The Design and Evaluation of Techniques for Route Diversity in Distributed Hash Tables

Cyrus Harvesf and Douglas M. Blough
Georgia Institute of Technology
School of Electrical and Computer Engineering
Atlanta, GA 30332-0250
{charvesf,dblough}@ece.gatech.edu

Abstract

To achieve higher efficiency over their unstructured counterparts, structured peer-to-peer systems hold each node responsible for serving a specified set of keys and correctly routing lookups. Unfortunately, malicious participants can abuse these responsibilities to deny access to a set of keys or misroute lookups. We look to address both of these problems through replica placement. We present a replica placement scheme for any distributed hash table that uses a prefix-matching routing scheme and evaluate the number of replicas necessary to produce a desired number of disjoint routes. We show through simulation that this placement can make a significant improvement in routing robustness over other placements. 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 failed 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.

1. Introduction

Structured peer-to-peer networks provide the benefits of efficiency, scalability, resilience, and self-organization when building distributed applications. As such, this architecture has been used in constructing lookup services (e.g., Chord [21], Pastry [16]) that can serve as a platform for a multitude of applications, including multicast, publish/subscribe, and persistent storage.

The structured architecture holds each node responsible for serving data items and correctly routing messages. Malicious nodes may abuse these responsibilities and act to tamper with the correct functioning of the system. Because of the multitude of potentially unforeseen vulnerabilities, it is important to design robustness into the system so it has the ability to function properly, even in the presence of unexpected attacks. For example, providing diverse routes between nodes can improve routing robustness and protect against many routing attacks.

In our work, we consider two classes of attacks: (1) malicious routing and (2) storage and retrieval attacks. The most commonly accepted defenses to these attacks are route diversity and replication, respectively. Our solution integrates these two approaches into a replica placement scheme that can be tuned to produce a desired number of disjoint routes.

2. Background

Sit and Morris [18] identify the major threats to structured peer-to-peer systems to be:

1. Routing attacks, in which a malicious node may misroute lookups or attempt to corrupt routing tables;
2. Storage and retrieval attacks, in which a malicious node may deny access to a key it serves; and
3. Miscellaneous attacks, in which a node may act arbitrarily to tamper with the correct operation of the system. These attacks include Sybil and Eclipse [6, 17] as well as denial of service (DoS) attacks [8, 20].

Our work mitigates the first two threat types. As suggested in [19], routing attacks can be mitigated with alternative lookup paths. To this end, they propose the notion of *independent lookup paths*. Two routes are said to be independent if they share no common node other than the source and destination nodes. Independent routes help improve routing robustness, but do not address storage and retrieval attacks because the routes share a common destination node. We define *disjoint routes* to be routes that share

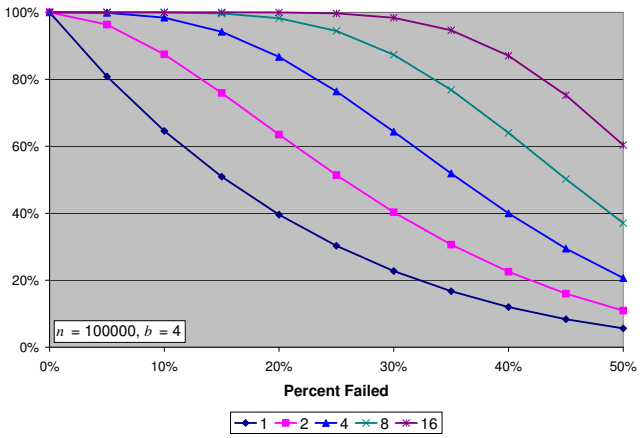

Figure 1. Routing Success Probability with Increasing Failure Rate for 1, 2, 4, 8 and 16 Disjoint Routes.

no common node other than the source. This is a more robust notion of route diversity.

To address storage and retrieval attacks, we use replication as Sit and Morris recommend. The novel contribution of our approach is that the replica *placement* naturally produces disjoint routes without significant modification to the underlying peer-to-peer system.

The impact of disjoint routes on routing robustness can be understood with a little analysis. If we assume that the probability that a node has failed is f and the average route length (for Pastry) is $R = \log_{2^b} n$, where b is the number of bits per digit in the prefix-matching scheme and n is the number of nodes, then the probability P_R that a route contains a failed node is $P_R = 1 - (1 - f)^R$. If we assume we have d disjoint routes, then the probability P that at least one of the d disjoint routes contains no failed nodes is $P = 1 - {P_R}^d$. We call P the *routing success probability*. The routing success probabilities for 1, 2, 4, 8 and 16 disjoint routes in a DHT with $n = 100,000$ and $b = 4$ are shown in Figure 1.

In order for disjoint routes to improve the robustness of our system, we require that the keys stored in the system be self-verifying; that is, we assume that a key's integrity can be verified by the client. This is necessary to detect when a node returns an incorrect entry. For types of data where self-verification is not natural, we can generate key identifiers by hashing the key object as in [5]. The client can verify the key's integrity by comparing its hash with the key identifier. Castro et al. [3] discuss self-verifying data as well as a method for managing mutable self-verifying data.

Under these assumptions, our system can tolerate attacks by up to $d - 1$ malicious nodes, where d is the number of disjoint routes. A similar approach was proven for Chord in [10]; however, we present a general solution applicable to all DHTs that use prefix-matching routing.

3. Related Work

To our knowledge, outside of our own research, no previous work has considered using replica placement to improve routing robustness in DHTs. However, many works have looked at peer-to-peer routing and replica placement independently. We outline these works in the following subsections and distinguish them from our work.

3.1. Peer-to-Peer Routing

Many works have looked to improve the security and robustness of peer-to-peer routing. Several approaches have been centered around the notion of alternative paths [19].

Artigas, et al. [2] use a multipath concept to secure routing. They propose Cyclone, an equivalence-based routing scheme that is built upon an existing peer-to-peer structured overlay. Independent lookup paths can be created by routing along different equivalence classes. The key difference between Cyclone and our work is that Cyclone creates independent lookup paths rather than disjoint paths. Since the paths created do not differ in the destination node, Cyclone will not prevent storage and retrieval attacks. Our work produces disjoint routes to several appropriately placed replicas, thereby preventing a limited number of attacks of this kind. Furthermore, Cyclone requires additional routing overhead per node and employs a complex routing scheme. We use a replica placement scheme that automatically produces disjoint routes without significant modification to the underlying routing mechanism.

The notion of independent paths has been considered in other works as well. In an effort to provide message confidentiality, [14] suggests that messages be split, encrypted, and sent to the destination along independent paths. Empirically, [14] determines the finger table offsets to minimize route overlap with high probability. We will show analytically that our placement scheme creates disjoint routes.

Castro et al. [3] propose a secure routing primitive using replication and two routing mechanisms. They combine efficient, locality-based routing with constrained routing mechanisms to find diverse routes to the replica set in the event of routing failure. The key mechanism in creating route diversity is the notion of *neighbor set anycast*. Rather than modifying the underlying routing mechanism, we improve routing robustness by carefully placing replicas. Using the robustness properties we prove herein for our replica placement, we will show that it can be combined with a neighbor set routing approach similar to Castro's to achieve even greater security.

Mickens and Noble [12] develop a framework for diagnosing broken overlay routes, whether they result from

IP-level link failures or malicious routers at the overlay-level. Once the source of the error is detected, the DHT can circumnavigate broken IP-level links or exclude misbehaving nodes to improve the routing robustness of the system. Rather than excluding misbehaving nodes (which may have been falsely diagnosed), our replica placement builds robustness by providing disjoint routes, which can be used to avoid faulty nodes.

3.2. Replica Placement

Replica placement has long been studied in realms outside of peer-to-peer systems. As more work has been done within peer-to-peer systems, it has become clear that replica placement can be used to manage load and satisfy capacity constraints while improving the quality of service in the system. Many studies have compared the performance of several placement schemes in terms of quality of service, availability, and time to recovery [4, 7, 11, 13]. However, to our knowledge, none have considered routing robustness as it relates to replica placement.

Ghodsi et al. [9] present the *symmetric* replica placement scheme, which is equivalent to *equally-spaced* replica placement. They present the benefits that can be achieved in terms of message overhead in node joins and leaves, load balance, and fault tolerance by using this placement. Although such a placement can improve routing robustness, as shown in [10], it is not efficient in the creation of disjoint routes for all DHTs. Our placement efficiently creates disjoint routes while preserving *adaptivity*, or the ability to change the replication degree with minimal overhead.

4. Route Diversity Techniques

In this section, we consider two methods for creating diverse routes in a DHT. First, we will present a replica placement such that the routes to a replica set form a set of disjoint routes. Second, we will describe the route diversifying technique employed by [3] and define *neighbor set routing*, a mechanism that can be used to increase route diversity.

4.1. MaxDisjoint Replica Placement

In the following discussion, we assume that node routing tables are organized as in Pastry [16] and that each entry may be optimized for locality awareness or constrained as suggested by [3]. Furthermore, we assume that routing is performed in an identifier space of size N and that the prefix-matching occurs in digits of base B, where $B = 2^b$ for b bits per digit.

We define $shl(i, j)$ to be the length (in base-B digits) of the shared prefix between the identifiers i and j as in [16]. All of our analytical results are proven within the context of a *full* DHT, which we define to be a DHT wherein every identifier in the identifier space is represented by a node.

The simplest method for generating disjoint routes is to ensure that each route uses a different routing table entry as its first hop. The following lemma embodies the premise of our approach and will be used to prove that our replica placement can be used to produce any desired number of disjoint routes.

Lemma. *In a full distributed hash table that uses prefix-matching routing, routes originating at the same source node that differ in the first hop are disjoint.*

Proof. Consider a query node q and two routes originating at q destined for keys k_1 and k_2. Suppose that the first hop of the routes are n_1 and n_2 ($n_1 \neq n_2$), respectively. The identifiers of nodes n_1 and n_2 differ in the $(i+1)$-th most significant digit, where $i = \min\{shl(q, k_1), shl(q, k_2)\}$. Furthermore, the subsequent hops in each route will share at least the first $(i+1)$ digits with the first hop. Therefore, the two routes cannot share any common nodes and are disjoint. □

Next, we present our replica placement algorithm and a theorem, which prescribes the replication degree necessary to produce any desired number of disjoint routes.

MaxDisjoint Algorithm. *The replica placement is performed in $m+1$ rounds, where $m = \lfloor \frac{d-1}{B-1} \rfloor$. The first m rounds each consist of $B-1$ steps and the final round consists of n steps, where $n = (d-1) \bmod (B-1)$. In the i-th round, B^{i-1} replicas are placed at equally-spaced locations over the entire identifier space at each step (spacing $s_i = \frac{N}{B^{i-1}}$). In step j of round i, the replica locations are given by:*

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

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

Theorem. *To produce d disjoint routes from any query node to a key k in a full distributed hash table using prefix-matching routing with base $B > 1$, 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. (By induction) When $d = 2$, the replica placement is performed in one step wherein a single replica is placed in addition to the master key location. Therefore, the key is replicated at k and $k + \frac{N}{B} \bmod N$. These two replicas differ in the first digit. Therefore, any query node would route to these replicas using two different routing table entries, which produces two disjoint routes.

Assume that the replica placement correctly creates d disjoint routes. We will show that performing an additional step in the algorithm produces one additional disjoint route. Let this additional step be step j in round i. Consider a query node q and select the replica $r \in R_{i,j}$ such that $shl(r,q) \geq (i-1)$. Such a replica exists and is unique because the replicas in $R_{i,j}$ take on all B^{i-1} possible combinations in the first $(i-1)$ digits and share the remaining digits. Let $i^* = shl(r,q)$. The query node q will create an additional disjoint route to r by using the appropriate entry in the i^*-th row of its routing table (the column corresponds to the value of r in the (i^*+1)-th digit). No other replica can use this entry because r differs from all other replicas (outside its own step) in the i-th digit.

The number of replicas necessary to create d disjoint routes can be computed by summing the number of replicas used in each step. The master key accounts for one route and the remaining $d-1$ are created through the replica placement. The number of full rounds is determined by the number of times $B-1$ divides $d-1$ ($m = \lfloor \frac{d-1}{B-1} \rfloor$) and the remaining routes are created in the final (partial) round ($n = (d-1) \bmod (B-1)$). Since, B^{i-1} replicas are placed at each step in round i, the total number of replicas is given by:

$$1 + \left[\sum_{i=1}^{m} (B-1)B^{i-1} \right] + nB^m = (n+1)B^m$$

□

Since each step creates an additional route by using a single routing table entry and the number of disjoint routes is bounded by the outgoing node degree, it is straightforward to show that this placement creates the maximum number of disjoint routes for the given replication degree. For this reason, we give our placement the name MaxDisjoint.

It is worth noting that disjoint routes are created without modifying the underlying routing mechanism. MaxDisjoint naturally creates disjoint routes using the prefix-matching property of the routing scheme. It can be shown rather easily that this result is consistent with the equally-spaced replica placement for Chord prescribed in [10].

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.

Note that when $n = 0$, MaxDisjoint is equivalent to equal-spacing. However, when $B > 2$, equal-spacing is not a low cost, *adaptive* replica placement. An adaptive solution allows for the replication degree to be changed with minimal cost. In order for equal-spacing to be adaptive without shifting replicas, the replication degree must be increased by a factor of two. Each time the number of replicas is doubled, a single replica is introduced between each pair of existing replicas, splitting the existing spacing in half. Thus, an adaptive equally-spaced solution must have a replication degree that is a power of two. The MaxDisjoint placement is a low cost alternative. A comparison of the number of replicas needed to produce a desired number of disjoint routes for MaxDisjoint and an adaptive equally-spaced placement is depicted in Figure 2. In most cases, an adaptive equal-spacing uses excessive replicas to achieve the same number of disjoint routes as our placement.

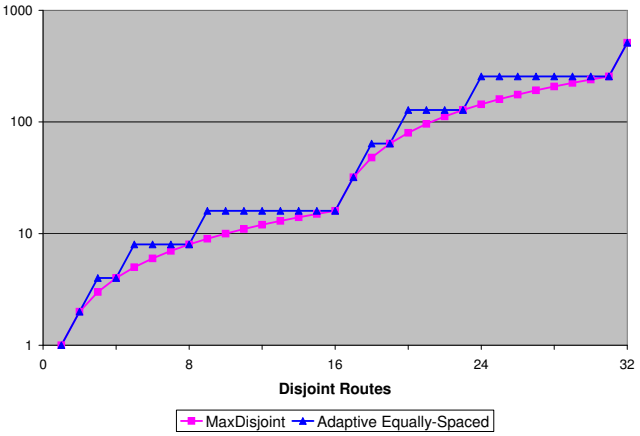

Figure 2. Required Replication Degree for Increasing Numbers of Disjoint Routes.

4.2. Neighbor Set Routing

Castro et al. [3] construct a *neighbor set anycast* to route to a clustered replica set. The premise is to route to nodes that contain the master key in their neighbor set. Using knowledge of the neighbor sets of those nodes, the correct nodes that are home to the replica set can be determined.

In order to reach the nodes containing the master key in their neighbor set, messages must be routed along diverse routes. To create diverse routes, messages are routed using the neighbor set of the query node. We call this *neighbor set routing*. Neighbor set routing sufficiently creates diverse routes when the replicas are distributed uniformly over the identifier space, as in CAN [15] and Tapestry [22]. However, when replicas are clustered, most routes will likely converge at the nodes preceding the replica set in the identifier space. In our experiments, we consider how well route diversity is created when replicas are clustered and measure the impact of combining it with our replica placement.

5. Experiments

Three experiments were run to evaluate our replica placement. First, we evaluated how well the placement improves routing robustness compared to other replica placements, even in DHTs that are not fully populated. Second, we measured the degree to which neighbor set routing improves routing robustness over replica placement alone. Finally, we considered the impact of our replica placement on response time and system load.

All experiments were performed using a Java-based simulator we developed. For the first two experiments, it is sufficient to use a simple route computer to compute the routes between any two nodes in a DHT. To accurately measure response time and, more importantly, the impact of queuing on response time, it was necessary to build a discrete event simulator over the route computer.

For all experiments, 1024 nodes were modeled in a Pastry DHT with a 20-bit identifier space and $b = 4$ (hexadecimal digits are used in prefix-matching). For the first two experiments, each data point in the results is the average of over 100,000 lookups. Because of the extended runtime for discrete event simulations, each data point in the response time experiments is the average of over 10,000 lookups.

5.1. Replica Placement

Experiments were performed to compare several replica placements in terms of the number of disjoint routes created and the impact on routing robustness. In addition to our replica placement, we also considered random placement, where replica locations are randomly selected from a uniform distribution; neighbor set placement, where replicas are placed within the neighborhood (leaf set in Pastry terms) of the master key; and spaced placement, where replicas are separated by a fixed spacing. For the spaced placement, we used spacings of 1024 and 8192.

The average number of disjoint routes created for each replica placement is depicted in Figure 3. For the parameters tested, our replica placement outperforms the other placements in creating disjoint routes. The random placement performs second best, but our replica placement produces an average of 10-15% more disjoint routes than a random placement and nearly 50% more for 16 replicas. Furthermore, in the worst case, a random placement could produce as few disjoint routes as neighbor set placement. Our placement creates a provable number of disjoint routes for every possible query.

The spaced replica placements perform quite poorly compared to our placement because the spacing should vary with the number of replicas to effectively create additional disjoint routes. Note that 1024-spacing creates about as many disjoint routes as neighbor set placement. This is because the average inter-node spacing is 1024.

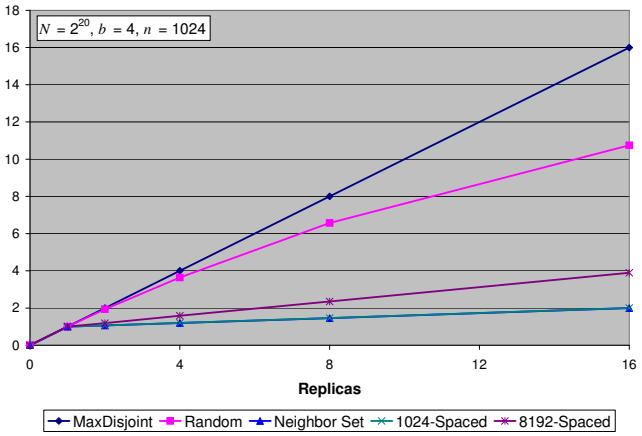

Figure 3. Number of Disjoint Routes with Increasing Replication Degree.

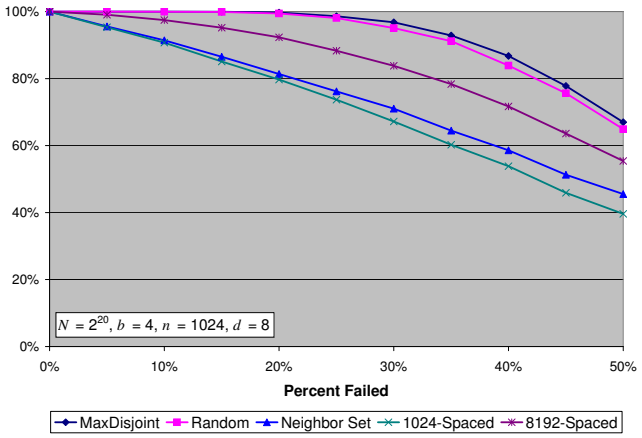

Figure 4. Probability of Routing Success with Increasing Failure Rate.

To measure the impact on routing robustness, we randomly selected nodes within the system to fail. We assume that a route that contains only correct nodes returns a correct result. For each query, the query node computes routes to the entire replica set. If there exists a route that returns a correct result, the lookup is deemed successful.

The impact of the placement of eight replicas on routing robustness can be seen in Figure 4. Our replica placement routes messages with the highest success rate over the range of failure rates tested. Even with a quarter of nodes failed, 99% of lookups are successful This is a dramatic improvement over the neighbor set placement, which only routes 75% of all messages succcessfully with a quarter of nodes failed. The random placement performs comparably to our placement, but introduces bias toward particular queries that

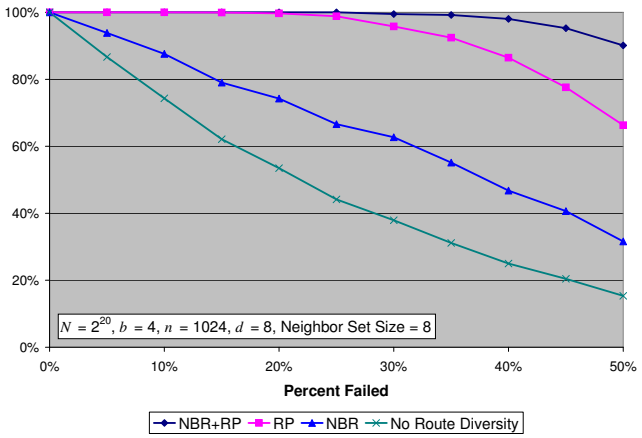

Figure 5. Probability of Routing Success of Replica Placement (RP) and Neighbor Set Routing (NBR) with Increasing Failure Rate.

could be exploited by an adversary.

The spaced and neighbor set placement do not sufficiently create route diversity to have a significant impact on routing robustness. The 1024-spaced placement performs worse than the neighbor set placement because spaced replicas may actually reside on the same node. Replicas placed in a neighbor set reside on distinct nodes, which creates more diversity over spaced placements with small spacings relative to the inter-node spacing.

5.2. Combining Route Diversity Techniques

To evaluate the relative impact of replica placement and neighbor set routing on routing robustness, we conducted a set of experiments to measure the routing success rate. We measured the success rate with neither route diversity mechanism, with each mechanism individually, and with both mechanisms together. When both mechanisms are used together, the query node routes to each replica using its neighbor set nodes as the first hop. The results for eight replicas and a neighbor set size of eight are shown in Figure 5.

Both replica placement and neighbor set routing can make a significant improvement in routing robustness over no route diversity at all. However, replica placement has a stronger impact on the routing success rate. This is because neighbor set routing cannot create disjoint routes. Instead, it attempts to create diverse routes, which at best share no nodes other than the destination node.

Nonetheless, there is benefit in using neighbor set routing in conjunction with replica placement. Especially at higher failure rates, neighbor set routing can introduce additional diversity that can increase routing robustness. With half of the nodes in the system compromised, using neighbor set routing and replica placement together can route 90% of all lookups successfully compared to only a 66% success rate with replica placement alone.

5.3. Response Time and System Load

Finally, since replica placement seems to be a reasonable method for improving routing robustness, it is natural to consider some of the practical concerns with using replication. When querying a replica set, response time can be reduced by querying the entire replica set in parallel. However, this may have a significant impact on the system load. Therefore, we consider three replica query strategies: parallel, sequential and hybrid.

Contrary to the parallel strategy, the sequential strategy queries replicas one at a time waiting for a response before querying the next replica. This strategy controls system load at the expense of response time.

We also considered 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 tuned using the set size to achieve the desired response time with a reasonable system load.

To present realistic response times, we modeled the inter-node delay with a log-normal distribution with mean 60ms and standard deviation 50ms and total response time as the sum of inter-node delays along a route. The log-normal distribution parameters were selected using results from a study of TCP connection round trip times [1].

We extended 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 would be necessary to use a timeout for the sequential and hybrid schemes. Correctly tuning these timeouts is beyond the scope of this paper.

The average response time and system load for eight replicas are shown in Figure 6. For the hybrid strategy, set sizes of two and four (marked “Hybrid-2” and “Hybrid-4”, respectively) were used. These experiments used a relatively low lookup rate of 100 lookups per second, which results in little or no queuing. Therefore, these results are representative of near ideal conditions.

In the presence of ideal conditions, the trade-off between response time and system load is clear. The parallel strategy provides less variability in response time. With increasing failure rate, the response time of the sequential strategy increases rapidly while the parallel strategy slowly increases. However, this comes at the expense of system resources. Using a hybrid strategy can exploit the trade-off to have the reduced response time variability of parallelization while

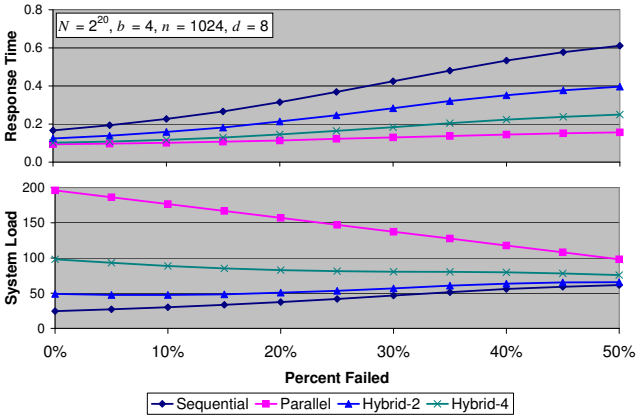

Figure 6. Average Response Time (in Seconds) and System Load (in Messages) for Successful Lookups.

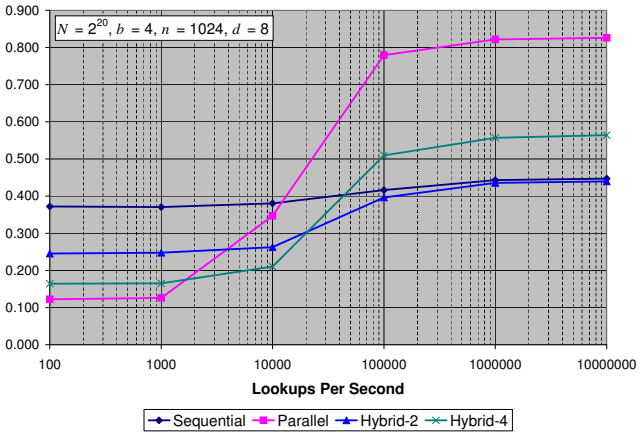

Figure 7. Average Response Time for Successful Lookups (in Seconds) with Increasing Lookup Rate (25% of Nodes Failed).

reasonably controlling the system load.

To measure the effect of message queuing on response time, the above experiment was repeated 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 time in a system with 25% of nodes failed is shown in Figure 7.

For applications with relatively low lookup rates (less than 1000 lookups per second), response times close to ideal can be expected. However, as the lookup rate increases, the response time of the parallel strategy degrades rapidly to nearly double the response time of the sequential strategy. This effect can be compounded with a larger replication degree. If the lookup rate of the application is known a priori, then the appropriate degree of parallelization can be chosen to achieve minimal response time without causing excessive strain on the system. When the lookup rate cannot be known a priori, the use of a hybrid strategy can help manage the trade-off. A potential area for future work is a distributed protocol for congestion control that can detect congestion in the system and tune the degree of parallelization appropriately to minimize response time.

6. Discussion

We have described a replica placement scheme that creates disjoint routes, which can be applied to any DHT that uses prefix-matching routing. Furthermore, we have shown through simulation that such a scheme can improve routing robustness. This solution requires no modification to the underlying routing scheme and produces desirable results, even for sparsely populated DHTs. Our experiments have shown that using our replica placement can be used to correctly respond to all queries with high probability even with a quarter of the system nodes failed.

Furthermore, we considered neighbor set routing as another route diversity mechanism and showed that it can be used to improve on replica placement in terms of routing robustness. With half of the nodes failed, using neighbor set routing with replica placement can improve the routing success rate substantially over replica placement alone. Therefore, using two route diversity mechanisms, like replica placement and neighbor set routing, can have a positive impact on routing robustness. The development of new route diversity mechanisms that can be used in concert to improve routing robustness is a topic for future work.

Finally, we considered some practical ramifications of using replica placement in a real implementation; that is, we evaluated the impact of the replica query strategy on response time and system load. In particular, we considered three query strategies: parallel, sequential and hybrid. As expected, there is a trade-off between response time and system load with the strategy used. The parallel strategy gives a low variability, fast response time at the expense of high system load. To the contrary, the sequential strategy reduces system load at the cost of response time. The hybrid strategy allows the application to tune the response time and system load to tolerable levels. The consideration of alternative query strategies is a potential area for research.

Further experiments were run to measure how quickly the added system load of the parallel strategy degrades the

benefits in response time. For lookup rates less than 1000 lookups per second, the parallel strategy was able to maintain a response time better than the other strategies considered. However, as the lookup rate increases, the response time quickly degrades and the parallel strategy is outperformed by the sequential strategy. A hybrid strategy can be used to control the rate at which the response time degrades.

Congestion resulting from the replica query strategy brings about several potential areas for future work. First, the development of a distributed congestion control protocol that detects when the query strategy is straining the system and changes the strategy to reduce the strain would be of great benefit to applications that have volatile lookup rates. For example, in a peer-to-peer voice over IP application, caller lookup rates may vary wildly with the time of day.

Second, we have studied congestion at the overlay layer. A more involved study would be concerned with the underlying physical topology and how congestion occurs there as a result of the query strategy. This may lead to strategies that vary the order in which replicas are queried such that replicas that are likely to return correct responses quickly are queried first. This approach should result in better response times and reduced system load.

Lastly, our replica placement assumes that data in the system is self-verifying. Since it is not always feasible to assume self-verifying data, we would like to eliminate this assumption. Byzantine fault tolerant algorithms exist that can be used to elect a correct value from a set of candidates. However, we cannot apply these algorithms blindly since the set of values returned do not necessarily contribute equally to the result. Two replicas may have come from routes that share a common malicious node. However, if we find the correlation between each replica and the route along which it was returned, we can assign appropriate weights and construct a working agreement protocol.

References

[1] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay. Variability in tcp round-trip times. In *Proceedings of ACM SIGCOMM IMC'03*, pages 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*, 9(6):50–57, 2005.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of OSDI '02*, pages 299–314, 2002.

[4] Y. Chen, R. H. Katz, and J. Kubiatowicz. Dynamic replica placement for scalable content delivery. In *Proceedings of IPTPS'02*, pages 306–318, 2002.

[5] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of ACM SOSP'01*, pages 202–215, 2001.

[6] J. R. Douceur. The sybil attack. In *Proceedings of IPTPS'02*, pages 251–260, 2002.

[7] J. R. Douceur and R. P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proceedings of MASCOTS'01*, pages 311–9, 2001.

[8] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. *SIGMETRICS Perform. Eval. Rev.*, 33(1):38–49, 2005.

[9] A. Ghodsi, L. O. Alima, and S. Haridi. Symmetric replication for structured peer-to-peer systems. In *Proceedings of DBISP2P'05*, pages 74–85, 2005.

[10] C. Harvesf and D. M. Blough. The effect of replica placement on routing robustness in distributed hash tables. In *Proceedings of P2P'06*, pages 57–6, 2006.

[11] Q. Lian, W. Chen, and Z. Zhang. On the impact of replica placement to the reliability of distributed block storage systems. In *Proceedings of ICDCS'05*, pages 187–196, 2005.

[12] J. W. Mickens and B. D. Noble. Concilium: Collaborative diagnosis of broken overlay routes. In *Proceedings of DSN'07 (to appear)*, 2007.

[13] G. On, J. Schmitt, and R. Steinmetz. The effectiveness of realistic replication strategies on quality of availability for peer-to-peer systems. In *Proceedings of P2P'03*, pages 57–64, 2003.

[14] M. Portmann, S. Ardon, and A. Seneviratne. Mitigating routing misbehaviour of rational nodes in chord. In *Proceedings of SAINT'04*, pages 541–545, 2004.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM '01*, pages 161–172, 2001.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of ACM Middleware'01*, pages 329–350, 2001.

[17] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *Proceedings of ACM SIGOPS'04*, pages 115–120, 2004.

[18] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of IPTPS'02*, pages 261–9, 2002.

[19] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured peer-to-peer systems: A quantative analysis. In *Proceedings of IEEE ACSAC'04*, pages 252–261, 2004.

[20] A. Stavrou, A. Keromytis, and D. Rubenstein. Exploiting structure in DHT overlays for DoS protection. Technical report, Columbia University, 2004.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM'01*, pages 149–160, 2001.

[22] 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 Journal on Selected Areas in Communications*, 22(1):41–53, 2004.