A Comparative Study of the DNS Design with DHT-Based Alternatives

Vasileios Pappas
Computer Science Department
UCLA
Email: vpappas@cs.ucla.edu

Dan Massey
Computer Science Department
Colorado State University
Email: massey@cs.colostate.edu

Andreas Terzis
Computer Science Department
Johns Hopkins University
Email: terzis@cs.jhu.edu

Lixia Zhang
Computer Science Department
UCLA
Email: lixia@cs.ucla.edu

***Abstract*— The current Domain Name System (DNS) follows a hierarchical tree structure. Several recent efforts proposed to re-implement DNS as a peer-to-peer network with a flat structure that uses Distributed Hash Tables (DHT) to improve the system availability. In this paper we compare the performance and availability of these two designs, enabled by caching and redundancy in both cases. We show that the caching and redundancy mechanisms in each design are closely bound to its *system structure*. We further demonstrate that each of the two system structures provides unique advantages over the other, while each has its own shortcomings. Using analysis and trace-driven simulations, we show that hierarchical structure enables high performance caching and that DHT structures provide high degree of robustness against targeted attacks. We further show that the current DNS design offers engineering flexibilities which have been utilized to optimize system performance under typical Internet failures and traffic loads, and which can be further extended to overcome DNS weaknesses against the aforementioned attacks.**

I. Introduction

The Domain Name System (DNS) [11] is a large-scale, hierarchical, distributed database spanning over a large number of administrative domains, providing indispensable naming services to the Internet. Its lean and extensible design has enabled it to meet unforeseen demands twenty years after its initial deployment. Despite all the successes, however, a number of operational issues have risen recently including configuration errors [13] and denial of service (DoS) attacks [1]. These problems have motivated proposals (*e.g.* [3], [15]) to re-implement DNS as a peer-to-peer network using Distributed Hash Tables (DHTs) [16], [17]. The rationale is that the ability of DHTs to self configure can eliminate misconfigurations, while their flat structure can withstand denial of service attacks.

Our goal in this paper is to compare the current DNS to DHT-based designs. We choose service availability and performance as the comparison metrics because all Internet applications desire short lookup times and high availability from DNS. We ground our comparison by using Chord [17] as a representative of DHT designs in our comparative study. However we claim that the results are general enough to be applicable to other DHT-based designs.

Both the existing DNS and proposed DHT-based alternatives employ the same two general mechanisms to improve their availability and performance: redundancy and caching. The similarities however end there; the specific realizations of these mechanisms are entirely different. Our thesis is that these realizations are closely bound to the structure of each system, that is they are applied to complement the distinctive properties of the hierarchical and flat structures. The end result is that the same mechanisms result in different behaviors, reflective of each system. This thesis is validated through analysis and trace-driven simulations, in which we show that DNS and DHTs exhibit distinctive behaviors that lead to respective advantages and disadvantages.

More specifically, we show that the current DNS structure with an average node degree of 2.57, is as resilient to random node failures as a Chord structure with an average node degree of 28 and roughly the same path length distribution as DNS. Conversely, Chord is proven to be more resilient to orchestrated attacks. Its failure rate is almost the same as in the case of random failures, whereas in DNS the failure rate under attacks is considerably higher when compared to random failures. Both observations are attributed to the fact that the deployed DNS is an engineered system in which higher level nodes are more important and thus wider replicated, whereas in DHTs with a flat structure all nodes have the exact same role and weight.

We also show that the performance of the two systems, measured as the number of application layer hops, has completely different properties. In contrast to previous studies [3], that attributed the poor performance of DHT-based DNS look up systems to long path lengths, our results show that that the dominant factor is cache effectiveness, rather than path length. For instance, there are cases where the performance of passive caching in Chord can deteriorate when the average path lengths decrease. Only when the *global* popularity of a record increases then its cache hit rate improves. In contrast, DNS cache performance is a function only of the *local* query distribution, i.e the traffic generated locally at each caching server, and benefits highly from the ability to cache the DNS structure.

In summary, our comparative study offers the following results: A DHT-based name look up system outperforms DNS only in terms of its resilience to orchestrated attacks. Under random node failures, the DHT-based system can provide availability comparable to the current DNS only when its node connectivity degree is high (one order of magnitude higher than DNS), which consequently leads to higher maintenance overhead. Moreover, our results show that DNS cache per-

formance outperforms passive caching in DHTs. Furthermore, achieving comparable cache performance in a DHT requires additional mechanisms, such as the proactive caching proposed in [15], which however generate additional overhead.

II. Impact of Structure on Path Redundancy and Caching

We compare DNS and DHTs using two metrics: performance and availability. We measure *performance* by counting the number of application-level servers visited while answering a query [1]. We chose this metric because it captures the key design features of the two systems. Although other factors, such as the response time of individual servers, also impact the performance of a name look up system, those factors can be addressed without any design changes. On the other hand, the number of hops is an intrinsic function of each system. We measure *availability* based on the system's static resiliency [5], *i.e.* the ability to resolve queries in the presence of failures while no recovery mechanism is active. While such recovery mechanisms will significantly improve a system's availability, our goal is to capture the fundamental properties of a system's structure.

While both DNS and DHTs rely on redundancy and caching to improve performance and availability, they implement the two mechanisms differently due to the underlying structural differences. DNS is a hierarchical tree, while DHTs use a flat structure. In the remainder of this section we provide a brief summary of each system and compare the implementation and effectiveness of redundancy and caching in the two systems.

a) DNS Background: Scalable management of the Internet's large and continuously expanding namespace is achieved via a distributed hierarchy in which separate administrative entities manage different portions of the DNS tree. The basic DNS management unit is called a *zone*, which covers a continuous subtree in the DNS namespace. Each zone can delegate parts of its namespace to *children* zones; there are no limitations on either the number of a zone's *delegations* or the depth of the tree. The hierarchical Internet namespace maps to a tree of zones, with the *root zone* at the top. Each zone provides a set of redundant servers, each of which can provide *authoritative* answers to all the data stored in the zone. Thus a zone's data is available as long as any of the *authoritative servers* is available. DNS uses a generic data structure, called a *resource record* (RR), to store all the naming information. The *Name Server* (NS) RR provides the authoritative servers for a zone, and it is stored both at the parent and the child zone. DNS queries are generated by *stub resolvers* implemented in all the hosts. A stub resolver sends DNS queries to a local *caching resolver* which performs iterative look-ups to generate the reply that is eventually returned to the end client.

b) Chord Background: Chord nodes lie on a one-dimensional cyclic identifier space $[0, \ldots, 2^m]$ on which the distance between identifiers A and B is calculated as the clockwise numeric distance from A to B on the circle. A node with identifier (ID) x in a Chord ring of N nodes, maintains pointers, i.e. *fingers*, to $logN$ neighbors where the i^{th} neighbor is the node closest to $x + 2^i$ on the circle. If the base of the ring is extended from 2 to b, the number of neighbors per node increases to $(b-1)log_bN$. Moreover, each node maintains an additional table with sequential neighbors (i.e. its successor nodes) to improve the resilience of the ring structure. A Chord-based DNS system maps DNS resource records (RRs) to nodes by first hashing the name of each record to a value in $[0, \ldots, 2^m]$, the key for the record, and then assigning the RR to the node v with the next larger ID. To provide high availability in the presence of node failures, the same RR is replicated across a fixed number of nodes which succeed node v in the ring. Note that in the Chord-based DNS each node serves both as an authoritative server and as a caching resolver, thus we refer to them as *servers* or simply *nodes* in the rest of the paper.

A. Redundancy

While a cursory inspection of the deployed DNS system might suggest that only a single look-up path exists between the root of the tree and the destination domain, this is not the case in reality. Each "node", i.e. zone, in the DNS tree is served by multiple servers. Because each of the redundant servers provides exactly the same pointers towards the destination, a resolver may use any of the servers at each level, that is its look-up path can be any of $P = \prod_i R_i$ total number of paths, where R_i is the the number of servers at level i.

In Chord, as well as in other DHTs, path redundancy is achieved by providing redundant connectivity between DHT nodes. Each node has a set of neighboring nodes, and a subset of one's neighbors leads towards each destination. A query is forwarded to one neighbor among the subset, preferably to the one that is closest to the destination. It has been shown that Chord can provide $(logN)!$ paths between any two nodes, where N is the number of nodes in the system [5], whereas other DHT designs provide limited or no choice of next hop nodes.

The design choices made by DNS and Chord lead to two important differences. First, a DNS query will reach the destination as long as any working path exists (e.g. at least one server of each zone in the path is reachable). In Chord, a node may fail to reach a destination even when a valid path exists. For example, a node may have two neighbors which can help forward a query, with the first one leading to a "dead-end" after a few hops, while the second leads to a viable path to the destination. The query will fail if it is forwarded to the first neighbor even though a working path exists. We will show later in the paper how this query forwarding scheme affects the availability of DHTs. Broadly speaking, DNS utilizes all the existing redundancy, while DHTs do not fully explore the underlying redundancy.

The second important difference stems from the fact that in DNS, all queries for the same zone follow the same logical path, i.e. the same sequence of zones, and vary only in the choice of server at each zone; in particular, all queries start from the root zone, making it of critical importance to the

[1] In the rest of this paper we use "number of servers" and "number of hops" interchangeably. Note that the latter means application-level server hops, and not IP-level router hops.

availability of the entire system. In contrast, DHT queries for the same destination that originate from different nodes follow different paths, which tend to merge only when they approach the destination. Consequently, servers for the root and TLD zones are more important than others in DNS, while nodes in a DHT system are more or less equally important. This equality among nodes is both an advantage and a liability. One can easily improve the overall availability of DNS by increasing the number of redundant servers serving important zones. In fact, the root zone, important TLD zones, as well as most popular domains, have been engineered with highly redundant servers; the level of server redundancy tends to decrease for zones at lower levels in the DNS hierarchy. However it is impossible to apply similar engineering adjustments to DHT-based systems. At the same time, the top level zones in DNS are an obvious target for malicious attacks, while it is not immediately clear how to effectively attack a DHT system.

B. Caching

Caching greatly improves the performance and availability of DNS. In addition to caching user query replies, a unique feature of caching in DNS is that resolvers also cache the information about the servers (NS RRs) for all the zones visited during a query resolution. For example, to answer a query for *www.cs.foo.edu*, a resolver obtains the server information for 3 zones: *edu.*, *foo.edu.*, and *cs.foo.edu.*. The resolver can handle a subsequent query for *ftp.cs.foo.edu* by directly contacting one of *cs.foo.edu.* zone servers if the zone's NS RRs are still in the cache; even a query for *www.ee.foo.edu* can start with one of *foo.edu.* servers. Because DNS queries follow a top-down search, in general resolvers have the root and popular TLD server NS RRs cached locally almost all the time.

In a DHT-based name lookup system, individual nodes can also cache data obtained from query replies. However because each name is hashed to a unique key in a *flat* space, and each node has a different next hop in forwarding queries, there is no concept of NS records or common servers that can be cached. When a node issues a query, the query must traverse the whole path to reach the destination. A DHT-based system uses recursive queries and allows en-route caching of records. After a record has been resolved, all the intermediate nodes, that forward the record back to the querying node, can store a local copy. Thus, subsequent queries for the same name that cross any of the nodes with cached copies can be answered immediately. As a result, the number of hops needed to resolve a query is decreased. This type of caching also improves availability. If the query crosses a node with a cached reply, the record is retrieved even if some subsequent servers along the path (or the destination itself) are unavailable.

Since both DNS and DHTs cache replies until the TTL expires, the probability that a record can be found in a node's local cache should be roughly the same. However the behavior, and hence the performance, of the two systems are rather different when a cache miss occurs. In DHTs, the node sends the query towards the destination, which either reaches the destination or reaches an intermediate node along the query path which has cached the record. The probability of encountering a cached record at intermediate nodes is a function of the record's *global* popularity and other system parameters that determine how likely and where the different paths to the destination may meet. Conversely, a DNS resolver uses its cached NS records to expedite queries in the event of a cache miss. Following the example used in the previous paragraph, even when a query for *mail.cs.foo.edu* does not find the corresponding record in the local cache, the NS records for *cs.foo.edu* may be present. Or if the NS records for *cs.foo.edu* are not present, the NS records for *foo.edu* may be present. As a result, the effectiveness of DNS caching depends only on the popularity distribution of *local* queries to a resolver, and is independent of the queries generated by other nodes.

Caching also helps improve the availability of both systems. In case of a cache hit, data can be retrieved even if the path to the authoritative servers or the authoritative server itself are not available. In case of a cache miss, caching in DNS and DHTs enables different availability "modes". Caching in DNS improves *path availability*. For example, if all of the root zone servers become unavailable and thus the query path is partitioned, a resolver can still access different subtrees of the namespace, especially those of local interest, by using their NS records in the cache; the resolver fails only if all the authoritative servers of a destination zone are unavailable (*data availability*). Caching in DHTs does not shorten the query path, although it is possible to retrieve a record if it happens to be cached at intermediate nodes on the path, even when all the destination servers failed.

III. Methodology

We use a combination of analysis and simulation to compare the two systems based on the following metrics:

- *Data Failure rate*: The percentage of queries that fail because all replicas that store the queried record are not available.
- *Path Failure rate*: The percentage of queries that fail to find a path between the querying node and any of the replica destination nodes, when at least one replica node is available.
- *Path Lengths*: The number of server hops needed to resolve a query. Unresolved queries are not counted in this case.

A. DNS Traces

We have collected a set of DNS packet traces. We use these traces both directly for a measurement analysis, and indirectly by feeding them to a trace-driven simulator. Our DNS traces were collected by capturing DNS traffic between three local caching resolvers in a university campus and the Internet. Thus, the traces include all transactions between the resolvers and the DNS servers and exclude any exchanges between the local stub-resolvers and the caching resolvers.

Table I provides the key characteristics of these traces. The first three rows correspond to traces captured at three different caching resolvers, while the last one is the sum of the above three. All traces are 12 days long. The table

Name	Start Date	End Date	Total Queries	Total Replies	Successful Questions	Unique Records	All Zones	Zipf-law α parameter
Trace 0	04/27/04	05/09/04	2,152,836	1,916,055	1,160,639	250,388	80,094	0.8622
Trace 1	04/27/04	05/09/04	1,681,524	1,356,120	719,430	218,664	78,612	0.7527
Trace 2	04/27/04	05/09/04	713,217	621,030	250,127	96,606	38,847	0.7095
All Traces	04/27/04	05/09/04	4,547,577	3,893,205	2,130,196	390,416	119,432	0.9365

TABLE I
STATISTICS OF THE COLLECTED DNS TRACES.

shows the total number of requests sent and replies received by the caching servers. Only the queries that resulted in successful responses were used in our measurement analysis and trace-driven simulation. We derive the total number of unique resource records from the successful queries for which we receive at least one reply (either positive or negative).

B. Trace-driven Simulation

We implemented a trace-driven simulator for DNS as well as Chord [17]. In both cases, we feed the simulator with the sequence of queries that appear in each packet trace. In this way, we are able to reproduce a realistic resolver workload and maintain the exact timing of the queries. Furthermore, we are able to preserve the exact mapping between DNS records and TTL values. Although synthetically generated traces can provide the same distribution of queries and TTL as in a real trace, they cannot preserve the mapping between query popularity and corresponding record TTL values.

1) DNS Simulation: The DNS simulation runs as follows: We collect all the zones that appear in each of the traces and build an exact image of the resulting DNS tree in our simulator. For each zone we preserve the number of authoritative servers as shown in the trace data by looking at the authoritative section of each reply. We ignore any potential misconfigurations that might have reduced the actual number of available authoritative servers [13].

After building the DNS tree structure, the simulator issues the exact queries that appear in the trace, preserving both the sequence and the specific timing for all the queries. The simulator runs in two different modes: *I)* Caching-disabled, where every query starts from the root zone and traverses the whole tree hierarchy to reach the destination zone server which replies back with an authoritative answer. *II)* Caching-enabled, where the local resolver can use previously cached records to speed up lookups. We use these two modes to separate the effect of redundancy and caching on the availability of DNS.

Finally, we consider two types of failure: physical failures (i.e. node crashes and uncorrelated network failures), and failures due to orchestrated malicious attacks. Physical failures are simulated by disabling a random set of DNS servers, and malicious attacks are simulated by disabling a selected set of servers. We assume the attacker can knock down the most important nodes of the DNS tree structure (e.g. root and top-level domain servers) to cause maximal damages.

2) DHT Simulation: The DHT simulation works as follows: First, the simulator initializes a Chord ring for a given number of nodes. In our simulation the network size ranges from 1024 to 65536 nodes and the base of the ring is either 2, 4 or 8. After the initialization phase, we assign every resource record from the DNS trace, to the appropriate node in the DHT system, and then replicate the record in the neighboring nodes, based on the degree of replication. We experimented with 3, 5 and 7 replica nodes (counting the original one).

Again, the DHT trace-driven simulator runs in two modes: *I)* Caching-disabled, in which all queries are issued by the same node in the DHT system, and query replies are not cached by the intermediate nodes. As in the case of DNS, queries are issued by preserving their sequence and their specific timing. *II)* Caching-enabled, in which intermediate nodes cache the answers for the queries that they forward according to the TTL values appeared in the DNS traces. We use a number of additional clients to populate the intermediate node caches in this mode. The locations of these participating nodes are randomly selected, their queries follow the same popularity distribution as derived from the DNS traces, with the sequence and timing randomly distributed. In this way we avoid synchronizing the queries generated from all the nodes while maintaining the same query distribution. Only one of the nodes, the one used for collecting the results, follows the inter-query timing as appears in the DNS traces.

Similar to DNS simulations, two types of failures are considered here. Physical failures are simulated by failing a set of randomly selected nodes; the set of failed nodes is different for each query. Failures caused by malicious attacks are simulated by failing the nodes occupying a continuous block of the identifiers space.

C. Discussions

Before presenting our results, we highlight some of the design choices made in our simulation and analysis methodology.

1) Recovery Mechanisms: We did not implement any of the recovery mechanisms to repair stale routing table entries in DHTs after node failures, so that we can measure the static resiliency [5] of the DHT system, rather that the availability with all the possible recovery mechanisms utilized. We chose to do so for the following two reasons. First, the degree of static connectivity shows how resilient the *structure* of each system is, and thus it indicates how necessary an adaptive recovery mechanism would be. Second, it allows us to do a direct comparison between the DNS and the DHT systems, because the former does not have an adaptive recovery mechanism. This does not mean that such a mechanism cannot be added to the current DNS when it is considered necessary. For example, DNS anycast could be considered as an adaptive mechanism against failures, though we do not evaluate its impacts on DNS availability, for the same reasons that we do not consider the impacts of recovery mechanisms on DHTs.

2) Node Failure Model: Our simulator uses a simple node failure model which assumes that servers fail randomly. This model is considered adequate for physical nodes failures (such as node crashes or reboots), or even network connectivity failures. However this model does not capture failures due to configuration errors which may lead to correlated failures. For example, measurements showed that a significant percentage of the DNS zones have all their servers placed in the same network [13], hence a single network failure can make all the servers of a zone unavailable. This observation relates only to the DNS system, as DHT-based systems are supposedly free from configuration errors.

In addition, our simulator does not use a realistic distribution model for the servers' down-times. Instead, all failed servers become available again after a short period of time in the order of tens of seconds. Failures and recoveries happen more or less instantaneously, thus queries that are issued closely spaced in time and follow the same query paths may still encounter different set of failed servers. However we believe that both issues have negligible impact on our results and findings, for the following reason: Although a realistic failure model could provide more accurate failure rates for both systems when deployed in a real environment, the main goal of this paper is a comparative evaluation of the two systems under the same setting to identify the *relative* advantages between each other, rather than providing *accurate* measures for their performance and availability in actual deployment.

3) Client Record Popularity: It is well-known that the distribution of DNS query popularity follows Zipf's law [8]. This observation leads to the following question: Do queries generated by different caching resolvers follow the same distribution? In other words, can we assume that the popularity ranking of a given query is the same across different resolvers, and consequently is the same when one aggregates queries from a large set of resolvers?

In [12] we show that record popularity differs considerably across caching resolvers even when they serve the *same* user population. Naturally one would expect resolvers serving different user populations to have even less overlap in their query popularity distribution. In our DHT trace-driven simulations, however, we assumed that all the nodes, i.e.. caching resolvers, follow the same query distribution. Thus, it is likely that our simulation results provide an optimistic estimation of DHTs cache performance. Nevertheless our simplified query distribution model still allows us to identify the upper bound of cache performance in a DHT-based system without proactive caching.

IV. Analytical Model

We start our evaluation by presenting analytical models for system availability as a function of node failure rate and for the effect of caching on the performance of both systems.

A. Availability Analysis

Let's assume a query distribution where Q_i is the percentage of queries that require i application hops to be resolved. In addition, let's assume that at each hop j there are R_j nodes on average that can be used in order to forward a query to the next hop, and that the maximum path length is L. If all nodes fail independently and with a uniform probability P, then the failure rate F_i for a query of length i is:

$$F_i = F_{i-1} + (1 - F_{i-1})P^{R_i}, F_0 = 0 \quad (1)$$

Thus, the average path failure rate for the system is:

$$F = \sum_{i=1}^{L} F_i Q_i \quad (2)$$

Equation 2 shows that the failure rate increases as the path length increases, or when R_i decreases. Figure 1 shows the path failure rate for DNS, by using the same distribution of path lengths and number of redundant servers that appear in our DNS packet traces. The path failure rate for Chord and for a tree-based DHT are also shown on the same graph. For the Chord protocol we don't account for sequential neighbors, given the analytical model cannot capture this system parameter. These analytical results match the simulation results presented later, as well as results from previous studies [5], and they confirm our reasoning about the importance of the higher level nodes in DNS tree: Even if the average node degree in DNS is considerable lower than in the Chord structure, 2.57 and 13 respectively, its overall availability is higher, given that the most important nodes have a higher degree of connectivity than the typical nodes.

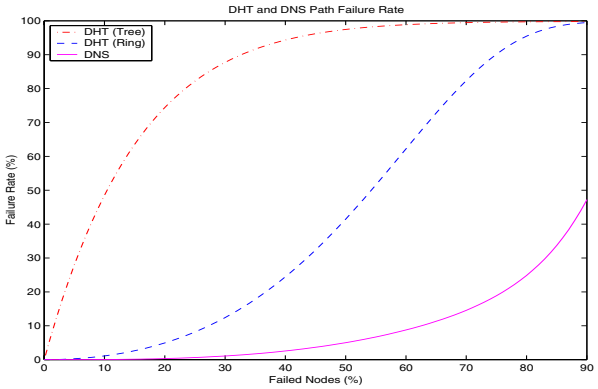

Fig. 1. Analytical results on the path failure rates of DNS, Chord and Tree based DHTs.

B. Cache Performance Analysis

Previous studies have modeled the behavior of DNS caching for queries that are issued from stub-resolvers to a local caching server [7]. In contrast, our goal is to model the behavior of caching in DNS for queries issued by the local caching server to authoritative servers. In other words, we model the effectiveness of caching for queries that have a local cache miss. Similarly, we model the behavior of DHT caches for the same type of queries, since cache hits can also be modeled by the methodology presented in [7].

a) DNS Cache Performance : As we explained in Section II-B, the effectiveness of caching in DNS critically depends on the caching of NS records. These records are

locally stored only when caching servers receive specific types of replies. Next, we classify replies based on their ability to contribute to the caching of NS records for a specific zone:

- *Type I:* Replies for records contained in a zone. These replies contain, apart from the actual answer, the zone's NS records. Thus, whenever a server receives such replies, it can insert or even refresh the zone's NS records in the local cache.
- *Type II:* Replies about non-existing records. These replies cannot directly refresh the zone's NS records, since NS records are not included in the reply.
- *Type III:* Replies that are referrals to a child zone. We count these replies only for the parent zone [2].

Let's assume that the number of replies of Type I, II and III coming from zone i, with in the period of one TTL, is R_{i1}, R_{i2}, and R_{i3} respectively. Note that the TTL corresponds to the TTL of the zone's NS records. If at time 0 the caching server does not have any cached entries, then in the period of one TTL the first query for the zone's NS records will lead to a cache miss (i.e. a question to a higher level zone), followed by $R_{i1} + R_{i2} + R_{i3} - 1$ cache hits (i.e. questions sent directly to the zone) until the TTL expires. Thus the cache hit rate for the NS records of zone i is:

$$P_i = 1 - \frac{1}{R_{i1} + R_{i2} + R_{i3}} \tag{3}$$

If we consider the fact that replies of Type I may refresh the zone's NS records, the cache hit rate should be even higher. Thus, Equation 3 realistically provides a lower bound for the zone's cache hit rate. Given the query distribution for the three types of replies, one can compute the cache hit rate P_i for every zone i that appears in the trace, by using Equation 3.

Thus, if Z_i is the total number of queries sent at zone i then we can compute the number of queries that are answered in j DNS hops as follows: Let N be the total number of zones in the trace, and P_i^j the cache hit rate of the zone that is the ancestor at distance j of zone i (the parent is at distance one and P_i^0=P_i). Then the number of queries H_j that need j hops to be answered is equal to the percentage of queries that have a cache miss for the NS records of zone i and it's $j-1$ ancestors, multiplied by the number of queries zone i receives, summed over all the zones:

$$H_j = \sum_{i=1}^{N} Z_i \big(P_i^j \prod_{k=0}^{j-1} (1 - P_i^k)\big) \tag{4}$$

Equations 3 and 4 show that the cache hit rate of DNS, for queries that experience a local cache-miss is a function of the following two parameters. The query distribution generated by a caching server (Equation 3) and the exact subpart of the DNS tree structure (Equation 4), as seen by the caching server. In Section V-B we use the collected traces to directly measure DNS cache performance, instead of using Equation 4, even if difference between analytical and experimental results is between 3-5%. We chose to look at the experimental results since they provide us with a more details about the real system.

b) DHT Cache Performance : We make the following assumptions to model DHT caches: records are stored only in one node (i.e. no replication) and all queries to all servers follow the same distribution. On the other hand the processes that generate the queries are i.i.d. (independent and identically distributed). While our analysis is based on the Chord protocol, it can be extended to other DHTs, when the path length distribution can be analytically expressed.

Let N be the total number of nodes in the Chord network, and C be the total number of requests for a certain record coming from distinctive clients in the period of the record's TTL. For now, we require that all the records receive the same number of requests per TTL (i.e. they have the same popularity), but we relax that assumption later. Finally, Q_i is the query distribution, i.e. the number of requests spanning i hops.

It is known [10] that the shortest path lengths in a Chord network with base 2 follow a binomial distribution with $p = 1/2$. This result can be extended to Chord rings with base b. In this case, the probability mass function L_i of the shortest paths again is binomial, but $p = 1 - 1/b$:

$$L_i = \begin{pmatrix} \log_b N \\ i \end{pmatrix} (1 - 1/b)^i (1/b)^{\log_b N - i} \tag{5}$$

where i is the length of the path. The request distribution Q_i follows also a binomial distribution, under the assumption that requests are randomly generated from any node in the network. Therefore, the number of clients C_i of a specific record that are i or more hops away from that record is:

$$C_i = \sum_{j=i}^{\log_b N} C \cdot L_j \tag{6}$$

Given that routing paths from different clients to the same record are different, the probability P_i of two clients having a common node at distance i on the path to the record is:

$$P_i = 1 - (1 - \frac{1}{N \cdot L_i})^{C_i} \tag{7}$$

Thus, two independent paths merge at distance i from the destination record with probability S_i:

$$S_i = P_i \cdot \prod_{j=i+1}^{\log_b N} (1 - P_j) \tag{8}$$

In consequence, the number of cache hits at distance i from the destination record is:

$$H_i = \sum_{j=i}^{\log_b N} S_j \cdot Q_j \tag{9}$$

Equations 5-9 show that the cache hit rate in Chord, is only a function of the size of the network, N, the base b of the the protocol and the total number of clients C. Note that originally

[2]For example, if the NS records of *ucla.edu* are not locally cached, and there is a question about *www.ucla.edu*, then the referral from the parent zone *edu* about the *ucla.edu* zone is considered a type III reply for the *edu* zone.

Number of Hops		0	3	5	7	9	11	13	15
500 Clients	Hit Rate % (Simulation)	2.61	4.67	15.47	17.40	8.62	1.69	0.12	0.00
	Hit Rate % (Analysis)	2.88	6.03	12.50	15.86	10.17	2.06	0.30	0.01
250 Clients	Hit Rate % (Simulation)	1.79	3.49	13.38	18.63	9.64	1.93	0.16	0.00
	Hit Rate % (Analysis)	1.76	4.60	11.55	16.76	11.75	3.54	0.38	0.01
100 Clients	Hit Rate % (Simulation)	0.42	2.77	12.56	19.78	12.01	2.86	0.26	0.00
	Hit Rate % (Analysis)	0.94	3.25	10.23	17.35	13.50	4.37	0.50	0.01

TABLE II
SIMULATION AND ANALYTICAL RESULTS OF CHORD CACHE HIT RATE FOR EACH HOP.

we assumed that all queries have the same popularity, i.e. C is constant. In reality, queries have different popularity, and thus we can compute the hit rate of the DHT systems, by taking the weighed average for the different values of C.

In order to evaluate the accuracy of the analytical model, we run a number of simulation and we compared the cache hit rate provided by the analytical model with the ones provided by simulation. From Table II we can see that the analytical model provides a good approximation to the simulation results, with maximum difference less than 3% between the cache hit-rate computed analytically and with simulation. Given the quality of this approximation and the fact that trace-driven simulations for large DHT networks are very time consuming, we chose to use our model in Section V-B to evaluate DHT cache performance.

V. EVALUATION

A. Availability

In this section we evaluate the availability of the two systems, by measuring the data and path failure rates, under random node failures, and failures due to malicious attacks. First, we investigate the contribution of record replication and of path redundancy in both systems. Then, we explore how caching affects their availability.

1) Data Replication & Path Redundancy: The results presented in this section are based on Trace 1. For the DNS simulations we used a system of around 95,000 servers, the total number of servers in Trace 1. We used a network of 8192 nodes for Chord. Note that we also tried a variable number of servers for both cases, in order to assess the impact of scale, and we found that the number of servers does not affect the static resiliency of the two systems.

Figure 2 presents the failure rate of the current DNS system, for different percentages of failing nodes. In addition, it gives the failure rates for a number of hypothetical replication schemes: for example with 3 replicas we require all the zones to have at least three redundant servers. The graph shows that the path failure rate is lower than the data failure rate. This happens mostly due to the fact that higher level zones have a higher degree of replication than leaf zones. Moreover, we can see that a higher degree of replication leads to a lower failure rate, both for data failures and path failures.

Figure 3 gives the corresponding results for Chord, with and without sequential neighbors [5]. As in the case of DNS, the data failure rate is lower when the degree of replication is higher. Similarly, but to a much smaller extend, the path failure rate decreases when the number of replicas increase. On the other hand, it is interesting to note that the path failure rate is higher than the data failure rate, when sequential neighbors are not included. This means there is higher probability of not being able to route a query to any of the replica nodes compared to the probability of all replicas being unavailable. This can be attributed mainly to the following reasons: the typical path lengths of the above network are longer than the path lengths of a typical DNS query, and that some of the available paths are not fully exploited, something that happens in a lower extend when sequential neighbors are included.

Figure 4 gives the path failure rate of DNS as a function of the path lengths. More specifically this graph is derived in the following way: for the queries that did not experience a data failure, we measured how many of them had a path failure. Then, we aggregated all the queries that would have normally required the same number of hops in order to be resolved, and we computed the average failure rate for each number of hops. The graph shows that the failure rate becomes higher as the path length becomes longer. This result indeed verifies our conjecture that higher level zones are more available, due to the higher degree of replication. In addition, we can see that there are virtually no path failures for the queries that go to the root or the TLD zones. This means that the root zone almost never fails, or in other words that existing root servers are more than enough to provide high availability even if the server failure rate is as high as 90%.

Similarly, Figure 5 verifies our conjecture about the Chord paths. We can see that the failure rate increases very rapidly within the first 3-4 hops, and then it remains almost constant for the rest of them. Given that the path lengths in the specific network follow a binomial distribution, most paths are 6-7 hops, and thus, they experience a high path failure rate. In contrast, in DNS most paths have length of 3-4 hops, which leads to a lower path failure rate.

All the previous results are based on DNS and DHT structures with a fixed number of nodes. We tried a variable number of nodes and surprisingly the failure rates for the two systems are the same. For DNS the explanation is the following: even though the number of servers is different, the probability of hitting a failed server is still the same for the same percentage of failed nodes. For Chord, the explanation is more involved: for a larger network size the path lengths increase and thus the failure rate should increase. Fortunately, the number of neighbors also increases and that increase in capable for compensating the path length increase. The above were verified through the simulation results and the analytical results of Section IV-A.

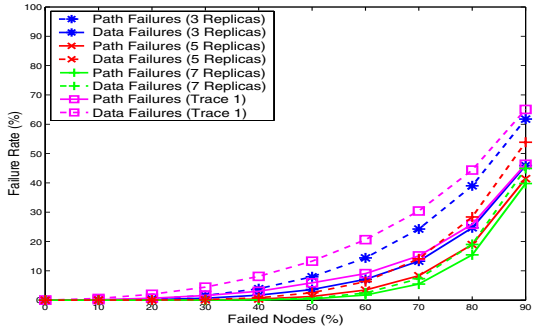

Fig. 2. DNS data and path failure rates

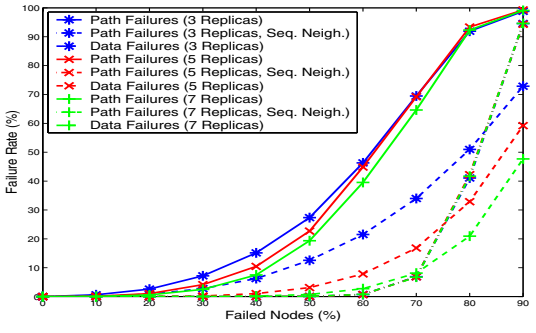

Fig. 3. Chord data and path failure rates

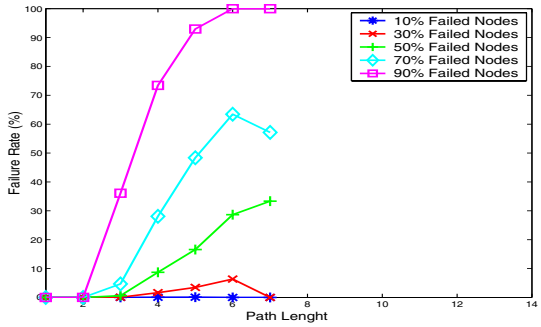

Fig. 4. DNS failure rate and query path lengths

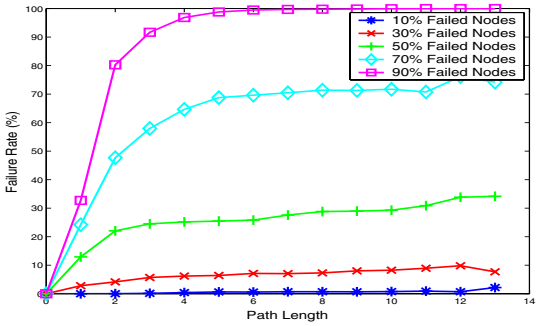

Fig. 5. Chord failure rate and query path lengths

Finally, Figure 6 gives the impact of node failures on the performance of the DNS system. It shows the increase in the path length of successful queries, or more specifically queries that do not encounter a path failure. The increase in the path lengths is measured as the additional round trip times that an iterative resolver needs in order to identify a working path. We can see that for lower node failure rates, such as 10%, only a small portion of paths, around 30%, encounters at least one failed server, and the paths almost never double in length. Note that a failure rate of 10% is a realistic failure rate for the DNS system when one takes into consideration various configuration errors [13]. In the extreme cases of very high failure rate, such as 70%, almost all the queries encounter a failed server, and the paths can become even seven times longer.

Similarly, node failures have an impact on the performance of the DHT systems. Figure 7 gives the increase in the path lengths of the DHT system for the different percentages of node failures. Comparing this graph with the corresponding DNS graph, we see that failures have a much greater impact on the DHT systems. Indeed, in the DHT system queries encounter more failed nodes, given that path failure rate is higher than in the DNS case. In order to overcome this limitation, nodes in DHT systems usually monitor the status of their neighbors and thus they avoid forwarding queries to non-responding servers. By following this approach, the increase of path lengths in DHT systems becomes comparable to the one in DNS (we present this results in our technical report [12]).

2) Availability & Caching: In this section we investigate the impact of the caching mechanisms on the availability of

Failed Nodes		10%	30%	50%	70%	90%
DNS Path Failures (%)	No Caching	0.03	0.66	3.62	13.69	46.38
	Caching	0.01	0.24	1.29	4.63	17.59
DHT Path Failures (%)	No Caching	0.57	7.03	26.29	70.45	98.62
	Caching	0.20	2.69	12.15	53.81	98.12
DNS Data Failures (%)	No Caching	0.05	1.57	7.96	24.24	61.81
	Caching	0.05	1.48	7.65	23.92	61.89
DHT Data Failures (%)	No Caching	0.18	1.39	10.29	23.65	65.46
	Caching	0.06	1.28	9.19	17.45	64.78

TABLE III
DNS AND CHORD FAILURE RATE WITH CACHING

the two systems. Table III shows the path and data failure rates for DNS, when caching is enabled. We can see that the data failure rate remains the same, while the path failure rates are considerably lower compared to the case when caching is disabled. Indeed, a caching server takes advantage of the zones NS records, whenever cached locally, by avoiding querying all the involved zones, starting from the root zone. Thus, even if the parent zone is not available, the caching server still has the ability to contact the authoritative server of the child zone. On the other hand, when all authoritative servers of a specific zone are unavailable, the presence of the its NS records in the local cache does not make any difference, given that there is no server available to answer the queries.

Similarly, Table III shows the impact of caching on the availability of DHT systems. The results are based on a network of 8,192 nodes where 10% of the clients issue queries. We see that the path failure rate decreases with caching enabled, but the relative difference is not as big compared to the DNS case. Indeed, caching in DHTs can improve the availability only opportunistically, meaning that a query,

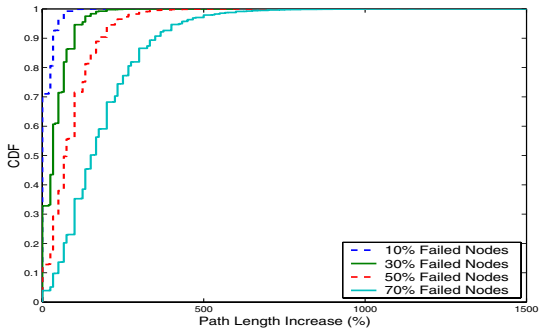

Fig. 6. DNS path length increase

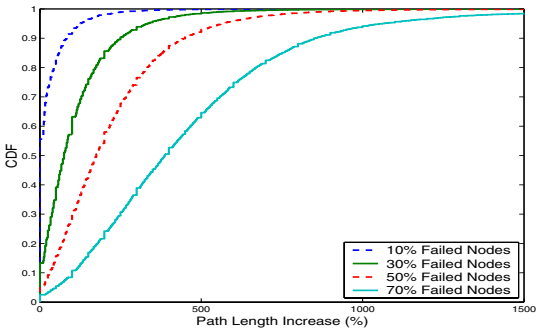

Fig. 7. Chord path length increase

Attack Duration (Hours)		3	6	12	24
DNS Path Failures (%)	100 Nodes Attacked	0.06	0.10	0.19	0.20
	500 Nodes Attacked	7.57	7.91	8.53	8.78
	1000 Nodes Attacked	20.32	20.49	20.82	22.64
DHT Path Failures (%)	100 Nodes Attacked	0.24	0.23	0.26	0.26
	500 Nodes Attacked	0.64	0.64	0.65	0.67
	1000 Nodes Attacked	0.75	0.74	0.76	0.77

TABLE IV
DNS AND CHORD PATH FAILURE RATE UNDER ATTACK

Number of Hops		1	2	3	4
Trace 0	Queries (%)	83.14	16.01	0.84	0.02
	Hit Rate (%)	83.14	94.91	97.59	100.00
Trace 1	Queries (%)	79.62	18.78	1.56	0.04
	Hit Rate (%)	79.62	92.17	97.81	99.19
Trace 2	Queries (%)	74.73	22.56	2.64	0.07
	Hit Rate (%)	74.73	89.26	97.41	99.42

TABLE V
DNS CACHE PERFORMANCE

that is going to fail later, may hit a node with a cached record by chance. On the other hand, caching in DHTs can improve data availability for the reason that cached records are actually replicated records (with loose integrity control). Later, in Section V-B we explore in more detail the specifics of DHTs' cache.

3) Availability & Malicious Attacks: In this section we consider the failure model where an attacker tries to cause the maximum damage. In the case of DNS the attacker makes the nodes placed on the top of the tree unavailable. Table IV shows the path failure rates when the attack duration ranges from 4 hours to 1 day, and the attacker has the ability to set out of operation 100, 500 and 1,000 nodes. For DNS, we see that for the 100 nodes, which shutdown the root zone, the failure rate is low, given that most queries start from the TLDs, which are almost always cached. In contrast when the number of failed servers is 1,000, almost 1% of the total number of servers, the path failure rate is very high, more than 20% and almost constant for the whole duration. In contrast 1% of random failure rates have a negligible effect in the DNS structure. The results above strongly indicate that the tree structure is vulnerable to malicious attacks, even if it is over-provisioned for normal operations.

For DHTs, the attacker shuts down all nodes that are placed in a continuous subspace of the virtual ring. We use a Chord network of 8,192 nodes and the attacker has the exact same abilities as in the DNS case, i.e. he can attack 100, 500 or 1,000 nodes. Our simulation results show that the path failure rate for these attacks are 0.26%, 0.67% and 0.77% respectively, which are considerably lower than DNS. If we compare the failure rate of this specific attack with the failure rate of random failures we see that the attacker cannot create considerably higher damage. For example, with 30% of the nodes under attack the path failure rate is around 15%, instead of 7% under random failures, and for 50% of failed nodes, the failure rate is around 49%, instead of 26%.

*4) **Summary of Results**:* DNS provides better availability under random failures for the following reasons: The hierarchical DNS structure favors the higher level nodes with a higher degree of replication. This leads to significant improvements on the overall system availability. In addition, the vast majority of DNS nodes are essentially leaf nodes, which do not forward queries, thus their failures can not considerably affect the global availability of the system. In contrast, all nodes in DHT networks have the same significance and one needs to improve all nodes' reliability in order to achieve better overall system availability. However, the results change dramatically with intentional attacks. The features that make DHTs weaker under random failures also make DHTs extremely robust under orchestrated attacks. Indeed, all DHT nodes have the same significance and an attacker cannot easily identify a strategy that will cause the maximum damage to the system. Finally, we showed that caching considerably improves path availability in DNS and data availability in DHTs.

B. Cache Performance

In the previous section we saw that the availability of both systems depends to a great extend on the caching mechanisms. In this section we take a more detailed view of caching and we evaluate its effectiveness. The evaluation of the DNS system is based on measurements performed on the packet traces, while the evaluation of the DHT systems is based on the analytical model of Section IV-B.0.b.

1) Caching in DNS: Table V shows the percentage of queries that are answered within a given number of application layer hops, and the cache hit rate for each number of hops. We can see that for all three traces a large fraction of queries, ranging between 75% to 83%, is answered within one hop. Moreover, almost all queries are answered within two hops,

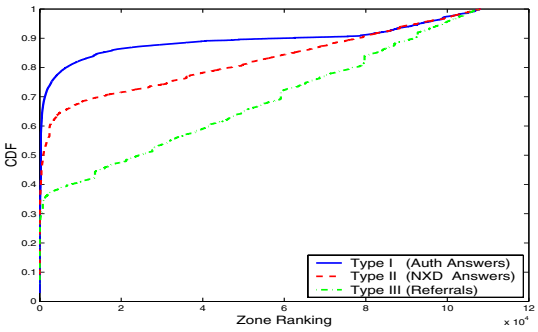

Fig. 8. Relation between cache hits and types of replies

which is an interesting result given the fact that most queries for all three traces require three or four hops in order to be resolved, when caching is not enabled. In addition, we can see that the cache hit rate increases as the path lengths increase. In other words, if there is a cache miss in trying to identify the NS records for a zone under question, then the probability of finding the NS records of the parent zone is very high. Moreover, in the case of a cache miss for the parent's NS records, the probability of finding the grand-parent's NS records is even higher, and so on.

While the above results indicate that caching in DNS is very effective, it is not clear which parameters of the system contribute to these results. Previous studies identified the main reasons of the effectiveness of DNS caching by considering the cache hit rate of individual records [8]. In contrast, our goal is to study the effectiveness of DNS caching when it comes to the cache hits of individual zones. More specifically we answer questions such as: which type of replies make the caching of DNS zones more effective, or what aspects of the DNS structure increase the zones' cache hit rate. Finally, we evaluate how close the simple analytical model, developed in Section IV-B.0.a, comes to the actual measurement results.

In order to identify the reply type that has the highest impact on the performance of DNS caching, we rank all the zones based on the number of replies they generate in TTL seconds, where TTL is their NS record TTL value. The ranking is done for each type of query separately: Thus, the zone with the highest number of authoritative answers (Type 1) in a period of one TTL is ranked number one for that type of reply, but it is not necessarily ranked number one for the other two types. Figure 8 gives the CDF of the cache hit rate for the three different types of zone ranking. From the graph we conclude that zones which reply with a high number of authoritative answers contribute the largest portion of the cache hit rates. On the other side, zones that reply with a high number of referrals contribute a smaller portion of the NS records cache hit rates. These results match our expectation, given that replies of Type 1 are the only ones that can refresh the cached NS records of the zone under question.

Table VI gives the average cache hit rate for the different levels of the DNS tree, by accounting only the zones that appear to have at least one delegation. Based on the this table, we conclude that the top level domains (level 1) are almost always locally cached, while the hit rate for the lower ones

Level	1	2	3	4
Hit Rate (%) (Trace 0)	98.55	89.65	85.41	86.80
Hit Rate (%) (Trace 1)	97.90	80.29	80.24	86.08
Hit Rate (%) (Trace 2)	95.84	78.38	54.98	77.78

TABLE VI
AVERAGE CACHE HIT RATE AND ZONE'S DEPTH

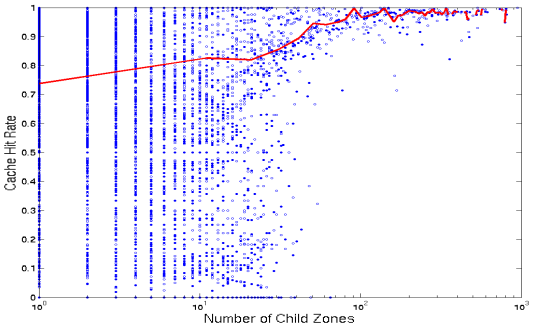

Fig. 9. Cache hit rate and number of delegations

decreases gradually. These results show that zones with more delegations, which usually happen to be placed higher in the DNS tree structure, have a higher hit rate. Indeed, Figure 9 shows the scatter plot of the zones' cache hit rate, for the different number of delegations appearing in the traces. It also shows the smoothed average of the cache hit rate (solid line). It is clear that zones with a high number of delegations have a higher cache hit rate than the average case. While the average cache hit rate is around 80%, zones with more than 100 delegations (appearing in our trace) have a cache hit rate higher than 95%. This difference can be attributed to the higher number of referral replies that come from those zones.

In Section IV-B.0.a we showed that the hit rate of a zone's NS records is bounded below by the total number of replies coming from that zone. Figure 10 verifies our conjecture. It shows, in a scatter plot, the NS records hit rates for the different number of replies, measured within a period of time equal to NS records TTL value. The solid line shows the lower bound specified by Equation 3. Most zones have a cache hit rate higher or close to the bound. We should point out that some of the zones lie lower than the predicted bound. The main reason is that for each zone we compute the average number of replies per TTL, for the duration of the 12 days. Thus, it is possible that the average may not correspond to the actual number of queries appearing in each TTL epoch,

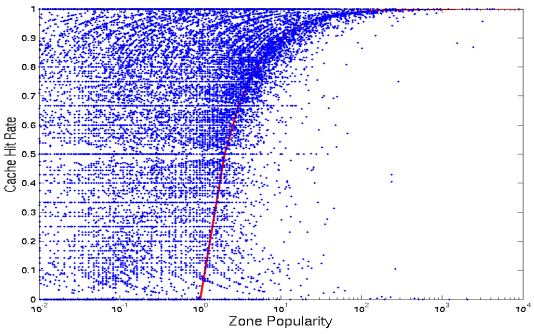

Fig. 10. Hit rates for different zone popularity.

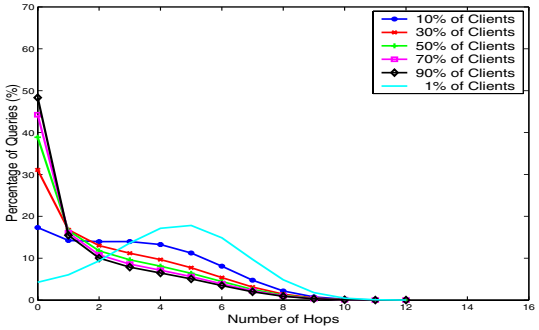

Fig. 11. Chord cache performance with constant network size

Clients	1%	10%	30%	50%	70%	90%
Hit rate (%)	61.7	86.0	92.4	94.6	95.7	96.4

TABLE VII
CHORD HIT RATE WITH CONSTANT NETWORK SIZE

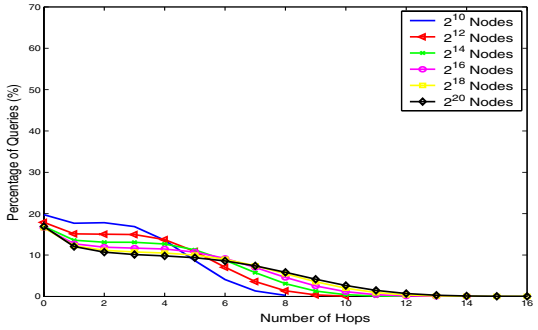

Fig. 12. Chord cache performance with constant system size

Nodes	1024	4096	16384	65536	262144	1048576
Hit Rate (%)	69.1	81.2	89.9	95.4	98.3	99.6

TABLE VIII
CHORD HIT RATE WITH CONSTANT SYSTEM SIZE

something that can skew the results.

2) Caching in DHTs: We used the analytical model of Section IV-B.0.b to evaluate the performance of caching in the DHT systems. Furthermore, we used Trace 1 from Table I to feed the analytical model with the record popularity distribution. We need to clarify that the term "client" refers to the equivalent of caching server in the DHT networks and not to individual end-hosts, i.e. stub resolvers.

Figure 11 shows the cache performance when the number of participating nodes is constant and the number of clients changes. The graph shows the number of queries for which there is a cache hit at a certain distance from the querying node. Note that when the hop number is 0 the record is locally cached. The results are based on the analytical model for a system size of 8,192 nodes. The previous graph shows only the path lengths for the queries that experienced a cache hit somewhere in the system. The total number of these queries is not the same for the different number of clients, as Table VII shows. The above results are indicative that the performance of cache in the DHT systems merely depends on the global popularity of the resource records. Thus, records that are accessed by a small number of clients have a relatively poor cache performance, whereas records accessed very frequently by a large number of clients exhibit much better performance.

Figure 12 gives the cache performance when the relative number of clients over the number of servers remains the same (constant system size). It shows that the cache hit rate deteriorates as the size of the system increases. On the other hand, Table VIII shows that the percentage of the queries that experience a cache hit increases as the size of the system increases. Thus, when the number of clients and servers increase both at the same rate, the absolute number of cache hits increases. However, because the cache hits happen far from from the querying node, the overall performance decreases. The above scenario can happen under a situation where the number of clients increases, which causes an additional load in the system that can be sustained only with a corresponding increase in the number of servers.

In a different scenario, where only the number of servers increases, for example in order to improve the redundancy of the system, or in order to accommodate more resource records in the system, the performance of the system becomes different. Figure 13 gives the cache performance when the number of clients is constant and the size of the system changes, and it shows that the cache performance deteriorates when the system size increases. Indeed, when the number of servers increases and the number of clients remains the same, the relative number of clients decreases and thus the cache hit rate becomes worse. On the other hand, the number of queries that experience a cache hit is almost the same, as we show in Table IX.

All previous results suggest that the use of a proactive replication schemes [15] becomes necessary when the relative number of clients and servers is low. In that case passive caching cannot provide performance comparable to DNS cache performance, but a proactive caching scheme can be tuned in order to provide the desired performance. On the other hand, when the number of clients that request the same records is very high, then both proactive and active caching have comparable performance, given that records are cached virtually everywhere in the DHT system.

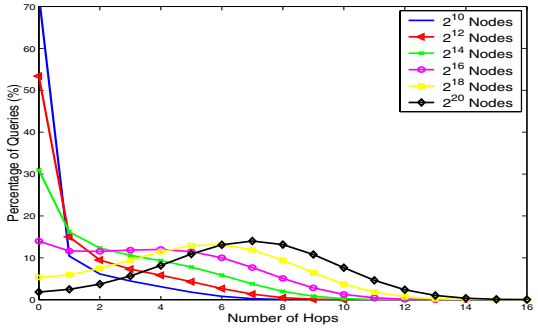

Fig. 13. Chord performance with constant number of clients

Nodes	1024	4096	16384	65536	262144	1048576
Hit Rate (%)	96.4	95.8	95.2	94.6	94.1	93.6

TABLE IX
CHORD HIT RATE WITH CONSTANT NUMBER OF CLIENTS

*3) **Summary of Results**:* In DNS, the effectiveness of caching is driven by the cache hit rate of "hint" pointers, i.e. the NS records, that can route a query close to the destination. The cache hit rate of those pointers is only a function of the local query distribution of each individual caching server. More specifically, the hit rate is higher for the zones that are queried more often, and for the zones that are higher in the DNS tree hierarchy. In contrast, caching in DHT systems is a function of the global popularity of records. Thus, when viewed from the perspective of one caching server, caching in DHTs is effective only when the server's query distribution follows roughly the global query distribution [3].

VI. Discussion

After quantitatively comparing the performance and robustness of the current DNS system and a DHT-based design, in this section we step up a level and consider two basic design issues in engineering a distributed system: engineering provisioning, and functionality versus complexity.

a) Engineering Flexibility: A common rule-of-thumb in improving computer system performance is to *"Make the common case go fast"*, and this rule has been widely applied to engineer the deployed DNS system. For example, because the top level domains of the DNS hierarchy are most critical for both the service availability and the system performance, they tend to have a much larger number of replicated servers compared to a regular zone at a lower level. Similar engineering enhancement is also done for most popular domains, e.g. those of popular websites, or for the domains which are deemed worthwhile. The enabler factor for such *selective engineering* is a design which divides a distributed system into separable pieces, as is done in DNS, so that one can enhance individual pieces without affecting other parts. In addition, the design of DNS caching naturally allows "the common case go fast", in that the NS records of top level domains and popular domains are almost always in the cache, which leads to very short look up paths even when the DNS data itself may change frequently.

In contrast, because the DHT design treats all the names equally in a flat space, it is difficult to separate some parts out for additional engineering tuning. Treating all the names equally facilitated the design of letting all the nodes play the same role. Consequently, performance improvements can only be done by improving the performance for the entire system but not selected parts, and if any names require exceptionally high availability service, that goal can only be achieved by providing all the names with the same high availability.

At the same time, a design with separable parts can also be abused. If some critical parts can be readily identified, they become easy targets for deliberated attacks. In this respect, one may argue that a hierarchical structure can be more vulnerable compared to a flat structure if the system is not designed to withstand concentrated attacks against the top of the hierarchy, as we have shown in this paper.

[3]Even a proactive replication scheme [15] cannot overcome this limitation

b) System Complexity: Despite the specific structure of a system, one can always enhance it by adding any missing functionality. For example, one can overcome the low caching performance of a DHT-based name look up system by implementing additional proactive caching mechanisms [15]. While implementing additional mechanisms on top of the system's basic structure can be a viable strategy, we would like to point out that no additional functionality comes for free, rather each of them brings with it added complexity and overhead. As the DNS system today is burdened with usage modes not envisioned in the original design (e.g. load balancing etc), DHTs can also strain under the burden of additional complexity from mechanisms required to improve performance and availability. A fine balance needs to be maintained between improving performance and increasing complexity.

c) Generality of our Conclusions: In this paper we compared DNS with a specific DHT implementation, the Chord. While most of our results would have been different under a different DHT implementation, our main conclusions apply to all DHT designs of a flat structure. For example, the fact that the DNS structure is more resilient to random failures than the Chord structure, with the same average node degree connectivity, holds for many other DHTs, given that Gummadi *et al.* [5] showed that the Chord structure is the one with the highest static resiliency among other DHT structures. Similarly, the results of the performance of passive caching in Chord, can be applied to other DHTs, whose path length distribution follows a binomial distribution, such as the CAN protocol with large number of nodes [10].

Similarly, one can argue that our results are specific to the failure modes that we consider. For example it is possible that node failures may be correlated, due to running the same software on all nodes. This is possibly a realistic scenario for a DHT system that is deployed by just one organization. On the other hand, DNS and DHT systems deployed by multiple organization run multiple versions of the same software, or even completely different implementations of the same protocol, which makes the events of correlated software failures more rare. Thus we believe that our failure modes cover a range of cases that are most possible to happen in a real system.

VII. Related Work

The first study [3] to explore an alternative design for DNS, based on the DHT systems, concluded that, despite the attractiveness of auto-configuration and resistance to DoS attacks, the long path lengths of DHT networks pose a hurdle in deploying such a system. In this paper we explore the different aspects of DHTs that affect their performance and investigate whether DHTs can achieve the same level of performance as DNS. A subsequent study [15] showed how a proactive replication mechanism can considerably improve DHTs' performance. While we do not consider the effects of proactive caching in this paper, we show when such a mechanism is necessary, and whether passive caching can provide comparable performance benefits.

The proliferation of different DHT-based system designs [17], [16] has created the need for studies that compare the relative benefits of each design and that identify trade-offs applicable to all DHT systems. In this area, the comparative studies [10], [5], [9] concentrated on the fault resiliency, the routing flexibility, and the behavior of various DHTs under churn, respectively. In contrast, our work concentrates on understanding the relative advantages and disadvantages of DHT designs of a flat structure, when compared to DNS, a hierarchical structure system. This comparison has enabled us to identify key issues not explored by previous comparative studies.

Several recent studies have proposed hybrid systems, combining features from hierarchical and flat systems. Specifically, works such as [18], [14], [4], have added features from DHT systems into different sub-parts of the DNS system. Similar efforts such as [2], [6], have included features from DNS into a DHT design. These approaches lead to hybrid systems with unique features. On the other hand, it is not clear how to attribute specific properties of each system to the numerous mechanisms used, and it is questionable if a hybrid design can always provide the best of the two designs. Our work focuses on understanding the relative benefits of hierarchical and flat structures and attempts to shed light on the advantages of each system. In this respect, it illustrates the design trade-offs that should drive any future hybrid systems.

While the effectiveness of DNS caching has been extensively studied in the past from the perspective of queries sent by stub resolvers to caching servers[8], we extend the previous results by showing how caching affects the performance of queries sent from a caching server to the Internet.

VIII. Conclusion

In this paper we compared two different implementations of distributed naming systems, one is the current DNS design that organizes all the name servers in a hierarchical tree structure, and the other is a DHT-based design using a flat peer-to-peer structure. We used a combination of analysis and simulations to evaluate the availability and performance of both implementations. Our results show that while DHTs with flat structure can only cache data, DNS can cache the hierarchy itself. This feature allows DNS to outperform DHTs under normal operation, with higher availability under random node failures and better cache performance for typical records. On the other hand, hierarchy makes DNS vulnerable to orchestrated attacks, something that DHT designs can handle naturally, due to the fact that all nodes have the same importance from a system perspective.

Our results suggest that replacing the current DNS system with a DHT system of a flat peer-to-peer structure will only make the service more resilient to orchestrated attacks, but it will not provide any additional benefits in terms of systems performance and availability under normal failures. On the contrary, a DHT-based naming system can achieve the same level of performance as the current DNS system only with a structure of a higher average node degree and with additional mechanisms, such as proactive caching [15]. Based on these observation, our position is that improving the resilience of the current system against malicious attacks is a more appealing solution compared to replacing the current system with a completely new design, that can achieve the same performance under normal operations and only with a higher cost.

In finishing this paper we would like to draw some general conclusions from our results. Like all other engineering designs, we showed that both structures have their advantages and disadvantages, with no clear winner. Our investigation into the *exact* behavior of both systems has deepened our understanding on how each of them works, which may seem like a surprise, as the DNS has been with us for over 20 years and DHTs, although a relative newcomer, have attracted extensive evaluation efforts. This shows us that we do not necessarily understand how a system works or why it works well, and even a seemingly well understood system may show surprises when analyzed in detail.

References

[1] Nameserver DoS Attack October 2002. http://www.caida.org/projects/dns-analysis/, 2004.

[2] A. Mislove and P. Druschel. Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays. In *Proceedings of IPTPTS*, 2004.

[3] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS Using a Peer-to-Peer Lookup Service. In *Proceedings of IPTPS*, 2002.

[4] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing Content Publication with Coral. In *Proceedings of NSDI*, 2004.

[5] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of SIGCOMM*, 2003.

[6] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A Scalable Overlay Network with Practical Locality Properties. In *Proceedings of USITS*, 2003.

[7] J. Jung and H. Balakrishnan. Modeling TTL-based Internet Caches. In *Proceedings of Infocom*, 2003.

[8] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and the Effectiveness of Caching. In *Proceedings of SIGCOMM IMW*, 2001.

[9] J. Li, J. Stribling, R. Morris, F. Kaashoek, and T. Gil. A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs Under Churn. In *Proceedings of INFOCOM*, 2005.

[10] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic Analysis of Structured P2P Systems: Routing Distances and Fault Resilience. In *Proceedings of SIGCOMM*, 2003.

[11] P. Mockapetris and K. J. Dunlap. Development of the Domain Name System. *SIGCOMM CCR*, 1988.

[12] V. Pappas, D. Massey, A. Terzis, and L. Zhang. A Comparative Study of Hierarchical and DHT Based Naming Systems. Tech. Report UCLA-CS-TR050023, 2005.

[13] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang. Impact of Configuration Errors on DNS Robustness. In *Proceedings of SIGCOMM*, 2004.

[14] K. Parka, V. Pai, L. Peterson, and Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proceedings of OSDI*, 2004.

[15] V. Ramasubramanian and E. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proceedings of SIGCOMM*, 2004.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of SIGCOMM*, 2001.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of SIGCOMM*, 2001.

[18] H. Yang, H. Luo, Y. Yang, S. Lu, and L. Zhang. HOURS: Achieving DoS Resilience in an Open Service Hierarchy. In *Proceedings of DSN*, 2004.