Replica Placement for Route Diversity in Tree-Based Routing Distributed Hash Tables

Cyrus Harvesf and Douglas M. Blough, *Senior Member*, *IEEE*

Abstract—Distributed hash tables (DHTs) share storage and routing responsibility among all nodes in a peer-to-peer network. These networks have bounded path length unlike unstructured networks. Unfortunately, nodes can deny access to keys or misroute lookups. We address both of these problems through replica placement. We characterize tree-based routing DHTs and define MAXDISJOINT, a replica placement that creates route diversity for these DHTs. We prove that this placement creates disjoint routes and find the replication degree necessary to produce a desired number of disjoint routes. Using simulations of Pastry (a tree-based routing DHT), we evaluate the impact of MAXDISJOINT on routing robustness compared to other placements when nodes are compromised at random or in a contiguous run. Furthermore, we consider another route diversity mechanism that we call neighbor set routing and show that, when used with our replica placement, it can successfully route messages to a correct replica even with a quarter of the nodes in the system compromised at random. Finally, we demonstrate a family of replica query strategies that can trade off response time and system load. We present a hybrid query strategy that keeps response time low without producing too high a load.

Index Terms—Distributed systems, peer-to-peer networks, distributed hash tables, routing, replica placement, robustness.

1 INTRODUCTION

PEER-TO-PEER (p2p) networks are a popular substrate for building distributed applications because of their efficiency, scalability, resilience to failure, and ability to self-organize. The p2p architecture relies on the distribution of responsibility among hundreds of thousands, if not millions, of nodes in the network. Therefore, if a small set of nodes fail to serve data objects, properly maintain routing information, or route messages, the integrity of a very large-scale system may be compromised.

The efficiency of lookups has become a central focus of p2p design because many popular applications, like name resolution, publish-subscribe, and IP communication, rely on a lookup service as a core functionality. A p2p distributed hash table (DHT) may be used to provide this functionality. DHTs [25], [26], [33], [34] structure the network topology in a way that enables routing algorithms to produce lookup paths of bounded length (typically $O(\log N)$).

Unfortunately, when deployed over the Internet, DHTs may be impacted by the failure or compromise of peers in the overlay and performance guarantees no longer hold. In fact, it may not be possible to fetch a desired object at all. Many p2p networks allow nodes to join without prejudice, leaving the network vulnerable to attack. Furthermore, the network could face coordinated attacks from competitors or other groups that have an interest in the failure of the network. This type of coordinated attack behavior has been reported, for example, in p2p file sharing systems. DHTs are inherently less resilient to these attacks than unstructured networks because unstructured networks typically broadcast messages, which is a more robust (and much more expensive) mechanism.

Sit and Morris [31] classify attacks on DHTs into three categories:

1. storage and retrieval attacks, which target the manner in which peers manage data items;
2. routing attacks, which target the manner in which peers route messages; and
3. miscellaneous attacks, which target other aspects of the system, such as admission control or the underlying network routing service.

The first class of attack is commonly addressed with replication. Objects are replicated at several peers in the network to increase the likelihood that there will be a correct replica available. The benefits of replication on load balancing and overall performance have also been studied. To our knowledge, ours is the first work that considers how the placement of replicas affects object reachability through the routing infrastructure.

Numerous works [2], [3], [32] have relied on route diversity to mitigate the effects of routing attacks. Srivatsa and Liu [32] introduced the notion of *independent lookup paths* to improve routing robustness. Two paths are said to be independent if they share no hops other than the source and destination peers. It is worth noting that route diversity has benefits in addition to improving routing robustness. For example, diverse routes can be used to improve load balance and fairness or to circumnavigate congested areas of the network.

Our work realizes the benefits of replication and route diversity in concert through replica placement. In this paper, we consider a class of DHTs that route messages using a scheme which we call *tree-based routing*. We show

- *C. Harvesf is with Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052. E-mail: cyrush@microsoft.com.*
- *D.M. Blough is with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, KACB, Room 3356, Atlanta, GA 30332-0765. E-mail: doug.blough@ece.gatech.edu.*

Manuscript received 14 Oct. 2008; revised 24 June 2009; accepted 22 Sept. 2009; published online 4 Dec. 2009.
Recommended for acceptance by A. Schiper.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2008-10-0158.
Digital Object Identifier no. 10.1109/TDSC.2009.49.

that there exists a replica placement, which we call MAXDISJOINT, that creates *disjoint routes* in DHTs of this type. We prescribe the number and placement of replicas necessary to produce d disjoint routes from any source node to the replica set. With this scheme, we are able to tolerate $d-1$ malicious peers, whether they are attacking the storage and retrieval of data items or the routing infrastructure. Our approach is targeted specifically at DHT-based (structured) p2p systems with a multilevel routing structure. So-called "one-hop" DHTs [14], [15] are discussed in Section 6.

In order for disjoint routes to improve the robustness of the system, the client must be able to verify the integrity of data items. This is necessary for the client to detect when a malicious peer has tampered with the result of a data item lookup. Therefore, we assume that data in the system are self-certifying. This assumption is quite common in peer-to-peer systems [3], [7], [27] and is discussed in more detail in Section 6.

Using Pastry as an example, we evaluate MAXDISJOINT through simulation and show that a DHT with typical configuration parameters can benefit from our replica placement. Our experiments show that with only eight replicas and a quarter of nodes compromised at random, a node can find a route, which consists of only uncompromised nodes, to a correct replica with greater than 97 percent probability. MAXDISJOINT also tolerates runs of compromised nodes; with 16 replicas and 85 percent of the DHT compromised in a run, lookups can be resolved with greater than 96 percent probability. Furthermore, we use a technique which we call *neighbor set routing* to increase route diversity and improve the probability of lookup success. For example, a lookup performed with neighbor set routing and MAXDISJOINT placement can be resolved successfully with greater than 97 percent probability with 40 percent of nodes compromised at random. Finally, we demonstrate that the strategy used to query replicas can have a significant impact on performance and we propose a hybrid query strategy that can be used to trade off response time and system load for the best performance.

2 Related Work

To place our work in context, we discuss related work on replica placement, peer-to-peer routing security, and general peer-to-peer security issues.

2.1 Replica Placement

Replica placement has long been studied in the realm of distributed computing. Many studies have compared the performance of different placement schemes in terms of quality of service, availability, and time to recovery in different types of serverless systems [6], [9], [19], [22]. The first DHT-based replication schemes were only concerned with availability and thus local replication, i.e., replicas placed close to the master copy in the ID space, was used [26], [33]. As detailed herein, such placements have very little routing robustness.

A very important paper, which proposed the first deterministic nonlocal replica placement scheme for DHT-based systems, was that of Ghodsi et al. [12]. This paper discussed a set of *symmetric* replica placement schemes that, for a replication degree of d, divide the ID space into equivalence classes, each of size d. If an object with its ID in a particular equivalence class is replicated, replicas are placed at all IDs in the class. This is a very general definition, which is completely independent of routing. Thus, for a particular DHT structure, some such schemes could produce a large number of disjoint routes while others might produce very few. To realize benefits of this approach for routing security, it is therefore necessary to instantiate particular schemes for different DHTs or classes of DHTs and evaluate their routing characteristics. This is exactly the problem considered in this paper. The instantiation of symmetric replication that was presented in [12] was equally spaced replication. The paper contained a thorough evaluation, which showed that the technique reduces the message overhead in node joins and leaves, provides better load balance, and improves fault tolerance. However, routing robustness was not considered. It is worth noting that the MAXDISJOINT placement is equivalent to equally spaced replication in Chord. In other DHT implementations, however, MAXDISJOINT provides added flexibility in terms of tuning routing robustness that equally spaced replication does not provide.

Our prior work is the first that considers the impact of replica placement on routing robustness in DHTs. Our initial work focused on the benefits of equally spaced placement in Chord [16] and has expanded into the MAXDISJOINT placement [17], a general solution that gives the benefits of equally spaced placement in Chord to all DHTs that employ a tree-based routing scheme.

2.2 Peer-to-Peer Routing Security

A number of works that look to improve routing security are centered around the notion of route diversity. For example, Artigas et al. propose Cyclone, an equivalence-based routing scheme deployed over an existing structured peer-to-peer overlay [2]. Independent lookup paths are created by routing across different equivalence classes. Since the paths are independent and do not differ in the destination, Cyclone does not naturally mitigate the effects of storage and retrieval attacks. Furthermore, each peer is required to maintain additional routing information, which incurs overhead. In contrast, our replica placement creates disjoint routes without requiring any additional routing state or modifying the underlying routing scheme.

Portmann et al. use route diversity to provide message confidentiality [24]. Messages are split in two, encrypted, and sent to the destination across diverse paths. Route diversity is created by routing messages through the routing table entries that minimize route overlap. The appropriate entries are chosen using empirical results that depend on the network size. They show that, in the best case, this method results in an average path overlap of 15-25 percent between a pair of routes. In other words, at best, 15 percent of the routes will be common to both paths. We show analytically that our placement creates multiple nonoverlapping routes.

Castro et al. combine secure node identifier assignment, secure routing table maintenance, and secure message forwarding to create a secure routing primitive [3]. Secure node identifier assignment ensures that an adversary cannot take arbitrary identifiers. It also ensures a uniform distribution of compromised nodes. Secure routing table maintenance ensures that the average fraction of compromised

routing table entries does not exceed the fraction of compromised nodes in the network.

Secure message forwarding guarantees that a message sent to a key is delivered to all of its replicas with high probability. This component most closely resembles our work. It relies on route diversity to deliver messages to the neighborhood of destination. Unlike our approach, this iterative redundant routing scheme requires modification to the routing infrastructure of the DHT. One such modification is the forwarding of messages through the neighbors of the source node, which we call *neighbor set routing*. We will show that neighbor set routing is useful in creating route diversity and can improve on the benefit of our replica placement.

Mickens and Noble develop a framework for diagnosing broken overlay routes, whether they result from IP-level link failures or malicious peers in the overlay [21]. Once the cause of the broken route is detected, the IP-level link is circumnavigated or the malicious peer is excluded from the system. Rather than excluding peers that may have been falsely diagnosed as malicious, we use route diversity to avoid faulty nodes.

It is worth noting that path disjointedness has been considered in other contexts as well. Castro et al. propose a p2p multicast system for high bandwidth content (e.g., streaming video) [5]. Content is split into "stripes" such that the quality of the content improves with the number of stripes downloaded simultaneously. The stripes are delivered to subscribers via multicast trees. To ensure that the failure of a single node does not compromise all stripes, the trees are constructed to be inner node disjoint. In other words, no node will be an inner node of more than one multicast tree. Therefore, if a single node fails, no more than one stripe is lost as a result. Inner node disjoint trees are constructed by selecting roots that vary in terms of the node prefixes. MAXDISJOINT creates disjoint paths in much the same way in Pastry.

2.3 General Issues in Peer-to-Peer Network Security

Consistent node identity is critical to key placement and routing. Most structured networks place a key at the node with identifier "closest" to the key identifier. If nodes do not maintain a single, consistent identity, key placement can be compromised. Furthermore, an adversary that can take multiple identities has the ability to partition the network [8], [30]. Secure admission control protocols are necessary to limit the number of identities an entity can obtain [28]. Other works have implemented node identities in a censorship-resistant, anonymous fashion [10], [29].

Nodes must also be able to store keys fairly in a manner that allows for verification. A number of works have aimed to create self-certifying data. CFS [7] uses the cryptographic hash of a data item as its key. PAST [27] relies on cryptographic signatures to verify data integrity. An alternative is to use replication and Byzantine-fault-tolerant algorithms [4] to maintain data consistency and correctness. Self-certifying data are discussed in more detail in Section 6.

3 Distributed Hash Tables with Tree-Based Routing

Distributed hash tables are often referenced by their geometry, i.e., ring (Chord [33]), torus (CAN [25]), tree (Plaxton [23]), or some hybrid (Pastry [26], Tapestry [34]). The geometry impacts neighbor and route selection, which can have an impact on flexibility, resilience, and proximity performance as studied in [13]. Although we use the term "tree-based routing," we are not referring to the geometry, but the routing algorithm. Tree-based routing algorithms have specific properties that we define herein.

3.1 Tree-Based Routing DHTs

Consider a DHT with an ordered id space I with size $N = |I|$ and a *branching factor* B such that $\log_B N$ is integral. The branching factor is used by each node to construct its routing table. The routing table of a node u has the following properties:

1. The node u partitions the entire id space into contiguous segments and selects one node from each id segment to include in its routing table. The partitioning is performed as follows:
   a. The node u partitions the id space into B equal size contiguous parts.
   b. Of the B id segments, u selects the id segment I' of which it is a member.
   c. Steps 1a and 1b are repeated to repartition I' until parts of size one are created. The part that consists of the node u is discarded (there is no need for u to maintain a routing table entry to itself).
   d. At the end of the partitioning process, u will have created $(B-1)\log_B N$ contiguous parts of the id space, with sizes $\frac{N}{B}, \frac{N}{B^2}, \ldots, \frac{N}{B^{(\log_B N)-1}}$, and 1, and with $B-1$ parts of each size.
2. For each part P, u selects a node $v \in P$ that *covers* P and places it in its routing table. A node is said to cover a part P if its routing table contains $k > 1$ entries that cover the nonempty parts $P_1, P_2, \ldots, P_k$ and $P = P_1 \cup P_2 \cup \cdots \cup P_k$. By definition, a node u is said to cover the part consisting of its id. This definition, combined with partitioning of P into nonempty parts, ensures that the recursive definition of coverage terminates.

The partitioning and routing table construction is shown graphically in Fig. 1.

Note that tree-based routing DHT implementations differ in the manner in which Property 2 is satisfied. For example, in Chord [33], since routing is performed in the clockwise direction, the most counterclockwise node in each part must be selected because it is the only node that covers the entire id segment. In prefix-matching routing DHTs, all of the nodes within a part share a common prefix and cover the entire id segment; therefore, any node within the part may be chosen as a routing table entry. This explains the flexibility in choosing routing table entries in DHTs like Pastry [26] and Tapestry [34].

Routing is performed by forwarding the message destined for the id d to the entry that covers the id segment that contains d. We define a DHT constructed in this manner to be a *tree-based routing* DHT. If the paths from any source node to all possible destinations are aggregated, the resulting topology is a tree, which is how tree-based routing gets its name. Note that many popular DHT implementations [20], [23], [26], [33], [34] exhibit these properties and,

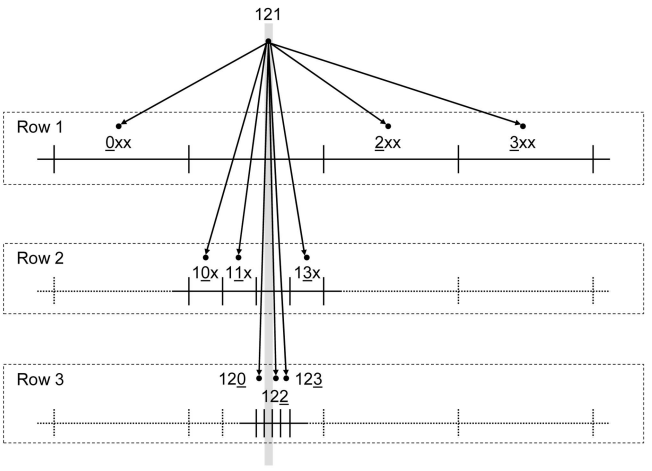

Fig. 1. Pastry routing table structure for the hypothetical node 121 ($N = 64$, $B = 4$). The space is partitioned into $(B-1)\log_B N = 9$ segments. The highlighted region marks the segment to which node 121 belongs.

therefore, employ a tree-based routing scheme. These DHTs have a number of useful properties, which we prove below.

First, tree-based routing DHTs are deterministic; that is, given a message destined for a node d, each node has one and only one routing table entry through which the message can be forwarded to d.

Lemma 1 (Determinism). *For the routing table of any node in a tree-based routing DHT, if the entries e_1 and e_2 cover id segments I_1 and I_2, respectively, then $I_1 \cap I_2 = \emptyset$.*

Proof. This follows naturally from the partitioning of the id space. □

Routes in tree-based routing DHTs are guaranteed to converge. This property holds when the DHT is *full*[1]; that is, every possible id is represented by a node.

Lemma 2 (Routing Convergence). *Consider the route in a full tree-based routing DHT from a source node s to a destination d represented as a series of nodes $n_1 = s, n_2, \ldots, n_{k-1}, n_k = d$ such that n_i is some entry e_j from the routing table of node n_{i-1} for $i > 1$. Suppose that $n_1, n_2, n_3, \ldots, n_k$ cover the id segments $I_1 = I, I_2, I_3, \ldots, I_k$.[2] Then,*

$$I_k = \{d\} \subset I_{k-1} \subset I_{k-2} \subset \cdots \subset I_2 \subset I_1.$$

Proof. Consider the hop from n_j to n_{j+1}. Since the node n_j covers the id segment I_j, it must partition I_j into B equal sized id segments. Since n_{j+1} covers I_{j+1}, which is one of the parts of I_j, then $I_{j+1} \subset I_j$. Furthermore, since the DHT is full, the node n_{k-1} has a part that contains only the destination d. □

Furthermore, it is possible to bound the number of hops in every route; we state this formally below.

Lemma 3 (Bounded Path Length). *Any path in a full tree-based routing DHT has at most $\log_B N$ hops.*

Proof. Since the id segment covered by each hop must be a subset of the id segment covered by the previous hop, the minimum ratio between the size of id segments covered by consecutive hops is the branching factor B. Therefore, the longest path repeatedly divides N by B with each hop until it reaches the destination. This requires $\log_B N$ hops. □

3.2 Creating Disjoint Routes with Tree-Based Routing

Determinism and routing convergence provide a natural avenue for creating disjoint routes. Routing convergence guarantees that once a path enters a segment of the id space, it will never proceed to a node that is outside of that segment. If two paths can be created originating in different segments, then the paths are guaranteed to be disjoint. Furthermore, the determinism property ensures that any two routing table entries will route to different segments. Therefore, we can create disjoint routes simply by routing through different routing table entries. This is stated formally in the following lemma.

Lemma 4. *In a full tree-based routing DHT, routes originating at a common source node with different first hops are disjoint.*

Proof. Suppose two routes originating at a common source node n have first hops e_1 and e_2, such that e_1 and e_2 are different routing table entries of n. Using the determinism property, if e_1 and e_2 cover id segments I_1 and I_2, respectively, then $I_1 \cap I_2 = \emptyset$.

Consider any hops h_1 and h_2 in the routes beginning with first hops e_1 and e_2, respectively. Suppose hops h_1 and h_2 cover the id segments I_1' and I_2', respectively. Using the routing convergence property, $I_1' \subset I_1$ and $I_2' \subset I_2$. Since $I_1 \cap I_2 = \emptyset$, $I_1' \cap I_2' = \emptyset$ and, therefore, the routes are disjoint. □

The source node can use any of its routing table entries as the first hop in a route; therefore, we can state the number of disjoint routes that can be created from any source node by counting routing table entries.

Lemma 5. *In a full distributed hash table that employs tree-based routing, there are at most d disjoint routes from any source node, where $d = (B-1)\log_B N$.*

Proof. Employing Lemma 4, we can create a disjoint route for each of the d routing table entries by routing to a destination in the segment covered by each entry. We cannot create $d+1$ or more disjoint routes because two or more routes would share the first hop and overlap. □

The proof for Lemma 5 alludes to choosing multiple destinations to create disjoint routes. This lends itself naturally to a replica placement. In the following section, we propose a replica placement that creates disjoint routes, which we call MAXDISJOINT.

4 The MaxDisjoint Replica Placement

Using Pastry as an example in the following sections, we will demonstrate how the properties of tree-based routing can be used to construct a replica placement that creates

1. Some tree-based routing DHT implementations have to provide an additional mechanism to ensure routing convergence when the DHT is not full. Pastry, for example, maintains the neighborhood and leaf sets for this purpose.

2. Any node can cover the entire id space; therefore, we can state that n_1 covers I.

disjoint routes. We begin with an example placement to provide the reader with some intuition and then move toward a more formal definition. After defining the placement, which we call MAXDISJOINT, we will evaluate the necessary replication degree to create a desired number of disjoint routes. Then, we will introduce the notion of a *run* and provide an expression for the maximum tolerable run length for a given replication degree. Next, we will discuss why MAXDISJOINT is a more adaptive and flexible solution than equally spaced replication. Finally, we outline the basic elements of an implementation of the MAXDISJOINT placement.

4.1 Intuition behind MaxDisjoint Placement

To create route diversity in Pastry via replica placement, it is necessary to place replicas such that a given replica set will use a diverse set of routing table entries for every possible source node. We use an example to provide the necessary intuition. Consider a Pastry ring with id space of size 64 and prefix matching in base-4 digits. We show the Pastry routing table structure graphically for a hypothetical node 121 in Fig. 1.

Suppose we wish to replicate an object with id 101 in this Pastry ring. Node 121 routes to this object through the routing table entry marked "10x" in Fig. 1. Suppose we replicate the object with the id 111 to target the routing table entry "11x" in the example. This approach creates an additional disjoint route for any lookups for object 101 originating at node 121. One route is forwarded via the entry "10x" and the other is forwarded via "11x." However, consider another source node 221. This node routes to the object 101 and 111 through the same entry marked "1xx" and, therefore, does not gain an additional disjoint route.

To move toward a more effective approach, consider all the replicas of object 101 that would create an additional disjoint route for node 121. These are: 001, 111, 120, 122, 123, 131, 201, and 301. Note that there are total of nine possible disjoint routes[3] (including the route to the object 101), which is the number of routing table entries for node 121. Of these replicas, there are only three that can create an additional disjoint route for every possible source node: 001, 201, and 301. These replicas create disjoint routes by targeting entries in the first row of the routing table. Note that targeting an entry in the first row of the routing table requires a single replica whose id differs from that of the master object in the first digit. To target entries deeper in the routing table, a larger number of replicas are required.

Suppose we wish to create five disjoint routes for all possible source nodes. Four routes can be created for every possible source node using the three replicas we have already discussed (001, 201, and 301) in addition to the object 101. To create the fifth route, we must target an entry deeper in the routing table. In the case of node 121, we may choose the replica 111. As alluded to before, this replica only creates a disjoint route for those source nodes whose ids start with the prefix "1" because these are the only nodes with an entry for "11x." Since there are four possible values for the prefix ($B = 4$), four replicas are required to target this routing table entry: 011, 111, 211, and 311. One of these four replicas will create an additional route for every possible source node depending on its prefix. The remaining three will be routed through previously used routing table entries overlapping a previous route. This is shown graphically in Fig. 2. Five disjoint routes are created for node 121, one each for the replicas R001 (or R011), R101, R111, R201 (or R211), and R301 (or R311). In a similar fashion, we can create a sixth disjoint route using the replicas 021, 121, 221, and 321; and a seventh using 031, 131, 231, and 331.

3. There is actually a tenth "zero-hop" path that can be created by placing a replica at 121.

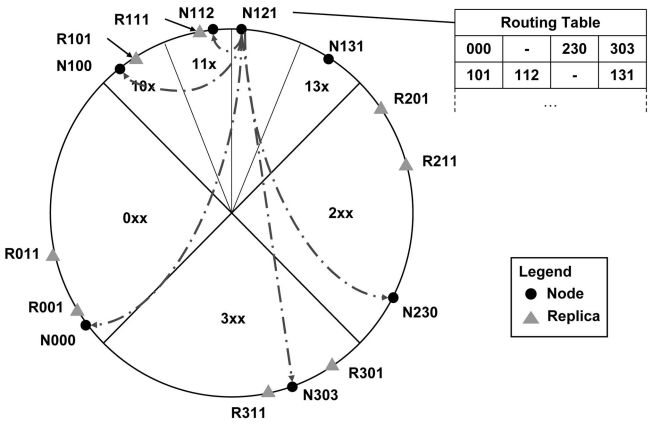

Fig. 2. MAXDISJOINT placement for object 101 to create five disjoint routes from query node 121 ($N = 64,\ B = 4$).

This pattern continues until the entire id space is exhausted. Note that in Pastry each node partitions the id space using prefixes and, therefore, we place replicas by varying their prefixes. However, in general, we are simply spacing replicas such that replicas exist in different parts of each node's partitioning of the id space. In the next section, we provide an algorithm that generates these replica ids.

4.2 Definition of MaxDisjoint Placement

MAXDISJOINT assigns each replica an identifier, which is used to determine its placement. The placement algorithm takes as input N, the identifier space size of the DHT; B, the branching factor; and d, the desired number of disjoint routes. We will prove that MAXDISJOINT creates the desired number of disjoint routes in a later section.

Algorithm 1 (MAXDISJOINT Replica Placement). *To create d disjoint routes, replicas are placed in $m + 1$ rounds, where $m = \lfloor \frac{d-1}{B-1} \rfloor$. Each round consists of $B - 1$ steps except for the final round, which consists of n steps, where $n = (d - 1) \bmod (B - 1)$. In the ith round, B^{i-1} replicas are placed at equally spaced locations over the entire identifier space at each step. In step j of round i, the replica locations are given by:*

$$R_{i,j} = \{k_{i,j}, k_{i,j} + s_i, k_{i,j} + 2s_i, \ldots \\ \ldots k_{i,j} + (B^{i-1} - 1)s_i\} \pmod N,$$

where $k_{i,j} = k + j\frac{N}{B^i}$.

Looking at the example in Fig. 2, consider the placement of an object 101 in a DHT with $B = 4$ and $N = 64$. Suppose we want to create $d = 5$ disjoint routes. Then, we perform

$m+1=2$ rounds of the replica placement algorithm and $n=1$ step in the final round. The replica locations and the corresponding rounds and steps of the algorithm are given below:

$$\begin{array}{ll}\text{Round 1, Step 1}: & 201\\ \text{Round 1, Step 2}: & 301\\ \text{Round 1, Step 3}: & 001\\ \text{Round 2, Step 1}: & 111,\ 211,\ 311,\ 011\end{array}$$

As described in the replica placement algorithm, in each round replicas are placed starting at the master key and working in the direction of increasing identifiers. The algorithm is presented as such for its simplicity. However, the steps within each round can be performed in any order. Each step is functionally equivalent to the others in its round. Therefore, a real implementation may reorder the steps in each round to distribute the replicas more uniformly across the identifier space. This will help to provide load balance and tolerate runs of contiguous failed nodes. For instance, in the example above, to create two disjoint routes, we need only the master object (101) and one of the replicas created in round 1 (001, 201, or 301). Any of these replicas would create the second disjoint route, but choosing replica 301 creates a more uniformly distributed replica set.

4.3 Evaluation of Disjoint Routes

The desired number of disjoint routes d is one of the tunable inputs to the MAXDISJOINT algorithm. Controlling the level of fault tolerance is an important design parameter and, therefore, d is a very useful input. In this section, we will formally prove that the algorithm indeed creates d disjoint routes in support of the intuition provided in earlier sections.

In our analysis, we assume that routing is performed in an identifier space of size N with branching factor B. All of our analytical results are proved within the context of a *full* DHT, but we will show, through experimentation, that these properties hold even in sparsely populated DHTs.

Our principal goal is to prove the following theorem:

Theorem 1. *The MAXDISJOINT Algorithm produces $d \leq (B-1)\log_B N$ disjoint routes from any query node to a key k in a full tree-based routing DHT.*

Proof. Every set $R_{i,j}$ is a unique set of B^{i-1} replicas equally spaced over the entire id space, which implies an interreplica spacing of $\frac{N}{B^{i-1}}$. Consider a source node u and one of its routing table parts $P=[u, u+\frac{N}{B^{i-1}})$. For each $R_{i,j}$, there exists one replica $r_k \in R_{i,j}$ such that $r_k \in P$. Furthermore, the replicas $\{r_k\}$ will be equally spaced within P, separated by an interreplica spacing of $\frac{N}{B^i}$. Regardless of how u chooses to partition P for its routing table, there exists a replica in each part of P. Therefore, there is a unique routing table entry for each replica r_k[4] and a disjoint route is created in every step of the algorithm. Using the definitions of m and n in the algorithm, it is easy to show that $d-1$ steps are performed. The $d-1$ disjoint routes created in the algorithm are combined with the route to k to create a total of d disjoint routes. □

4. It is possible that one of the replicas is placed at u. Even though no routing is performed to fetch this replica, we count it as a route of zero hops.

As a corollary, we give the necessary replication degree to create d disjoint routes.

Corollary 1. *To produce $d \leq (B-1)\log_B N$ disjoint routes from any query node to a key k in a full tree-based routing DHT with branching factor $B>1$, the key k must be replicated at $(n+1)B^m$ locations determined by the MAXDISJOINT Algorithm, where $m=\lfloor\frac{d-1}{B-1}\rfloor$ and $n=(d-1) \bmod (B-1)$.*

Proof. Since we have proved that d disjoint routes are indeed created by the algorithm, we need to show that performing the algorithm with input d places a copy of k at $(n+1)B^m$ locations in the id space. The key k accounts for one location and the remaining locations are determined by the replica placement. Since, B^{i-1} replicas are placed at each step in round i, the total number of replicas is given by:

$$\begin{aligned}&(\text{key}) + \begin{pmatrix}\text{replicas in}\\ \text{first } m \text{ rounds}\end{pmatrix} + \begin{pmatrix}\text{replicas in}\\ \text{last } n \text{ steps}\end{pmatrix}\\ &\quad = 1 + \sum_{i=1}^{m}(B-1)B^{i-1} + nB^m\\ &\quad = (n+1)B^m.\end{aligned}$$

□

We give our replica placement the name MAXDISJOINT because Corollary 1 prescribes the minimum replication degree to create the desired number of disjoint routes using this placement. We state this formally in the following theorem:

Theorem 2. *To produce d disjoint routes from any query node to a key k in a tree-based routing DHT with $B>1$, the key k must be replicated at no fewer than $(n+1)B^m$ locations determined by the MAXDISJOINT Algorithm, where $m=\lfloor\frac{d-1}{B-1}\rfloor$ and $n=(d-1) \bmod (B-1)$.*

Proof. Assume d disjoint routes can be created with $(n+1)B^m-1$ locations determined by the MAXDISJOINT Algorithm, where $m=\lfloor\frac{d-1}{B-1}\rfloor$ and $n=(d-1) \bmod (B-1)$. In other words, d disjoint routes are created with one fewer replica than prescribed by Theorem 1. Let r be the missing replica. We will show that there exists a query node q for which d disjoint routes are not created. Suppose $r \in R_{i,j}$, then let q be a node in $[r, r+j\frac{N}{B^i})$. The replica r is the only replica in $R_{i,j}$ that can create a disjoint route for the query node q. Therefore, step j in round i does not create a disjoint route and we cannot form d disjoint routes with one fewer replica. □

It is worth noting that MAXDISJOINT provides these properties without modifying the underlying routing mechanism. MAXDISJOINT naturally creates disjoint routes using the properties of the tree-based routing scheme.

4.4 Chord as a Tree-Based Routing DHT

To provide consistency with previous work [16], we reconsider Chord as a tree-based routing DHT. It is straightforward to show that Chord finger tables are constructed like tree-based routing tables with $B=2$. Therefore, we can apply Theorem 1 to give the following corollary:

Corollary 2. *To produce $d \leq \log_B N$ disjoint routes from any query node to a key k in a full tree-based routing DHT with*

branching factor $B = 2$, the key k must be replicated at 2^{d-1} equally spaced locations in the ring.

Proof. When $B = 2$, $m = d - 1$ and $n = 0$. Therefore, d disjoint routes are created by replicating k at 2^{d-1} locations in the ring. Furthermore, round i will place 2^{i-1} replicas equally spaced over the id space. Aggregating the replicas placed in each round will result in 2^{d-1} replicas equally spaced over the id space with interreplica spacing $\frac{N}{2^{d-1}}$. □

Note that the claim in Corollary 2 is consistent with the findings in [16].

4.5 Toleration of Runs

We define a run of length l starting at peer m to be the contiguous set of peers with identifiers in the interval $[m, m + l)$. As indicated in [33], an adversary can create imbalance in the distribution of peers in the DHT by appropriately selecting identifiers. In the worst case, an adversary can take control of a contiguous sequence of identifiers or, using our terminology, a run of peers.

We claim that MAXDISJOINT replication can tolerate adversarial runs of bounded length in tree-based routing DHTs. Before proving the tolerable length of a run, we provide some intuition of how a run may be used to disrupt routing in the DHT. Consider a query node q. An adversary can reduce the number of replicas reachable from q by $\frac{1}{B}$ by controlling the peer with identifier $q + \frac{N}{B}$, where N is the identifier space size and prefix matching is performed in base B. This is because all of the replicas in the interval $[q + \frac{N}{B}, q + 2\frac{N}{B})$ are routed through the node $q + \frac{n}{B}$ ($\frac{1}{B}$ of all replicas are in this interval). If the adversary controls a run of nodes ending at $q + \frac{B-1}{B}\frac{N}{B}$, he can control a larger number of replicas.

Theorem 3. *A full tree-based routing DHT with an identifier space of size N, and $r \geq B$ replicas placed using MAXDISJOINT placement can tolerate any adversarial run of length $1 + N(\frac{B-1}{B} - \frac{1}{B^m})$, where $m = \lfloor \log_B r \rfloor$.*

Proof (By Induction). Since we are using MAXDISJOINT placement, it is straightforward to show that $m = \lfloor \log_B r \rfloor = \lfloor \frac{d-1}{B-1} \rfloor$. This implies that the maximum length run tolerable by the DHT changes with each round of the MAXDISJOINT algorithm.

Consider a peer q. For $B \leq r < B^2$, $m = 1$ and there exists a replica k in the interval $[q, q + \frac{N}{B})$. If we assume that the adversary has control of the peer $q + (B - 1)\frac{N}{B}$ (which is the worst case), then we must ensure that k is not in the run. Thus, the maximum length run is $[q + \frac{N}{B}, q + (B - 1)\frac{N}{B})$, which has length $(B - 1)\frac{N}{B} - \frac{N}{B} + 1$ or $1 + N(\frac{B-1}{B} - \frac{1}{B})$.

Assume that the maximum length run tolerable with r replicas is $1 + N(\frac{B-1}{B} - \frac{1}{B^m})$, where $m = \lfloor \log_B r \rfloor$. Consider a query node q. If we assume the adversary takes control of the peer $q + (B - 1)\frac{N}{B}$, then he does not control any peers in the interval $[q, q + \frac{N}{B^m})$. Furthermore, there must be at least one replica in this interval. If another round of the MAXDISJOINT algorithm is performed, then there will be B replicas in this interval separated by a spacing of $\frac{N}{B^{m+1}}$. Thus, the length of the tolerable run increases by $\frac{N}{B^{m+1}}$ to $1 + N(\frac{B-1}{B} - \frac{1}{B^{m+1}})$. □

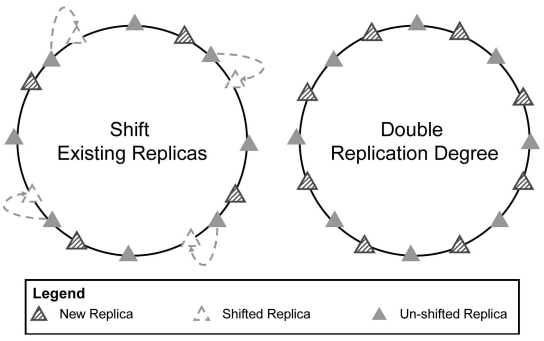

Fig. 3. Two methods for increasing the replication degree when using equally spaced replica placement.

In a Pastry DHT with a typical value of $B = 16$ and 16 replicas, an adversary may control a run of more than 85 percent of the identifier space and there will be a route to at least one replica for every possible query. As the replication degree approaches the identifier space size, the maximum length run tolerable by the DHT approaches $\frac{B-1}{B}N$.

4.6 Adaptivity and Flexibility of MaxDisjoint

There is a noted similarity between the MAXDISJOINT and equally spaced placements. In fact, when $n = 0$, MAXDISJOINT produces equally spaced replica locations. One may argue that equally spaced replica placement is as effective as MAXDISJOINT in creating disjoint routes. However, replica placements have other desirable properties other than their ability to create route diversity; equally spaced placement fails to deliver some of these benefits when $B > 2$.

Two properties in particular are adaptivity and flexibility. Adaptivity is the ability to easily change the replication degree of an object without incurring a large overhead. If the replication degree of an object is changed, we would like to minimize the number of messages exchanged and objects shifted. Flexibility is the ability to easily vary the replication degree of different objects without incurring a large overhead. Certainly, some objects may be more popular or critical than others and require a higher degree of replication. The placement should replicate objects to varying degrees without using excessive time or state at the time of insertion and lookup.

MAXDISJOINT provides adaptivity more effectively than an equally spaced placement. A change in the replication degree must be handled carefully to maintain equal spacing. Consider an increase in the replication degree to add an additional disjoint route. With MAXDISJOINT, the additional replicas are placed at the locations determined by performing another step in the placement algorithm leaving the existing replicas in their current locations.

With equally spaced replicas, additional replicas can be introduced in two different ways. The first option is to compute the equally spaced replica locations for the new replication degree and shift existing replicas, if necessary. In some cases, the existing replica locations will not be a subset of the new replica locations. This implies that existing replicas will have to be shifted, which has a non-negligible cost. The second option is to double the current replication degree such that no existing replicas must be shifted. These two options are depicted graphically in Fig. 3.

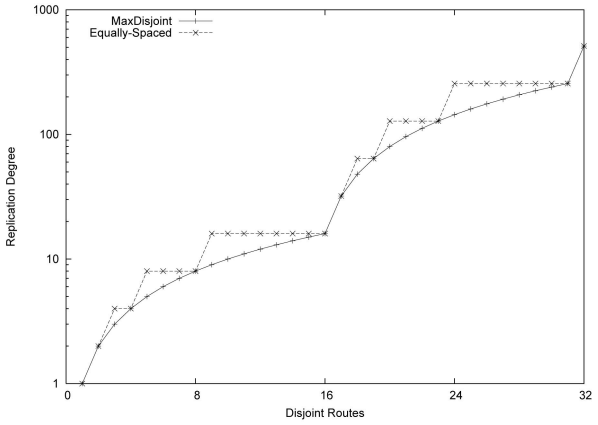

Fig. 4. Replication degree for increasing numbers of disjoint routes ($B = 16$).

Doubling the replication degree avoids the cost of shifting existing replicas, but may come with the added burden of storing an excessive number of replicas. The number of replicas prescribed by Theorem 1 is sufficient. For example, in Fig. 3, only four additional replicas are needed to create an additional disjoint route; however, doubling the replication degree introduces eight new replicas. The excess storage burden created when doubling the replication degree is shown quantitatively for $B = 16$ in Fig. 4. MAXDISJOINT is able to create a desired number of disjoint routes more effectively than equally spaced placement when there is a change in the replication degree.

A sound replica placement also provides flexibility; that is, the replication degree of one data item is not dependent on the replication degree of any other data item. Fortunately, flexibility can be provided easily by MAXDISJOINT placement.

The problem of flexibility arises at the time of lookup. Without knowledge of the replication degree of the target, the replica locations cannot be determined. As a solution, rather than fixing a system-wide replication degree for all data items or storing the replication degrees for all objects, we fix a maximum replication degree r_{max} for all objects. For a data item with replication degree $r \leq r_{max}$, the r replicas will be located at a subset of the r_{max} replica locations. As a result, for some lookups, we may query a replica that does not exist, but we trade off these extra messages for the reduction in node state.

4.7 Implementation

To uniquely identify each replica, we use a *key identifier pair* (k, v), where k is the *key identifier* and v is *virtual key identifier*. For each replica, v gives the location of the master key. By definition, the master key k is denoted by the pair (k, k). We require an ordered pair because two data items may have replicas that reside on the same peer.

When a key k is inserted into the DHT with replication degree d, we first compute the key identifier pairs for each replica: $(k, k), (k_1, k), (k_2, k), \ldots (k_{d-1}, k)$. Once the key identifier pair for each replica is computed, we use the DHT key insertion mechanism to insert the replicas. That is, we perform a lookup for each key identifier and store the replica in their respective locations.

Next, the DHT lookup primitive must be modified to accommodate the new replication scheme. When a peer is queried for a key, the query node must compute the locations of all replicas. The key identifier is used to route to the replicas. Once a replica's home node is found, the key identifier pairs are compared to return the appropriate replica.

It is worth noting that no additional routing table entries are required to route to the replicas. In addition, if the query node dispatches the lookups for the entire replica set simultaneously, there may be an improvement in performance because the query node can return the first correct response received (which may have returned along a route shorter than the route to the master key). However, if the added load of the extra lookups puts strain on the system, the performance may improve only slightly or even degrade. We evaluate this hypothesis through experimentation.

Finally, when peers join or leave the DHT, the DHT join and leave mechanisms can be used by simply ignoring the virtual peer identifiers in each key identifier pair. Note that DHT replica placement schemes that place replicas in the neighborhood of the master key require modification to the node join and leave mechanisms. To maintain the replication degree, replicas will need to be shifted for every join or leave. Ghodsi et al. [12] discuss the effect of churn on symmetric (equally spaced, MAXDISJOINT) replication and show that only $O(1)$ messages are needed to maintain the replication degree for every join or leave compared to $\Omega(r)$ messages for a successor-list (neighbor set) placement, where r is the replication degree.

5 Experiments

To confirm that our analytical results hold for sparsely populated DHTs or DHTs with clustered id spaces, we conducted a series of experiments through simulation. First, to measure the impact of replica placement on routing robustness, we consider the number of disjoint routes created for several replica placements. Furthermore, we find the probability of lookup success when nodes are compromised at random or in a run of several nodes.

Second, we consider a heuristic used for creating route diversity in [3] that we call *neighbor set routing*. We measure its ability to create route diversity and the impact on the probability of lookup success.

Finally, having shown that replica placement can improve routing robustness, we consider the impact of parallel queries on response time.

1. *Simulation Environment*: All of our experiments were performed using a Java-based simulator we developed. We are able to model Chord and Pastry routing, uniform and clustered node distributions, and two adversarial models. Nodes may be compromised at random with some failure probability or in a run of several nodes. The simulator is extensible to model other DHT implementations, node distributions, and adversarial models.

   Each data point in our results is representative of over 100,000 lookups performed in 10 different random node distributions. We simulate a lookup

TABLE 1
Pastry Simulation Parameters

Symbol	Parameter
N	Identifier Space Size
n	Number of Nodes
B	Branching Factor (2^b for Pastry)
r	Replication Degree
f	Fraction of Compromised Nodes
C	Number of Clusters
σ	Cluster Width (standard deviation)

by randomly selecting an uncompromised query node and a target key. In reality, if the query node is compromised, it can affect the outcome of the lookup. However, if we assume that data items are self-verifying, a compromised query node can only cause the client to time out and select another query node. We deem a lookup successful if there exists a route consisting of only uncompromised nodes from the query node to any replica of the target key. If all routes from the query node to the replica set contain a compromised node, then the lookup is deemed to fail.

For most experiments, it is sufficient to compute routes in the network using the appropriate routing protocol. However, to measure response time, it was necessary to modify our simulator to be event driven. The remaining simulation parameters are summarized in Table 1.

2. *Replica Placements Considered*: In our experiments, we consider four replica placements: MAXDISJOINT; *neighbor set*, where replicas are placed at distinct nodes in the neighborhood of the root (e.g., Chord successor list, Pastry leaf set); *random*, where replica locations are uniformly distributed; and *spaced*, where replica identifiers are separated by a uniform spacing s. It is worth noting that two replicas may be placed at the same node with spaced replication.

   In the case of neighbor set placement, some implementations may attempt to reduce load by querying the entire replica set with a single lookup message. This naturally creates route overlap; for a fair assessment, we dispatch a separate lookup for each replica in the replica set.

5.1 Measurement of Disjoint Routes

First, to verify the correctness of our analysis, we measure the average number of disjoint routes created using the considered replica placements. These results are depicted in Fig. 5. For the parameters tested, MAXDISJOINT placement outperforms the other placements in creating disjoint routes.

The neighbor set placement does not create a significant number of disjoint routes as expected. Routes toward keys that are close to each other in the identifier space are likely to converge. Since the neighbor set placement clusters replicas, an adversary can eliminate the entire replica set if he can compromise a node common to all routes destined for that neighborhood. By increasing the route diversity, we eliminate these single points of vulnerability.

The performance of the spaced placement scheme is dependent on the spacing chosen. If the spacing is small, then the spaced placement is very similar to the neighbor

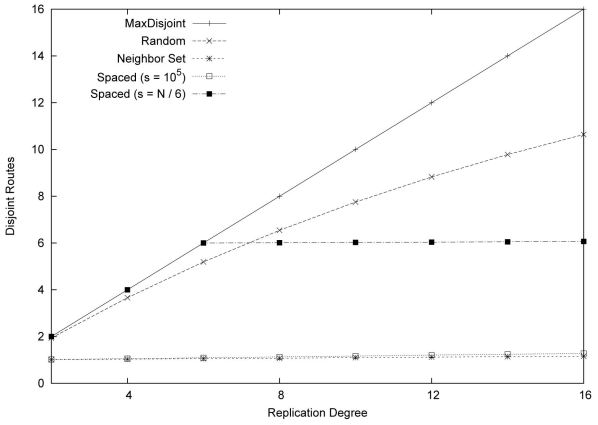

Fig. 5. Number of disjoint routes with increasing replication degree in Pastry ($N = 2^{28},\ n = 8{,}192,\ B = 16$).

set placement. In the extreme case, if the replica spacing is much less than the average internode spacing, two or more replicas may be placed at the same node. We observe this phenomenon for the spaced placement with $s = 10^5$ in Fig. 5. As we increase the spacing, there is a tendency to increase the number of disjoint routes. However as we continue to increase the replication, we will reach a saturation point where replica locations wrap around the identifier space and no additional disjoint routes will be created. We observe this phenomenon when the spacing $s = N/6$. This implies that the spacing should be a function of the replication degree, which is fundamental to how MAXDISJOINT creates disjoint routes.

The random placement creates nearly as many disjoint routes on average because replicas are uniformly distributed. However, there is a significant difference between MAXDISJOINT and random placement in the worst case. We present an argument in support of this claim at the end of this section.

To ensure that this number of disjoint routes could be created in more sparsely populated id spaces, we varied the number of nodes in the DHT starting from 32 and measured the number of disjoint routes for various replication degrees. These data are depicted in Fig. 6. The actual number of disjoint routes converges quickly to the theoretical value, which implies that MAXDISJOINT placement can be used effectively in very sparsely populated networks. In these

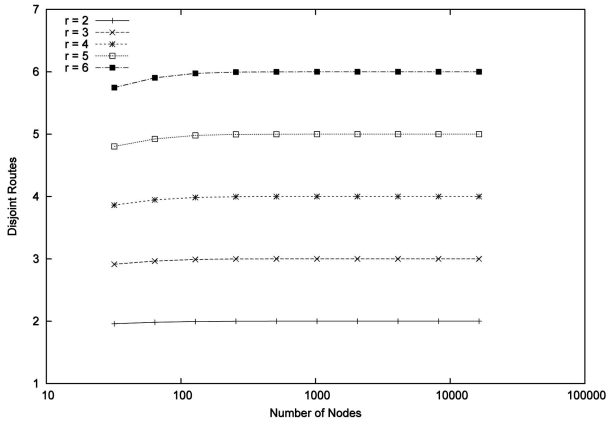

Fig. 6. Number of disjoint routes with increasing number of nodes in Pastry ($N = 2^{28},\ B = 16,\ d \in \{2, 3, 4, 5, 6\}$).

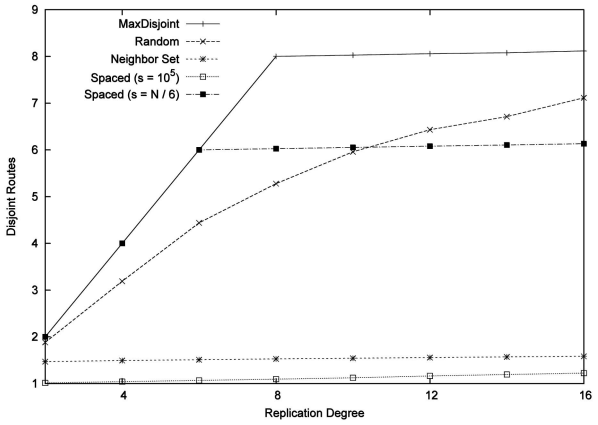

Fig. 7. Disjoint routes for tightly clustered nodes in Pastry ($N = 2^{28},\ n = 8{,}192,\ \sigma = 2^{15}$).

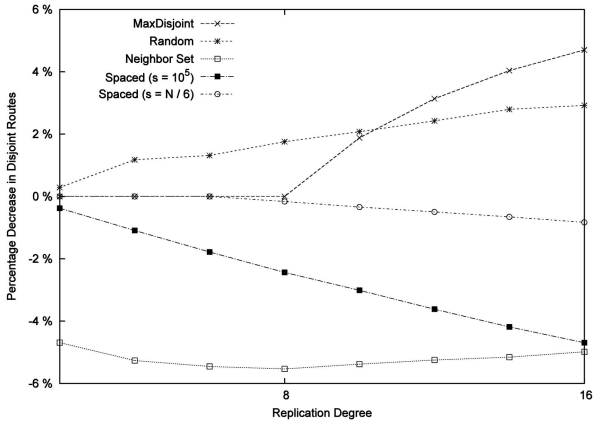

Fig. 9. Percent decrease in disjoint routes from uniformly distributed nodes to a clustered distribution in Pastry ($N = 2^{28},\ n = 8{,}192,\ C = 4,\ \sigma = 10^{16}$).

results, the theoretical number of disjoint routes is achieved for loads greater than 1,000 nodes, which is less than a thousandth of a percent of the identifier space.

5.2 Resilience to Node Clustering

To model the population distribution that may result from the collusion of several malicious nodes, a series of experiments were run on clustered DHTs. To model a clustered distribution, four node ids were randomly selected as cluster means such that the clusters are nonoverlapping and unpopulated gaps exist in the id space. The cluster density was tuned using the standard deviation of the Gaussian distribution centered around each cluster mean. The number of disjoint routes created for $\sigma = 2^{15}$ and $\sigma = 2^{16}$ are shown in Figs. 7 and 8, respectively.

Clustering can marginally reduce the number of disjoint routes created for small replication degrees, but the impact is more dramatic when creating a larger number of disjoint routes. The impact of clustering is intensified with tighter clusters (i.e., decreasing σ) because replicas are not located at the positions that MAXDISJOINT prescribes. Although the replica ids are properly assigned, the replicas themselves are confined to the clusters. One or more of the replicas may lay in the unpopulated gaps between clusters. These replicas get "pushed" to next cluster in the id space, possibly eliminating a disjoint route. This is more likely to happen when creating a large number of disjoint routes (which requires more replicas).

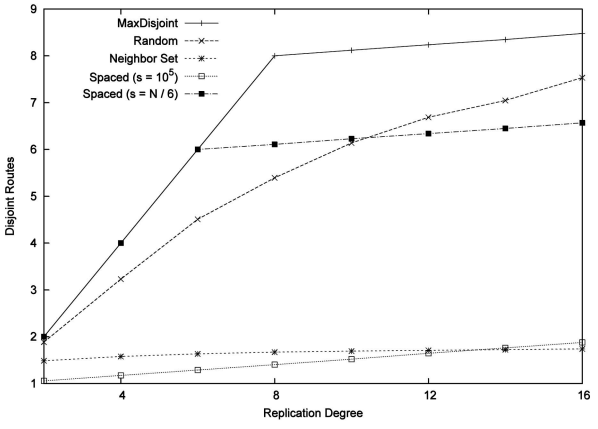

Fig. 8. Disjoint routes for loosely clustered nodes in Pastry ($N = 2^{28},\ n = 8{,}192,\ \sigma = 2^{16}$).

We computed the percentage decrease in disjoint routes that results from clustering. This analysis produced some very interesting results that are depicted in Fig. 9. One interesting conclusion is that not all replica placements are negatively affected by node clustering. In particular, when replicas are placed close together in the id space, e.g., neighbor set or spaced (for small spacings) placement, there can actually be an increase in the number of disjoint routes created. This is because a long string of closely packed replicas may span a gap between clusters, which pushes some of the replicas to another cluster. The clustering actually helps distribute the replicas better resulting in more disjoint routes. Nevertheless, the 4-6 percent increase in the number of disjoint routes is insubstantial because these placements fail to create a sufficient number of disjoint routes with uniformly distributed nodes.

To the contrary, the MAXDISJOINT and random placements are affected negatively by clustering. Placements that distribute replicas in the id space may place a replica in the unpopulated gaps between clusters. These replicas get pushed to a cluster and a disjoint route may be lost. This phenomenon takes effect for MAXDISJOINT when $r = 8$. At this point, the interreplica spacing is 2^{25}. If we use two standard deviations to capture 95 percent of each cluster, we can estimate the intercluster gaps to about 2^{26}, which is twice the interreplica spacing. That implies that about half of all replicas are located in the gaps between clusters. Note that the random placement suffers from this problem over the entire range of replication degrees because it is not guaranteed to distribute replicas across the entire id space. Furthermore, as we increase the replication degree, more replicas will be located in the gaps and we lose the benefit of increased replication degree. Nevertheless, the three to five percent decrease in the number of disjoint routes is relatively insignificant because MAXDISJOINT creates so many more disjoint routes than the other placements in a uniformly populated id space.

5.3 Impact of Replica Placement on Routing Robustness

To demonstrate that the number of disjoint routes has a significant impact on the robustness of the DHT, we measure the probability of lookup success with a random

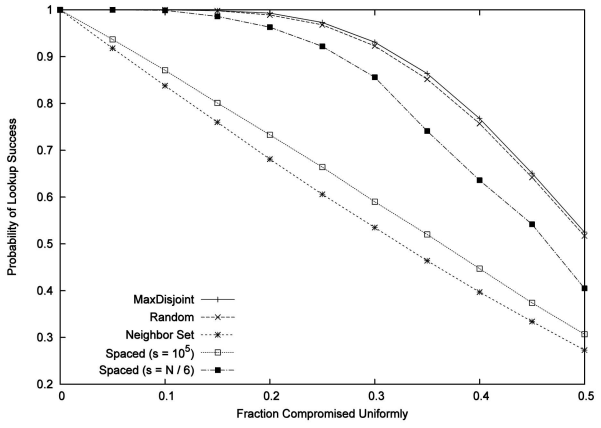

Fig. 10. Probability of routing success with uniformly compromised nodes in Pastry ($N = 2^{28},\ n = 8{,}192,\ B = 16,\ r = 8$).

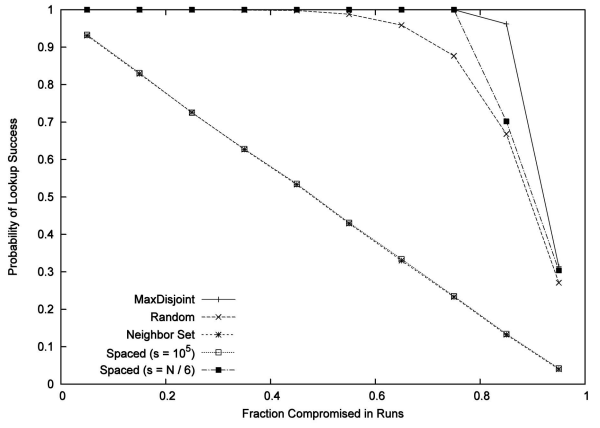

Fig. 11. Probability of routing success with runs of compromised nodes in Pastry ($N = 2^{28},\ n = 8{,}192,\ B = 16,\ r = 16$).

fraction of faulty nodes. A faulty node may be a failed or compromised node. The probability of routing success is shown in Fig. 10.

These results indicate a correlation between the number of disjoint routes and the probability of lookup success. The MAXDISJOINT and random placements most effectively create disjoint routes and, thus, have the most positive impact on the probability of routing success. Furthermore, we can conclude that using MAXDISJOINT placement instead of neighbor set placement can dramatically improve the probability of routing success. With a quarter of nodes compromised at random, greater than 97 percent of all lookups can be resolved successfully with MAXDISJOINT placement compared to only 60 percent with neighbor set placement.

With the network configuration in Fig. 10, the MAXDISJOINT placement creates eight disjoint routes. Therefore, an adversary could prevent the correct resolution of a given query by compromising only eight nodes. However, with far more nodes than that compromised at random, nearly all queries are resolved successfully. This is a strong indication that replica placement plays a critical role in providing robustness in DHTs.

Our analysis indicates that MAXDISJOINT should be able to tolerate runs of compromised nodes better than placements that cluster replicas closely together. Experimental results that confirm this hypothesis are depicted in Fig. 11. With 16 replicas and 85 percent of the identifier spaced compromised in a run, greater than 96 percent of lookups are resolved successfully with MAXDISJOINT placement compared to only 13 percent with neighbor set placement. Furthermore, these results demonstrate an exploit of random placement. With moderately long runs, MAXDISJOINT is able to successfully resolve a higher fraction of queries than the random placement. For example, with 85 percent of the identifier spaced compromised in a run, only 66 percent of lookups are resolved successfully with a random placement. This is because the random placement may cluster replicas for a significant fraction of keys. We investigate this further in the following section.

5.4 MaxDisjoint versus Random Placement

In the experimental data presented thus far, random placement seems to have performed on par with MAXDISJOINT on average. We argue, however, that a truly random placement is expensive to implement in practice and the average performance of a random placement does not trickle down to the worst case.

The worst case attack against a replica placement is to target the replica locations themselves. One may argue that a placement with random locations would make it more difficult for an adversary to determine and target the replica set than a deterministic approach like MAXDISJOINT. However, a truly random placement also complicates matters for the query node. The query node must be able to compute the replica locations for any object at the time of lookup. To avoid keeping a considerable amount of state at each node, the only practical proposal of which we are aware is to use multiple hash functions to generate "random" replica locations. However, this implementation is not truly random and, with knowledge of the hash functions, it is as vulnerable as MAXDISJOINT to attacks that target all replica locations for a particular object. This makes "random" replication simply a different deterministic placement that is, in a sense, an approximation of equally spaced replication. Yet, as we show next, the performance of random replication falls considerably short of MAXDISJOINT.

The performance results we have shown thus far depict the average number of disjoint routes created. To consider the worst case performance, we measured the minimum number of disjoint routes created for MAXDISJOINT and random placement. These results are depicted in Fig. 12.

The results in Fig. 12 confirm our hypothesis of the worst case performance of a random placement. For a few objects, the placement may create as few as half of the desired number of disjoint routes. To establish that the worst case was not an isolated case, we measured the number of disjoint routes created over all lookups. The cumulative distribution of the number of disjoint routes with eight replicas is shown in Fig. 13.

For the case depicted in Fig. 13, MAXDISJOINT created the expected eight disjoint routes for every single lookup. Random placement created six or fewer disjoint routes for 45 percent of lookups and, in some cases, it created as few as four, or only half of the number produced by MAXDISJOINT. We observed the same behavior in other cases as well. Therefore, if delivering a consistent level of fault tolerance across all lookups is a design constraint, random placement is not a reasonable solution.

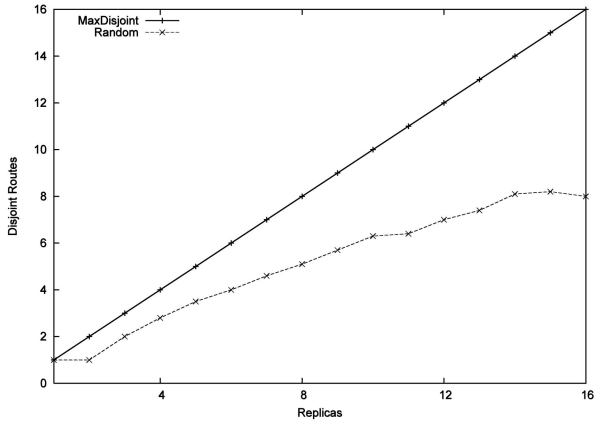

Fig. 12. Worst-case performance of MAXDISJOINT and random replica placement in Pastry. The minimum number of disjoint routes created over all lookups is shown ($N = 2^{20},\ n = 8{,}192$).

5.5 Replica Placement and Neighbor Set Routing

We believe replica placement is an efficient way of creating disjoint routes because it does not require significant modification to the underlying DHT routing scheme. Other works have considered reusing the existing routing information to create route diversity [3], [24]. Although these approaches do not create provably disjoint routes, we believe there is value in introducing some additional form of route diversity. Furthermore, we believe that these techniques could be combined with our replica placement to provide additional benefit. To evaluate this claim, we consider the route diversity technique introduced by Castro et al. [3], which we call *neighbor set routing*.

Castro et al. use neighbor set routing to find diverse routes toward the neighborhood of a key. To create diverse routes, messages are routed via the neighbors of the source node. This is depicted graphically in Fig. 14. Castro et al. claim that this technique is sufficient in the case when replicas are distributed uniformly over the identifier space, as in CAN and Tapestry. We consider the ability of neighbor set routing to create diverse routes to a replica to enhance the routing robustness of MAXDISJOINT.

To evaluate the relative impact of replica placement and neighbor set routing, we consider four scenarios:

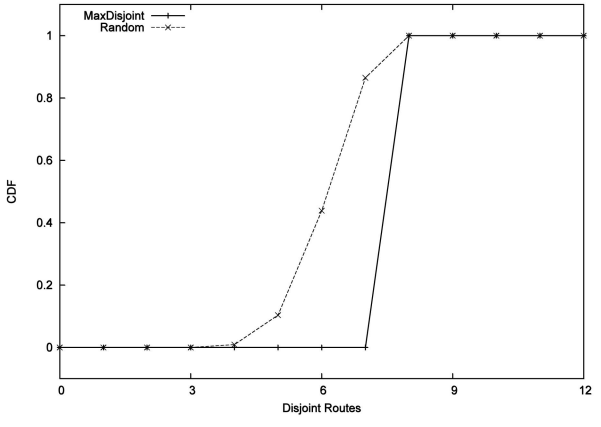

Fig. 13. Cumulative distribution of disjoint routes created for MAXDISJOINT and random placement in Pastry ($N = 2^{20},\ n = 8{,}192,\ r = 8$).

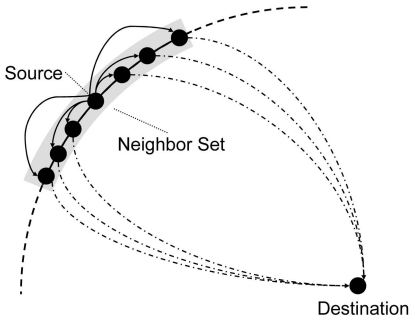

Fig. 14. Graphical depiction of neighbor set routing.

1. neither replication nor neighbor set routing,
2. only neighbor set routing through eight neighbors,
3. only MAXDISJOINT placement with eight replicas, and
4. both neighbor set routing and MAXDISJOINT placement.

These results are depicted in Fig. 15.

Both MAXDISJOINT placement and neighbor set routing can have a positive impact on the probability of lookup success. However, the trend is stronger with replica placement. With a quarter of nodes compromised at random, over 97 percent of lookups are resolved successfully with MAXDISJOINT placement compared to 63 percent with neighbor set routing alone. At best, neighbor set routing can create independent routes, since all paths will converge at the destination. If the destination node is compromised, no amount of route diversity can increase the probability of lookup success.

Nonetheless, the added route diversity that neighbor set routing provides can benefit MAXDISJOINT placement, especially with a large fraction of compromised nodes. With 50 percent of nodes compromised, the probability of lookup success of using MAXDISJOINT placement improves from 52 to 84 percent with neighbor set routing.

5.6 Parallel Queries and Response Time

Finally, since replica placement seems to be a reasonable method for improving routing robustness, it is natural to

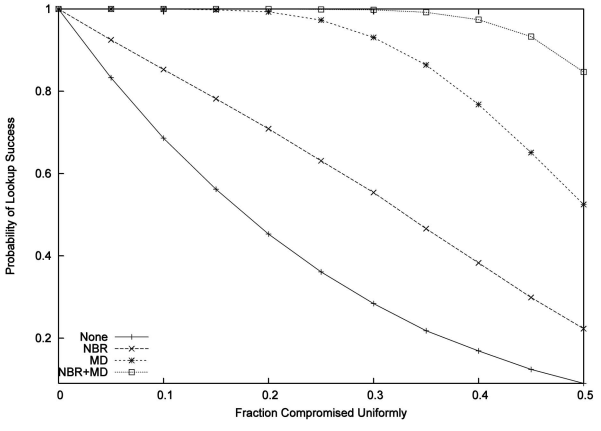

Fig. 15. Probability of routing success with neither replication nor neighbor set routing (None), neighbor set routing only (NBR), MAXDISJOINT placement only (MD), and both neighbor set routing and MAXDISJOINT placement together (NBR+MD) ($N = 2^{28},\ n = 8{,}192,\ B = 16,\ r = 8$).

consider some practical concerns of using replication. When querying a replica set, response time may be reduced by querying the entire replica set in parallel. Alternatively, this could have a significant impact on the system load, which may result in congestion and increased response time. Therefore, we consider three replica query strategies: parallel, sequential, and hybrid.

Unlike its counterpart, the sequential strategy queries replicas one at a time waiting for a response between replicas. This strategy controls system load at the expense of response time. With very few failures, the sequential strategy should result in reasonable response times without inducing a significant load on the network. However, the response time may not be as resilient to an increase in failure rate as the parallel strategy.

We also consider a hybrid approach in which replicas are queried in sets of two or more replicas. Sets are queried one at a time waiting for a response before querying the next set. With this strategy, the trade-off can be managed using the set size to tune the response time with load. In figures, the hybrid strategy series are labeled "Hybrid-S," where S denotes the set size.

To present realistic response times, we model the internode delay with a log-normal distribution with $\mu = 60$ ms and $\sigma = 50$ ms and total response time as the sum of internode delays along a route. The log-normal distribution parameters were selected using results from a study of TCP connection round trip times [1].

We extend our fault model to assume that failed nodes correctly forward lookups to create added system load, but return incorrect responses that the query node is able to detect. Therefore, a failed route will result in the same system load as a successful route, but will add to the overall response time of the lookup. In a real system where a failure may result in no response at all, it may be necessary to use a time-out for the sequential and hybrid schemes.

To measure the effect of message queuing, we measure response time for lookup rates varying from 1×10^2 to 1×10^7 lookups per second. Since the underlying physical topology is difficult to predict and we are more concerned with the queuing that results from our query strategy, we modeled queuing in the overlay, rather than in the physical network. We assume that each node in the overlay is a leaf node in the underlying physical topology and has a 1 megabit per second link to its gateway router. Furthermore, we assume that the message size is 1 kilobyte, which is consistent with real Pastry implementations.

The average response times for the discussed replica query strategies are depicted in Fig. 16. We repeated the experiment for f equal to 5, 10, and 15 percent; these results are shown in Figs. 16a, 16b, and 16c, respectively.

The trend across these graphs confirms our intuition that the response time of the sequential strategy increases with the fraction of nodes compromised in the network. Furthermore, the response time does not seem to vary significantly with changes in the lookup rate because the effects of an increased lookup rate are not compounded by the parallelization of replica queries.

To the contrary, the parallel strategy is not resilient to changes in the lookup rate. As the lookup rate increases beyond 10,000 lookups per second, the response time of the parallel strategy increases dramatically. With a relatively low fraction of nodes compromised (Fig. 16a), the sequential strategy can even outperform the parallel strategy in terms of response time. The additional load resulting from parallelization results in message queuing and delayed responses. This effect may be stronger in small networks, which have reduced capacity, and in networks with a higher replication degree.

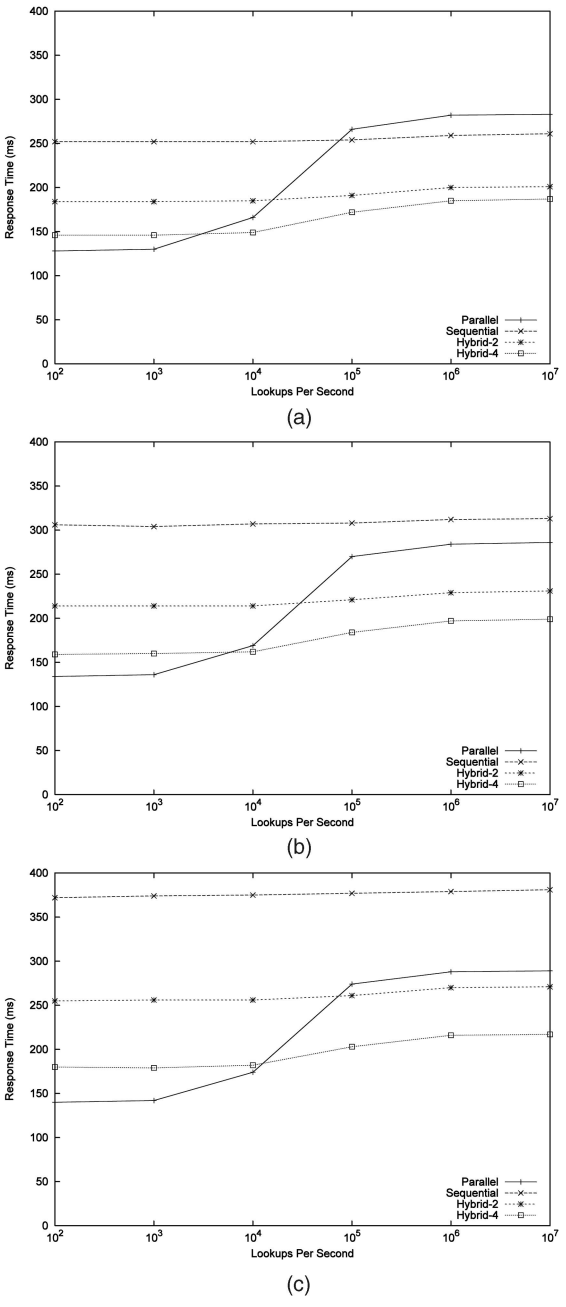

Fig. 16. Response time (ms) for successful lookups with increasing lookup rate with $f =$ (a) 5 percent, (b) 10 percent, and (c) 15 percent ($N = 2^{28},\ n = 8{,}192,\ B = 16, r = 16$).

The hybrid strategies offer a suitable trade-off between the parallel and sequential strategy. The set size can be increased to reduce response times and improve resilience to changes in the fraction of compromised nodes. To improve the resilience to changes in lookup rate, the set size should be decreased. The value of this parameter should be determined by the needs of the application. For example, caller lookup rates in a voice over IP (VoIP) application may change wildly with the time of day and a smaller set size should be used to control congestion.

6 Discussion

In this paper, we characterized a class of DHTs, which employ a tree-based routing scheme. We proved that for every DHT of this class there exists a replica placement that can create a provable number of disjoint routes. We defined MAXDISJOINT, a replica placement that creates disjoint routes in a full distributed hash table that employs a tree-based routing scheme. Through simulation, we showed that this placement creates disjoint routes effectively in DHTs that are sparsely populated. In addition, MAXDISJOINT creates disjoint routes without modification of the underlying routing scheme; therefore, its implementation is independent of the underlying DHT chosen, provided that the underlying DHT employs tree-based routing. Furthermore, we demonstrated that disjoint routes have a positive impact on routing robustness and the probability of routing success when nodes are compromised at random or in runs. Specifically, we showed that a vast majority of queries can be resolved successfully even with a quarter of nodes compromised.

We also compared our replica placement with another mechanism for creating route diversity called neighbor set routing. MAXDISJOINT has a stronger impact on routing robustness than neighbor set routing; however, when the mechanisms are combined, substantial benefit is gained, especially with a large fraction of nodes compromised. Therefore, using two or more route diversity mechanisms, like replica placement and neighbor set routing, can have a positive impact on routing robustness.

Finally, we considered some of the practical limitations of using MAXDISJOINT in a real implementation; that is, we evaluated the choice of the replica query strategy on response time. Of particular concern was the impact of a parallel strategy on the system load and, as a result, the response time. We observed that the parallel strategy is adversely affected by an increase in the lookup rate; however, it is resilient to changes in the fraction of nodes compromised. Conversely, the sequential strategy is not significantly affected by changes in the lookup rate, but the response time increases with the fraction of nodes compromised. As a solution, we offered a hybrid scheme, in which sets of two or more replicas are queried sequentially. We explained how the set size could be tuned to give a reasonable response time and resilience to changes in the lookup rate.

Our assumption of self-certifying data is a common one in the field [3], [7], [27]. When object contents do not change frequently, one option is to use the hash of the contents of an object as its key. When contents are retrieved, they can be used to compute a new hash, which can then be verified against the key value. This is the approach taken, for example, in CFS [7]. This implies that queriers know the hash of the contents they are seeking, which can be ensured by periodically downloading a master list of names and hashes from a centralized server or from one of a distributed set of replicated servers. If objects are intended to be written only by trusted authorities but can be read by any peer, then the contents can be digitally signed. This approach is used, for example, in PAST [27]. Another potential application of this approach is domain name service, which was one of the earliest proposed DHT applications [33]. In this application, use of DNSSEC [18] would provide the necessary certification mechanism. For data with multiple writers, consistency among different replicas is an issue, which necessitates Byzantine-fault-tolerant replication and requires that clients retrieve multiple replicas and perform a voting operation to ensure correctness. This situation is more complicated than what we describe in this paper. However, our approach can still be used as a building block within this more complicated scheme, similar to what is described in [3]. In [3], Castro et al. describe a scheme where replica groups are stored using self-certifying data, peers perform certified lookups of replica groups, and then peers execute a Byzantine-fault-tolerant read operation to collect a correct copy of the object.

Concerning the types of attacks that MAXDISJOINT tolerates, we have considered random combinations and contiguous runs of compromised nodes that can act arbitrarily in routing and responding to queries. Another attack that has been identified is the eclipse attack, where a small group of nodes attempts to have themselves placed in the routing tables of as many nodes as possible in the system [30]. In an eclipse attack, a small group of nodes can have a large impact on routing performance. While we do not explicitly consider eclipse attacks herein, our replica placement scheme can work seamlessly with proposed approaches to handle eclipse attacks such as degree bounding [30], which do not alter the tree-based routing structure of the network.

Another recently explored category of p2p networks is "one-hop" DHTs [14], [15]. These DHTs create routes with $O(1)$ hops by restructuring the id space and maintaining more routing state at each node. One of the proposed DHTs in [14] is a true one-hop DHT, where every node maintains a complete routing table with information about every other node in the network. In this situation, since all paths are one hop long, then any set of nodes holding r replicas forms r disjoint paths from any query node. Thus, replica placement is not an issue in true one-hop networks. However, the second proposed DHT in [14] (the two-hop DHT) and the Kelips network in [15] both have hierarchical routing structures, albeit simple one-level hierarchies. It is interesting to note that the principles of MAXDISJOINT are applicable to the two-hop network of [14] and to Kelips. This is because in both networks, nodes are partitioned (into affinity groups in Kelips and into slices in [14]), different first hops from a query node reach nodes in different parts, and routes do not leave their specific part after the first hop. Thus, by placing replicas in different parts of the partition, k disjoint paths will be produced from any query node, where k is the number of parts ($O(\sqrt{n})$ in Kelips and a configurable parameter in [14]). This replica placement is a simple special case of MAXDISJOINT and is an obvious choice for these two network structures. Nevertheless, it is interesting that the same principles that led to the design of MAXDISJOINT apply to these "one-hop" networks also.

We believe that both one-hop DHTs and DHTs with multilevel hierarchies will be used in the future for different types of applications. Applications with an extremely large user base that are not very latency sensitive in the lookup phase, e.g., BitTorrent and Skype, will continue to prefer the better scalability of multilevel hierarchies. Also, the availability of open-source software such as FreePastry makes it a popular choice for research use and for rapid development and deployment of new p2p applications [11]. Applications that are latency sensitive for lookups and do not need to scale to massive numbers of clients will prefer one-hop DHT technology.

ACKNOWLEDGMENTS

This research was funded in part by the US National Science Foundation under Grant ITR-NHS-0427700.

REFERENCES

[1] J. Aikat, J. Kaur, F.D. Smith, and K. Jeffay, "Variability in TCP Round-Trip Times," *Proc. ACM SIGCOMM Internet Measurement Conf. (IMC '03),* pp. 279-284, 2003.

[2] M.S. Artigas, P.G. Lopez, and A.F.G. Skarmeta, "A Novel Methodology for Constructing Secure Multipath Overlays," *IEEE Internet Computing,* vol. 9, no. 6, pp. 50-57, Nov./Dec. 2005.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach, "Secure Routing for Structured Peer-to-Peer Overlay Networks," *Proc. Symp. Operating Systems Design and Implementation (OSDI '02),* pp. 299-314, 2002.

[4] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proc. Symp. Operating Systems Design and Implementation (OSDI '99),* pp. 173-186, 1999.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," *Proc. ACM Symp. Operating Systems Principles (SOSP '03),* pp. 298-313, 2003.

[6] Y. Chen, R.H. Katz, and J. Kubiatowicz, "Dynamic Replica Placement for Scalable Content Delivery," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* pp. 306-318, 2002.

[7] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide Area Cooperative Storage with CFS," *Proc. ACM Symp. Operating Systems Principles (SOSP '01),* pp. 202-215, 2001.

[8] J.R. Douceur, "The Sybil Attack," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* pp. 251-260, 2002.

[9] J.R. Douceur and R.P. Wattenhofer, "Large-Scale Simulation of Replica Placement Algorithms for a Serverless Distributed File System," *Proc. Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems (MASCOTS '01),* pp. 311-319, 2001.

[10] M.J. Freedman, E. Sit, J. Cates, and R. Morris, "Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer," *Proc. Revised Papers from the First Int'l Workshop Peer-to-Peer Systems (IPTPS '01),* pp. 121-129, 2002.

[11] "FreePastry," http://freepastry.org/, freepastry.org, May 2009.

[12] A. Ghodsi, L.O. Alima, and S. Haridi, "Symmetric Replication for Structured Peer-to-Peer Systems," *Proc. Int'l Workshops Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P '05),* pp. 74-85, 2005.

[13] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The Impact of DHT Routing Geometry on Resilience and Proximity," *Proc. ACM SIGCOMM '03,* pp. 381-394, 2003.

[14] A. Gupta, B. Liskov, and R. Rodrigues, "Efficient Routing for Peer-to-Peer Overlays," *Proc. Conf. Symp. Networked Systems Design and Implementation (NSDI '04),* 2004.

[15] I. Gupta, K. Birman, P. Linga, A. Demers, and R.V. Renesse, "Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '03),* pp. 160-169, 2003.

[16] C. Harvesf and D.M. Blough, "The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables," *Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P '06),* pp. 57-66, 2006.

[17] C. Harvesf and D.M. Blough, "The Design and Evaluation of Techniques for Route Diversity in Distributed Hash Tables," *Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P '07),* pp. 237-238, 2007.

[18] "DNS Security Extensions," IETF, http://www.dnssec.net/, 2009.

[19] Q. Lian, W. Chen, and Z. Zhang, "On the Impact of Replica Placement to the Reliability of Distributed Block Storage Systems," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS '05),* pp. 187-196, 2005.

[20] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* pp. 53-65, 2002.

[21] J.W. Mickens and B.D. Noble, "Concilium: Collaborative Diagnosis of Broken Overlay Routes," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '07),* pp. 225-234, 2007.

[22] G. On, J. Schmitt, and R. Steinmetz, "The Effectiveness of Realistic Replication Strategies on Quality of Availability for Peer-to-Peer Systems," *Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P '03),* pp. 57-64, 2003.

[23] C.G. Plaxton, R. Rajaraman, and A. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Proc. ACM Symp. Parallel Algorithms and Architectures (SPAA '97),* pp. 311-320, 1997.

[24] M. Portmann, S. Ardon, and A. Seneviratne, "Mitigating Routing Misbehaviour of Rational Nodes in Chord," *Proc. Symp. Applications and the Internet (SAINT '04),* pp. 541-545, 2004.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM '01,* pp. 161-172, 2001.

[26] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. ACM Middleware '01,* pp. 329-350, 2001.

[27] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. ACM Symp. Operating Systems Principles (SOSP '01),* 2001.

[28] N. Saxena, G. Tsudik, and J.H. Yi, "Admission Control in Peer-to-Peer: Design and Performance Evaluation," *Proc. ACM Workshop Security of Ad Hoc and Sensor Networks (SASN '03),* pp. 104-113, 2003.

[29] A. Serjantov, "Anonymizing Censorship Resistant Systems," *Proc. Revised Papers from the First Int'l Workshop Peer-to-Peer Systems (IPTPS '01),* pp. 111-120, 2002.

[30] A. Singh, M. Castro, P. Druschel, and A. Rowstron, "Defending against Eclipse Attacks on Overlay Networks," *Proc. ACM SIGOPS '04,* pp. 115-120, 2004.

[31] E. Sit and R. Morris, "Security Considerations for Peer-to-Peer Distributed Hash Tables," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* pp. 261-269, 2002.

[32] M. Srivatsa and L. Liu, "Vulnerabilities and Security Threats in Structured Peer-to-Peer Systems: A Quantitative Analysis," *Proc. IEEE Ann. Computer Security Applications Conf. (ACSAC '04),* pp. 252-261, 2004.

[33] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM '01,* pp. 149-160, 2001.

[34] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE J. on Selected Areas in Comm.,* vol. 22, no. 1, pp. 41-53, Jan. 2004.

Cyrus Harvesf received the BS degree in computer engineering and the MS and PhD degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, in 2004, 2006, and 2008, respectively. Since the spring of 2009, he has been a software design engineer at Microsoft Corporation in Redmond, Washington, where he focuses on the delivery of cloud-based access control in the Windows Azure Platform.

Douglas M. Blough received the BS degree in electrical engineering and the MS and PhD degrees in computer science from the Johns Hopkins University, Baltimore, Maryland, in 1984, 1986, and 1988, respectively. Since the fall of 1999, he has been a professor of electrical and computer engineering at the Georgia Institute of Technology, where he also holds a joint appointment in the School of Computer Science. From 1988 to 1999, he was on the faculty of electrical and computer engineering at the University of California, Irvine. He was a program cochair for the 2009 IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS) and the 2000 International Conference on Dependable Systems and Networks (DSN). He has been an associate editor of the *IEEE Transactions on Computers* and the *IEEE Transactions on Parallel and Distributed Systems*, and is currently an associate editor of the *IEEE Transactions on Mobile Computing*. His research interests include dependability and security of distributed systems, and design and evaluation of wireless multihop networks. He is a senior member of the IEEE and the IEEE Computer Society.