is possible though their design and formal correctness proofs may require a significant effort.

6. REFERENCES

[1] H. Alt, T. Hagerup, K. Mehlhorn, and F.P. Preparata. Deterministic simulation of idealized parallel computers on more realistic ones. *SIAM Journal on Computing*, 16:808–835, 1987.

[2] J. Aspnes and G. Shah. Skip graphs. In *Proc. of the 14th ACM Symp. on Discrete Algorithms (SODA)*, pages 384–393, 2003.

[3] B. Awerbuch and C. Scheideler. Group Spreading: A protocol for provably secure distributed name service. In *Proc. of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, 2004.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. of the 5th Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.

[5] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of the 2nd Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 1999.

[6] S. Crosby and D. Wallach. Denial of service via algorithmic complexity attacks. In *Usenix Security*, 2003.

[7] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, efficient shared memory simulations. In *Proc. of the 5 ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 110–119, 1993.

[8] J. R. Douceur. The sybil attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[9] P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[10] D. Dubhashi and A. Panconesi. Concentration of measure for the analysis of randomized algorithms. Unpublished manuscript, accessible via http://www.cs.unibo.it/∼pancones/papers.html, October 20 1998.

[11] A. Fiat, J. Saia, and M. Young. Making Chord robust to Byzantine attacks. In *Proc. of the European Symposium on Algorithms (ESA)*, 2005.

[12] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The round complexity of verifiable secret sharing and secure multicast. In *Proc. of the 33rd ACM Symp. on Theory of Computing (STOC)*, pages 580–589, 2001.

[13] K.T. Herley and G. Bilardi. Deterministic simulations of PRAMs on bounded degree networks. *SIAM Journal on Computing*, 23:276–292, 1994.

[14] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahi. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *29th ACM Symp. on Theory of Computing (STOC)*, pages 654–663, 1997.

[15] F. Kuhn, S. Schmid, and R. Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[16] F. Luccio, A. Pietracaprina, and G. Pucci. A new scheme for the deterministic simulation of PRAMs in VLSI. *Algorithmica*, 5:529–544, 1990.

[17] McDiarmid. Concentration. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, pages 195–247. Springer Verlag, Berlin, 1998.

[18] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel mamories. *Acta Informatica*, 21:339–374, 1984.

[19] M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2003.

[20] S. Nielson, S. Crosby, and D. Wallach. Kill the messenger: A taxonomy of rational attacks. In *Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[21] V.N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[22] G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM '01*, 2001.

[24] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *USENIX Annual Technical Conference*, 2004.

[25] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[26] C. Scheideler. How to spread adversarial nodes? Rotate! In *Proc. of the 37th ACM Symp. on Theory of Computing (STOC)*, pages 704–713, 2005.

[27] A. Singh, M. Castro, A. Rowstron, and P. Druschel. Defending against Eclipse attacks on overlay networks. In *Proc. of the 11th ACM SIGOPS European Workshop*, 2004.

[28] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[29] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *Proc. of the 20th IEEE Computer Security Applications Conference (ACSAC)*, 2004.

[30] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[31] A. Stavron, D. Rubenstein, and S. Sahn. A lightweight robust P2P system to handle flash crowds. In *Proc. of the IEEE Intl. Conf. on Network Protocols (ICNP)*, 2002.

[32] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of the ACM SIGCOMM '01*, 2001. See also `http://www.pdos.lcs.mit.edu/chord/`.

[33] E. Upfal and A. Wigderson. How to share memory in a distributed system. *Journal of the ACM*, 34:116–127, 1987.

[34] B.Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, UCB/CSD-01-1141, University of California at Berkeley, 2001. See also `http://www.cs.berkeley.edu/∼ravenben/tapestry`.