Information Leak in the Chord Lookup Protocol

Charles W. O'Donnell and Vinod Vaikuntanathan
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139, USA
{cwo,vinodv}@mit.edu

Abstract

In Peer-to-peer (P2P) systems, it is often essential that connected systems (nodes) relay messages which did not originate locally, on to the greater network. As a result, an intermediate node might be able to determine a large amount of information about the system, such as the querying tendencies of other nodes. This represents an inherent security issue in P2P networks. Therefore, we ask the following question: Through the observation of the network traffic in a P2P network, what kind of information can an adversarial node learn about another node in the same network ? In this paper, we study this question in the case of a specific P2P system - Chord [10]. We also study the effects of the parameters of Chord (such as finger-table size) and the various enhancements to Chord (such as location caching and data caching) on the amount of information leaked.

1. Introduction

Peer-to-peer (P2P) systems that are not designed with security as one of the main objectives, tend to require a remarkable amount of trust from their participants for correct operation. In particular, a node must trust that the other nodes implement the same protocol and that they do not pool together data to infer some global information about the system. On the other hand, designing a P2P system that is perfectly secure usually involves a considerable compromise in efficiency. In this context, it is interesting to see how much security is preserved by a P2P system even though it is *not designed to be secure*.

Privacy in a distributed system has different manifestations - *data privacy*, wherein a participant $\mathcal{P}$ wants the content of his communication to be hidden from eavesdroppers and *anonymity*, wherein $\mathcal{P}$ does not want eavesdroppers to know that it is participating in some communication.

Anonymity is a desirable property of a distributed system such as a Peer-to-Peer (P2P) network. In the context of a P2P system, anonymity has at least two dimensions - *Requester anonymity* protects the privacy of the initiator of a message whereas *Storage anonymity* attempts to hide the eventual destination of a message. Unstructured P2P systems such as Freenet [2] have been designed with anonymity as one of the main goals, whereas P2P systems such as Gnutella [9] and structured lookup protocols such as Chord [10] do not mention anonymity as a design objective. In this paper, we evaluate the anonymity of the Chord lookup protocol in the presence of a restricted class of adversaries that we call *passive observers*.

Chord is a basic lookup mechanism that can be used as a building block in a larger protocol that implements, for example, a distributed filesystem. The Cooperative Filesystem (CFS) is an example of such a higher level application that uses Chord [3]. Such an application might enhance the features of Chord, such as data caching, which potentially affects the anonymity properties of the system. We study some typical situations in which Chord is used and how these situations affect the anonymity of Chord.

Many variations of the Chord protocol make it highly parameterizable. We study the effect of various parameters such as finger table size and location caching on the anonymity of the Chord lookup protocol. Specifically, we show that such variations improve the requester anonymity without degradation in some other parameters of concern such as lookup time.

2. Outline of Chord protocol

In this section, we provide a brief description of the basic version of Chord [10], the anonymity of which we subsequently analyze. It is necessary to work with a specific version of the protocol, since the different versions have widely differing degrees of anonymity (as we show later).

Chord is a distributed hash table (DHT) that supports just one operation: given a key κ, determine the node N where κ is stored. The nodes and data items are assigned identifiers from flat key space, typically from the set $\{0, 1, \ldots, 2^m - 1\}$. The identifier of a node is derived by hashing its IP address and a virtual node identifier, using a cryptographic (one-way) hash function. The identifier of a data item is computed by hashing the keyword descriptors of the data. The nodes and data are arranged in the identifier circle modulo 2^m. The successor of a node or data item is the node that is closest (in a clockwise sense) to it in the identifier circle.

Chord uses Consistent hashing [6] to associate data items with nodes. Each data item is stored in its successor node. This invariant is maintained when nodes join or leave. In this work, we are concerned with a stable Chord network. i.e, one in which no node joins or leaves.

Finger Table: Each node n_i stores in its finger table the address of the successor node of the identifier ID(n_i) + 2^i (mod 2^m) for all i from 0 to $m-1$. This information is maintained so that queries can be routed quickly to the destination.

Routing of Queries : Typically, the query for a data item has to be propagated along the Chord ring till it reaches the node that stores the data (assuming that the data item exists in the ring). In the *recursive mode* of querying, each node finds the entry in its finger table that is the closest preceding node of the data item and routes the query to that node. This request gets propagated recursively forward. After it reaches the node that stores the data item, some information about the data is passed back along the reverse path. This information could be either the data itself or the IP address of the node that stores the data. In the latter case, the requester has to contact the node that stores the data (using the IP address that it obtained) and retrieve the data. In the *iterative mode* of querying , the initiator of a request queries nodes that are successively closer to the data item. When a query reaches a node, it replies with the entry in its finger-table that is the closest preceding node of the data item. Finally, the initiator of the request queries the node that actually stores the data, at which point, the data item is returned.

The specific implementation of Chord that we work with uses SHA-1 to obtain the identifiers of nodes and data items. In fact, any one-way hash function whose output "looks random" suffices for our purposes. For more details on the Chord protocol, we refer the reader to [10].

3. Adversaries and Anonymity

Now we describe our model of the adversaries and a quantitative tool we use to measure the anonymity in the system. We are concerned with the case when the adversary is a *passive observer*. A passive observer chooses the virtual node identifier that it wants to use and joins the Chord ring. It follows the Chord protocol in forwarding the queries and has a valid finger table. It logs all the queries that pass through it and tries to infer information about the system. Note that a passive observer can only view those requests which are routed through itself, it cannot eavesdrop on other links.

Others have developed models of anonymity measurement [4], and its affect on communications protocols [11] [12] [8]. We use the anonymity set formulation from [5].

We define the *per-request anonymity set* [5] of a node N with respect to a request x to be the set of possible originators of x as seen by node N. Also, if node N does not see a message x, then the per-request anonymity set of N with respect to x is defined to be the set of all nodes (other than N itself) in the system. This convention is justified because, when a node N does not see request x, it has no information about the activity in the network (such as the routing of request x) and is therefore, completely oblivious of the possible initiators of x.

The larger the per-request anonymity set of a node N with respect to a request x, the more anonymous the originator of x is relative to node N. This intuition is made more precise below.

3.1. Definitions and Notations

Definition 1 (Per-Request Anonymity Set). *The per-request anonymity set for a node N with respect to a request x for a data item D, denoted $P_x^{N,D}$, is the set of possible initiators of request x, as seen by node N.*

When the node N and the data D referred to are clear from the context, we abbreviate $P_x^{N,D}$ to P_x.

Definition 2 (Average Anonymity Set). *The size of the average anonymity set for a node N with respect to data item D, denoted $A^{N,D}$, is the expected value of $|P_x^{N,D}|$, when computed over a uniform distributions on the set of possible requests for the data item D.*

$$A^{N,D} = \mathbb{E}_x(|P_x^{N,D}|)$$

We use the size of the average anonymity set $A^{N,D}$ to measure the sender-anonymity of the system with respect to a particular node N and a data item D. Note that this implicitly assumes that the requests for the data D originates from nodes that are randomly distributed in the Chord ring. This is indeed the case when a cryptographic hash-function (such as SHA-1) whose output "looks" random is used to obtain the node IDs, and the adversaries in question are passive observers.

Hereafter, we use N and M to denote specific nodes in the Chord ring, and n to denote the total number of nodes. Also, m denotes the number of bits in the node identifier.

4. Anonymity in Chord

In this section, we present some analytical results on the anonymity of Chord. First, let us look at how much flexibility an adversary has in placing herself at a specific position in the Chord ring. Note that if the IP address and a virtual node identifier are assigned by a centralized authority, the passive observer can do no better than random in positioning herself in the Chord ring. This is because of the fact that the output of the hash function is randomly distributed.

Another issue that confronts us is to determine how many requests a particular node sees on an average. The larger the number of requests for a data item D that a node sees, the better is its estimate of the frequency with which D is accessed.

Theorem 1. *Given a data item D, the expected number of messages that traverse a random node A is $\Theta(\log n)$.*

Proof. Assume that each node sends out a single request for a particular data item. Each request passes through at most $O(\log n)$ hops before it reaches the data. The total number of pairs (N, R) such that node N has seen request R is, therefore, $O(n \log n)$. The average number of requests that a particular node N sees is therefore, $O(\log n)$.

The results of our simulations show that this value is close to $\frac{1}{2} \log n$. Therefore, a randomly placed node sees roughly $\frac{\log n}{2n}$ fraction of the requests for a data item D.

Next we estimate the amount of requester anonymity and storage anonymity that the two versions of Chord - iterative and recursive - provide.

4.1. Storage Anonymity

We say that a P2P system provides storage anonymity if the requests for a particular data item does not reveal the node where the data is stored. It is easy to see that neither the iterative nor the recursive flavor of Chord provide *any* storage anonymity. In both the iterative and recursive version, the adversary could request for a data item D, and in the response obtain the IP address of the node that stores D.

4.2. Requester Anonymity

It is an easy observation that the iterative version of Chord provides *no* requester anonymity at all. But it turns out that the recursive flavor of Chord provides high degree of anonymity against passive observers in a certain statistical sense.

Let us obtain some intuition about our result. First of all, observe that if the (adversarial) node N is further away from the data D, it sees fewer requests for D, and is therefore, more oblivious of who is requesting D. On the other hand, when N does see a request for D, it is much more certain about where the request originated from. For example, in Figure 1, when node 1000 gets a request for the data, it *knows* that the request must have originated from 0000. On the other hand, when N is close to D, it sees more requests, but for every request that N sees, the number of possible originators is relatively large. This trade-off makes it interesting to study how the anonymity varies with the distance between N and D.

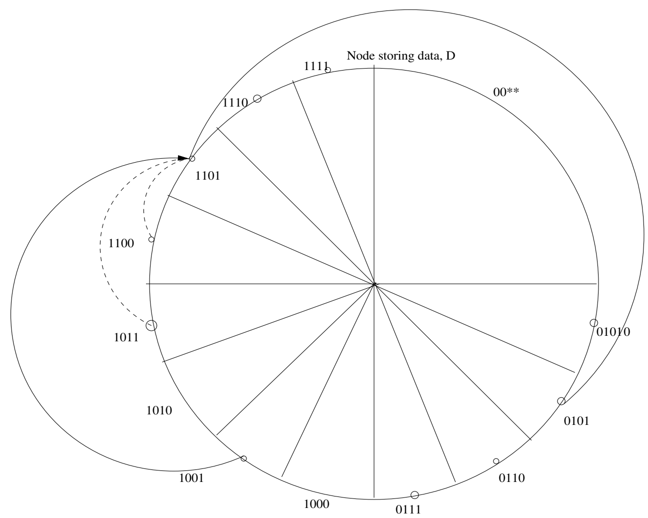


Figure 1. Slices in a chord ring of size 16. The slices are numbered relative to the data item. The arrows indicate that the node at slice 1101 can receive a request for data D from nodes at slices 1100, 1011, 1001 or 0101.

Theorem 2. *Given a data item D, the expected size of the anonymity set of a node N at a distance d (counter-clockwise) from the data with respect to the requests for D, i.e, $A^{N,D}$ is at least $\frac{n}{12d^2} + n(1 - \frac{1}{d}) - 2$.*

Proof. Divide the Chord ring into equally-sized slices labeled $0, \ldots, n-1$ such that the node storing the data item is in slice 0. Also assume that the nodes are uniformly distributed in the Chord ring, such that each slice of $\frac{2^m}{n}$ identifiers contains exactly one node and that node appears in the midpoint of the slice (Figure 1) - we call this the *uniformity assumption*.

We represent the slice number s as a binary string of length $\log n$. i.e, $s \in \{0,1\}^{\log n}$. We also denote a set of slices such that their slice numbers have 1's in the first k bits and 0 in the $(k+1)^{th}$ bit as $I(k)$. We also define slice $n-1$ to be in $I(\log n)$. For example, $I(0)$ is the set of all slices from $0, 1, \ldots, 2^{\log n - 1} - 1$.

Note that a node in slice x can receive requests only from another node in slice $x - 2^i$ for some i. Suppose the node is in $I(k)$. It receives requests from k other nodes, in slice numbers $x - 2^i$, for $i = \log n - 1, \ldots, \log n - k$. This is because such nodes have x as their longest hop to the data. On the other hand, it does not receive requests from nodes in slices $x - 2^i$, for $i = \log n - k - 1, \ldots, 0$, since such nodes have another longer hop to the data.

Node N will receive from node M all the requests of other nodes pending at node M as well as node M's own request. How many requests does node N see? It can be proven by induction that if a node N is in slice $x \in I(k)$, then it receives $2^k - 1$ requests in total from k other nodes. It can also be proven that x receives 2^{i-1} requests from a node in slice $x - 2^{\log n - i}$. When x receives 2^{i-1} requests from a node, it is confronted with 2^{i-1} possibilities for the origin of those requests (since each node initiated exactly one request, the 2^{i-1} requests that x gets has 2^{i-1} possible origins). Also, node x does not see $n - 2^k$ requests. Therefore the average size of its anonymity set is

$$\frac{1}{n}\Big(\sum_{0<i\leq k} 2^{2(i-1)} + (n - 2^k)(n-1)\Big) \geq \frac{4^k}{3n} + n - 2^k - 2$$

Suppose a node is in $I(k)$. Then its distance d to the slice 0 is bounded as $2^{\log n - k - 1} \leq d < 2^{\log n - k}$. i.e, $\frac{n}{2^{k+1}} \leq d < \frac{n}{2^k}$. This means that the anonymity set of node x is at least of size $\frac{n}{12d^2} + n(1 - \frac{1}{d}) - 2$.

Corollary 1. *The average size of anonymity set over all nodes in the Chord ring is* $\Omega(n)$.

Corollary 2. *The number of requests for a data D seen by a node N at a distance d from D (counter-clockwise) is* $O(\frac{n}{d})$.

Corollary 3. *Assume that a node N gets a request x to data D from node N' which is at a distance d from N. Then, $P_x^{N,D}$ is* $O(\frac{n}{d})$.

Thus we can see that the minimum size of the average anonymity set $A^{N,D}$ of any node N is about $n/12$ (which happens when the node is closest to the data). Since the maximum size of the anonymity set is $n - 1$, it follows that Chord provides a high degree of anonymity.

The uniformity assumption used in the proof is justified because nodes are distributed uniformly in the Chord ring, due to the randomness of the output of the hash function. The simulation results also support Theorem 2.

5. Simulations

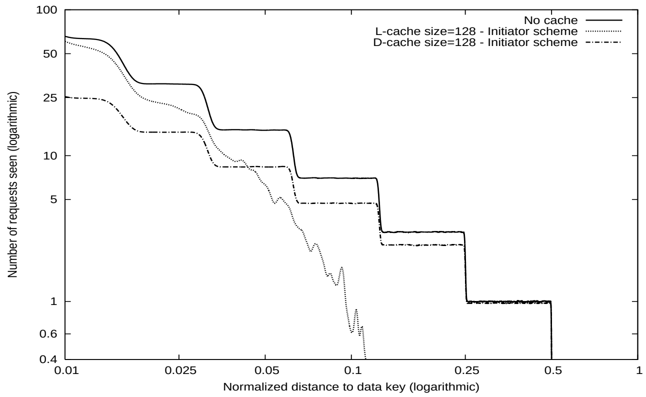
Empirically, we would like to confirm our theoretical predictions of the average size of the anonymity set of a node at any given distance from a data key which is being queried. Also important are the number of requests seen at different distances, which we will normalize to be a distance between 0 and 1. A node which owns the data key κ being queried is considered to be at distance 0, while the furthest possible node from this has the identifier value $\kappa + 1 \bmod 2^m$.

We would also like to inspect the effect of implementation variations to the Chord protocol on average anonymity. The Chord protocol definition requires only that every node maintains a single pointer to that nodes' successor node for protocol correctness. Beyond this, Chord implementations choose to maintain more state information, such as finger table and caches, to improve performance. Naturally the addition of pointers to other nodes decreases the number of hops taken during a request, and consequentially decreases the expected number of requests any node will see around the ring.

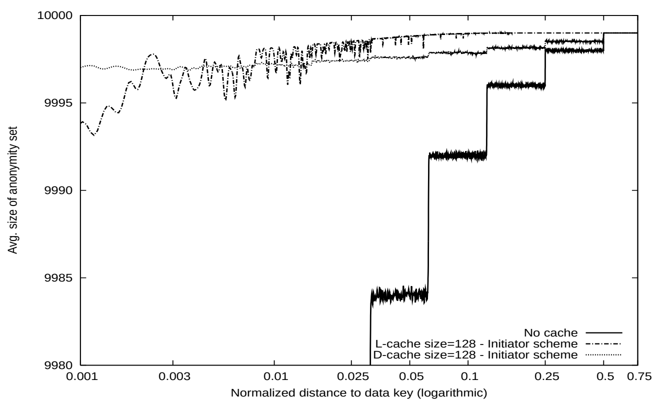
We examine the differences between implementation parameters by simulating lookups on a stable Chord network. Not only does this confirm theoretical results, but most easily shows the patterns of non-deterministic request propagation caused by modifications such as caches. Unless otherwise noted, in all of the following simulations, a ring of $10,000$ uniformly random nodes was created, using 32 bits for the key/node identifiers, a standard finger table size of $\log m = 32$ and successor list size of 1. To determine the average number of requests seen at any given distance, we place 5000 data elements in the network (in a uniformly random manner) and had every node make exactly one request for each element in a similarly random order. All requests are assumed to be carried out successfully, and the results are averaged across all 5000 lookups according to distance. Since every node makes only one request, the average requests seen count is indicative of how many unique nodes have propagated a message through any given distance point, and the count can then be used to determine the average anonymity set size. All graphs are smoothed using a moving window average so as to distinguish the most important trends.

6. Data Caching

The existence of a data cache is one of the more common implementation extensions to a Chord base system. With data caching, every node will cache data values (and corresponding keys) which it has already made a query for and retrieved. Data caching can be independent of the underlying Chord protocol, and exist at a higher application layer. Cache entries do not change the forwarding decisions of lookups, but simply return the requested data before the lookup reaches the responsible owner node. In this fashion, a lookup is routed as it normally would until it is prematurely halted with the halting node returning the requested data. Systems such as CFS [3] implement data caching in what is called the $block\ store$ layer which sits above the $Chord$ layer.



(a)



(b)

Figure 2. Data caching effects on anonymity (a) Number of requests seen per node (b) Anonymity set size per node

In an *initiator* data caching scheme, when an initiating node queries for a specific data key, and receives a valid data value, that data value is placed within the initiating node's data cache, possibly evicting other nodes within the cache. Alternately in a *path caching* scheme, a tighter protocol/caching coupling would allow every node along the request return path to add the data item to its data cache. This is what is done in CFS and gives a much greater dispersement of any one data value around a Chord ring, although at the cost of higher eviction rates (on average $\frac{1}{2}\log N$ nodes will cache a data value on a single request). In all of our simulations, we assume an LRU replacement policy and that caches never become stale or invalid.

In Figure 2(a) we see that adding data caching impacts the number of requests seen at distances close to the data key tremendously. The data cache all but removes the requests seen at the nodes within a very close proximity to the data key. The result, shown in Figure 2(b), emphasizes the difference between the sizes of the uncached average anonymity sets and the cached average anonymity sets. One can see that, in an uncached system, a point must be at a distance of approximately 0.025 to have an average anonymity set size above 99%, and must be at a distance of 0.25 to match the average anonymity set size of a point at 0.1 in a cached system. In a network of $10,000$ nodes, this corresponds to 250 more nodes which boast an average anonymity set size of greater than 99%.

In Figure 2(b) we only mention a data cache of size 128, however, it can be seen from Figure 3(a) and 3(b) that even a single element of cache in every node will exhibit a similar utilization rate. This logarithmic graph shows two nearly overlapping utilization rates for initiator caching, and two nearly overlapping rates for path caching. The difference in data cache size makes almost no difference at distances near to the data key. The utilization rate is the frequency with which the data cache is used, opposed to the finger table, as the source for the next forward hop of a request.

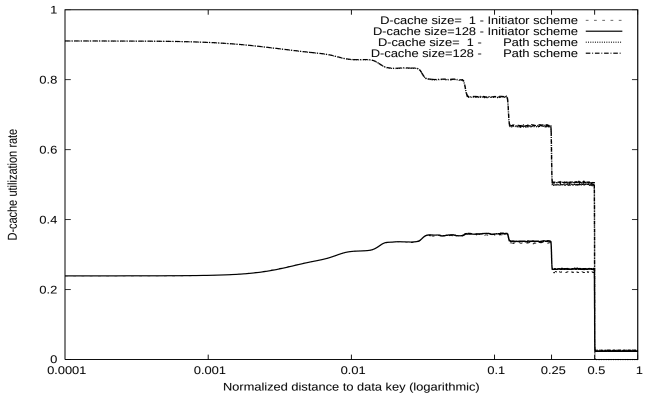
Our simulations tested both initiator caching and path caching, as can be seen in Figure 3(a). Path caching produced a significantly higher utilization rate than initiator caching, likely because single data requests fill the data caches of as many as $logn$ nodes which were on the request path. While the utilization rate increase is evident, it turns out that this does not improve average anonymity set size all that much. This follows as the initiator caching scheme already raises this metric to very high levels, where there is little room for improvement.

7. Routing Variations

A more tightly coupled protocol/application implementation can be created which allows for changes to the forward routing of queries. Unlike data caching, which simply halts a request path, a routing variation will simply supply the lookup protocol with alternative forward nodes to send a request to. This has the possibility to forward a request to a node which is much closer to the data key, decreasing the total number of hops that request will take. Some of the ways these alternative nodes can be discovered are through variable finger tables, successor list extension, and location caching.

7.1. Variable Finger Table Size

As described [10], the Chord protocol prescribes each of the n nodes within the network to maintain a finger table. Accordingly, a network with an m-bit key/node identifier space will have nodes which contain finger tables of size



(a)

$Scheme$	$Cache\ Size$	$Avg. Utilization\ Rate$
$Data/Init$	1	14.7%
$Data/Init$	32	15.0%
$Data/Init$	128	16.2%
$Data/Path$	1	30.7%
$Data/Path$	32	31.0%
$Data/Path$	128	32.2%

(b)

Figure 3. Data caching utilization rates

m. For every node n_i, each i^{th} entry of its finger table is the first node which succeeds ID(n_i) by at least 2^{i-1}.

However, the finger table size is relatively unimportant as long as the entry pattern is maintained. Therefore, it is possibly advantageous to fix the finger table size to a different value to perhaps adhere to strict node memory requirements, or to take advantage of extra memory space. The finger table for a m-bit key/node identifier space can then be defined more generally as a table of size f where the i^{th} finger table entry points to the successor of ID(n_i)+$2^{\frac{m}{f}(i-1)}$. With this, the maximum hop count for a lookup decreases to $\frac{\log N}{1+\log\frac{m}{f}}$, although the finger table size increases by a factor of $\frac{m}{f}$.

Using our model which has $m = 32$ bits in the key/node identifiers, we show the average requests seen per node in Figure 4 given the finger table sizes of $f = \{64, 32, 24, 16, 8\}$ entries.

What can be seen, is that a variation in the finger table size merely causes a shifts in the curve of the requests seen, allowing a few requests to be seen further away from the data key. The number of requests seen in these ranges are minimal though, so the average anonymity set size remains relatively unchanged. The number or requests seen within the closest distances to the data key still increase at an exponential rate, as described by the above equation. Thus, variation in finger table size does very little to boost the most problematic average anonymity set sizes, found at nodes very near the data key.

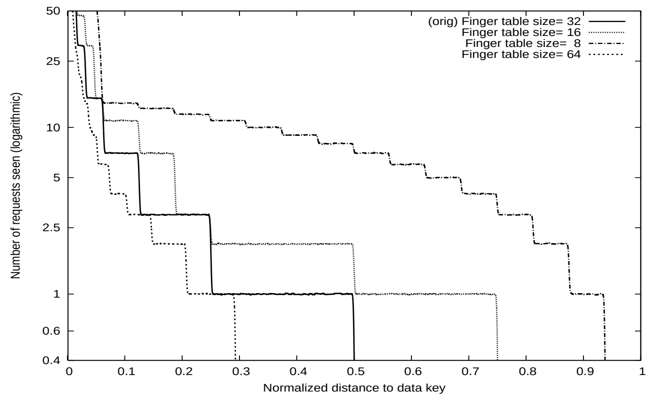


Figure 4. Variable finger table size effects on anonymity

7.2. Successor List size

As already mentioned, for correctness, a chord node is required to maintain one successor node. However, an implementation may choose to retain a list of its S successors to improve lookup performance (as well as aid in node joins/leaves). In this case, when a node n receives a request for a data key κ, a node will first search its successor list to determine if any entry s_i is the owner of that data key ($ID(n) < \kappa < ID(s_i)$). If no entry is the data key's owner, the node continues to check its finger table for forwarding of the request.

By nature, an increase in the successor list tends only aid those nodes immediately preceding the data key owner, as can be seen in Figure 5. Here its quite evident that the average anonymity set size remains nearly the same across all sizes of successor lists until approximately a distance of 0.004 away from the data key. More exactly, the average anonymity set sizes diverge at a distance equal to the size difference between the two successor lists. Increased successor list sizes can boost the average anonymity set sizes at distances near the data-key, however the increase could be considered marginal, a form of "too-little too-late."

7.3. Location Caching

Location caching uses a cache to remember prior queries initiated by a node, just as data caching, however it does not store the data value, but instead the location of the data owner. These data owner nodes are then used in conjunction with the existing statically determined finger table to choose

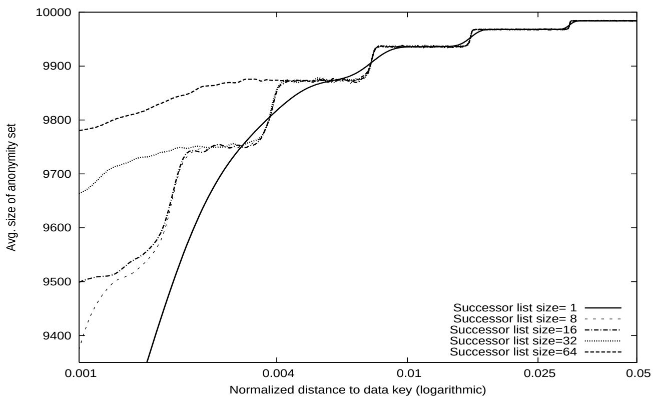


Figure 5. Successor List Avg. anonymity set size for

how to best forward new requests which pass through. This is a kind of *dynamic finger table* which creates entries which branch out to the extent of recent successful requests.

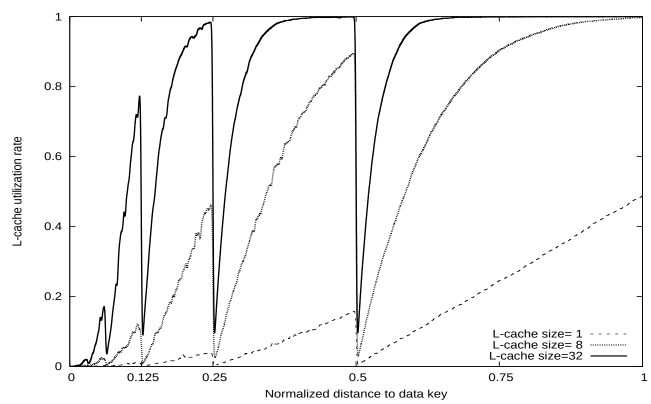
Similar to data caching, an initiator caching scheme will save the identifier of some data's owner in the initiating node's location cache following a successful query for that data. A path caching scheme can alternately have every node along the request path place the data owner identifier into its location cache. We make similar assumptions on the location cache as we did for data caches.

When processing an incoming query, a node along the request path will inspect its finger table and location cache to determine a forwarding node which is closest to the requested data key. It is worth pointing out that location cache entries not only refine the granularity of matches in the finger table, but can also contain nodes which are far beyond the ID$(n_i)+\frac{m}{2}$ maximum entry for an m-bit identifier space system. In this sense a location cache can $extend$ the finger reach of Chord nodes. Further, location cache entries are much more versatile than data cache entries because a location cache entry will match all keys between itself and the nearest preceding finger table entry, instead of just a single data key. Even though a data cache match will terminate the forward path of a query, a location cache can drastically decrease the number of required hops beginning even at the initiator.

Figure 2(b) shows how location caching impacts average anonymity set sizes, while Figure 2(a) shows the number of requests seen at any particular distance. Similar to data caching, it is clear that location caching drastically improves the average anonymity set size at distances near to the data key, although not quite as much. Interestingly, a location cache performs better than a data cache for the distances between 0.05 and 0.5, however a data cache gives better results for the distances which are closest to the data key.

The caching method shown is initiator caching, because it was found that, for location caching, there was almost no difference between path caching and initiator caching, as can be seen in Figure 6(b). This follows because path caching would tend to lower the utilization rate at distances which see more requests on average (because of higher eviction rate). It was also found that the size of the location cache mattered very little to the average anonymity set size. Figure 6(b) shows the difference between varying location cache sizes, however, as can be seen in Figure 6(a), the location cache is predominantly used at distances from the key which normally would have large average anonymity set sizes.

The nature of how the location cache is used can be seen by its intersection with the data cache average requests seen at the distance 0.05 in Figure 2(a), as well as its utilization pattern in Figure 6(a). Distances far from the data key tend to have high utilization rates, where they presumably forward the request to points very near to the data key. Once a request has reaches a distance near to the data key, the chances of location cache aiding are reduced greatly. From this we conclude that location caching does well to increase the average anonymity set size for the vast majority of distances, and data caching does a better job of increasing the average anonymity set size at the very closest distances.



(a)

$Scheme$	$Cache\ Size$	$Avg. Utilization\ Rate$
$Loc/Init$	1	14.4%
$Loc/Init$	8	56.1%
$Loc/Init$	32	80.9%
$Loc/Path$	1	14.3%
$Loc/Path$	8	55.9%
$Loc/Path$	32	80.8%

(b)

Figure 6. Location cache utilization rates (a)(b)

8. Related Work

Other recent work concerning the anonymity of P2P systems has been done, concurrently to our own. Kannan and Bansal [7] also investigated anonymity concerns of a Chord network, but they focus on quantifying the anonymity of *request initiators* as opposed to bounding an attacker's advantage. They emphasize the use of virtual nodes by an initiator as a means to increase anonymity. Borisov and Waddle [1] go further and construct a more general model of anonymity for any peer-to-peer network.

9. Conclusion

We have shown that even though Chord does not have anonymity as a design goal, it does in fact provide a high amount of *requester anonymity*, against restricted types of adversaries. Though upon first inspection, Chord appears highly ordered and publicly observable, we show that a high majority of adversaries have little chance of identifying the initiator of a lookup.

We confirmed our theoretical results via simulations, and investigated possible extensions introduced in Chord systems. We displayed the kinds of tradeoffs involved between the amount of state kept by each participant node, and the amount of anonymity in the system as a whole. We also showed the impact of the different extensions to Chord on its anonymity. Even minimal location caching creating increases the amount of anonymity within a system, while data caching requires marginally larger cache sizes to achieve similar results. Larger successor lists also increase the anonymity set size of nodes close to data keys, an important consideration, however variation of finger table size does little to increase the practical anonymity of a system.

It has to be added that our approach is limited in that we formally analyze the impact of only passive observers. To see how active adversaries can complicate the situation, consider the following simple extension to the passive observer model: We give a node in the Chord ring the flexibility to choose its virtual node identifier(VID) maliciously. This allows a group of passive observers to skew the density distribution in the Chord ring, which could potentially be useful to the adversary.

Further work in this topic could include a consideration of multiple cooperating adversaries, and what a confederation of active adversaries could do to undermine the inherent anonymity of the Chord protocol. It would also be interesting to see an analysis of how splits and merges could affect the anonymity of a system. Investigations into other adversarial attacks would also be useful, such as active attacks which attempt to render a denial-of-service for a major portion of the network.

10. Acknowledgments

We would like to thank Hari Balakrishnan for his invaluable comments and the inspiration for this topic, as well as all of the anonymous reviewers for their useful suggestions.

References

[1] N. Borisov and J. Waddle. Anonymity in structured peer-to-peer networks. *www.cs.berkeley.edu/~kubitron/courses/cs294-4-F03/ projects/ borisov_waddle.pdf*, Dec. 2003.

[2] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[4] C. Diaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[5] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. *Proceedings of the ACM Conference on Computer and Communications Security (CCS 9). Washington, D.C.*, 2002.

[6] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed cachine protocols for relieving hot spots on the world wide web. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[7] K. J. Kumar and M. Bansal. Anonymity in chord. *www.cs.berkeley.edu/~kjk/chord-anon.ps*, Dec. 2002.

[8] A. D. R. Michael K. Reiter. Crowds: Anonymity for web transactions. *ACM TISSEC*, 06 1998.

[9] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Transactions on Networking*, 11, 2003.

[11] M. Wright, M. Adler, B. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. In *ISOC Symposium on Network and Distributed System Security*, Febrauary 2002.

[12] M. Wright, M. Adler, B. Levine, and C. Shields. Defending anonymous communication against passive logging attacks. In *Proc. IEEE Symposium on Research in Security and Privacy*, Berkeley, CA, May 2003.