Reliable and Scalable Distributed Hash Tables Harnessing Hyperbolic Coordinates

Telesphore Tiendrebeogo, Daouda Ahmat, Damien Magoni

Reliable and Scalable Distributed Hash Tables Harnessing Hyperbolic Coordinates

Telesphore Tiendrebeogo, Daouda Ahmat, and Damien Magoni
University of Bordeaux – LaBRI
Talence, France
{tiendreb,adaouda,magoni}@labri.fr

***Abstract*—Distributed hash tables (DHT) need to route requests in a scalable way. Although several solutions do exist, they often require nodes to connect to each others in a given way. Greedy routing schemes based on virtual coordinates taken from the hyperbolic plane have attracted some interest in recent years. Such schemes can be used for building distributed hash tables while letting the nodes connect to the others as they wish. In this paper, we define a new scalable and reliable DHT system based on the use of hyperbolic geometry. We provide a scalability analysis and we assess its efficiency by performing simulations. Results show that our DHT system brings more flexibility to the nodes while still being scalable and reliable in presence of churn.**

***Index Terms*—Distributed hash table, greedy routing, hyperbolic plane.**

I. Introduction

Distributed Hash Tables (DHTs) such as Chord [1], CAN [2], Pastry [3], Tapestry [4] and Kademlia [5] have been extensively studied for their potential to provide scalable data storage and retrieval to various wide scale P2P applications. Existing solutions all have their pros and cons as shown in [6] but they usually have to set up constrained topologies.

We propose a system called Cloak able to build an overlay network without imposing a particular topology on the nodes (such as a ring, a tree or a hypercube). Each overlay node manages its portion of the addressing space and nodes can connect arbitrarily to each other. They can join and leave the P2P network efficiently without the cost of maintaining any global knowledge. Our approach is based on the work of Kleinberg [7] that we have enhanced in order to manage a dynamic topology able to grow and shrink over time. We also propose a specific mapping function to store (key;value) pairs on the overlay nodes such as to make retrieval efficient and to avoid overloading a particular area of the overlay. We have carried out analysis and simulations to demonstrate the potential of our proposal in terms of scalability and reliability to build a DHT using hyperbolic coordinates. This work is a follow up of our previous work [8] describing the use and limitations of the hyperbolic plane as a virtual addressing space.

In this paper, our contributions are as follows:

- We describe the distributed addressing and greedy routing algorithms used by the DHT overlay (Section II).
- We define the mapping algorithm used by our DHT system (Section III).
- We compare the complexity costs of our solution to various existing DHTs (Section IV).
- We provide the results of our performance evaluation obtained by simulations (Section V).

II. Hyperbolic greedy embedding and routing

The model that we use in our system to represent the hyperbolic plane is called the Poincaré disk model. In the Poincaré disk model, the hyperbolic plane is represented by the open unit disk of radius 1 centered at the origin. In this specific model:

- Points are represented by points within this open unit disk.
- Lines are represented by arcs of circles intersecting the disk and meeting its boundaries at right angles.

In this model, we refer to points by using complex coordinates. An important property of the hyperbolic plane is that we can tile it with polygons of any sizes, called p-gons. Each tessellation is represented by a notation of the form $\{p, q\}$ where each polygon has p sides with q of them meeting at each vertex. There exists a hyperbolic tessellation $\{p, q\}$ for every couple $\{p, q\}$ obeying $(p-2)*(q-2) > 4$. Our purpose is to partition the plane and address each vertex uniquely. We set p to infinity, thus transforming the polygons into a regular tree of degree q. This particular tiling splits the hyperbolic plane in distinct spaces and constructs an embedded tree that we use to assign unique addresses to the vertices (that we call nodes hereafter). We call *addressing tree* this embedded spanning tree that covers all the nodes.

In the Poincaré disk model, the distances between any two points z and w are given by curves minimizing the distance between these two points and are called geodesics of the hyperbolic plane. To compute the length of a geodesic between two points z and w and thus obtain their hyperbolic distance $d_{\mathbb{H}}$, we use the Poincaré metric which is an isometric invariant given by the formulae:

$$d_{\mathbb{H}}(z, w) = arg cosh(1 + 2 \times \frac{|z - w|^2}{(1 - |z|^2)(1 - |w|^2)}) \quad (1)$$

This formulae is demonstrated in [9] and used by the greedy routing algorithm 2.

Through the paper, we call peers or nodes, the members of the overlay. The first step in the creation of an overlay is to start the first peer and to choose the degree of the addressing

Algorithm 1 Calculating the coordinates of a peer's children.

procedure CALCCHILDRENCOORDS($peer$, q)
  $step \leftarrow argcosh(1/sin(\pi/q))$
  $angle \leftarrow 2\pi/q$
  $childCoords \leftarrow peer.Coords$
  for $1 \rightarrow q$ **do**
    $ChildCoords.rotationLeft(angle)$
    $ChildCoords.translation(step)$
    $ChildCoords.rotationRight(\pi)$
    if $ChildCoords \neq peer.ParentCoords$ **then**
      STORECHILDCOORDS($ChildCoords$)
    end if
  end for
end procedure

Algorithm 2 Routing a packet in the overlay.

function GETNEXTHOP($peer$, $packet$) **return** Peer
  $w = packet.destinationPeerCoords$
  $m = peer.Coords$
  $d_{min} = argcosh\left(1 + 2\frac{|m-w|^2}{(1-|m|^2)(1-|w|^2)}\right)$
  $p_{min} = peer$
  for all $neighbor \in peer.Neighbors$ **do**
    $n = neighbor.Coords$
    $d = argcosh\left(1 + 2\frac{|n-w|^2}{(1-|n|^2)(1-|w|^2)}\right)$
    if $d < d_{min}$ **then**
      $d_{min} = d$
      $p_{min} = neighbor$
    end if
  end for
  return p_{min}
end function

tree. The hyperbolic coordinates (represented by a complex number) of a node of the addressing tree are used as the address of the corresponding peer in the overlay. Any node of the tree can attribute the addresses corresponding to its children in the tree. The management of the addressing space is thus distributed because a node can compute the coordinates of its children only by using its own coordinates and applying the algorithm 1. The degree of the tree thus determines how many addresses each peer will be able to give. The degree of the tree is defined at the beginning for all the lifetime of the overlay. The overlay is then built incrementally, with each new peer linking to one or more existing peers and obtaining an address from one of its neighbors. In our overlay, a new peer can connect to any other peer at any time in order to obtain an address. If a peer has no free address left, it replies with the list of its neighbors to enable the new peer to extend the search for obtaining an address. Over time, the peers will leave the overlay until there is no peer left which is the end of the overlay. The algorithm 1 shows how to calculate the addresses that can be given to the children of a peer. The first peer takes the hyperbolic address (0;0) and is the root of the tree. The root can assign q addresses while all other nodes can assign $q-1$ addresses.

When a new peer has connected to peers already inside the overlay and has obtained an address from one of those peers, it can start sending DHT request packets (PUT and GET) as explained in section III. The routing process is done inside each peer on the path (starting from the sender) by using the greedy algorithm 2 based on the hyperbolic distances between the peers. When a packet is received by a peer, the peer calculates the distance from each of its neighbors to the destination and forwards the packet to its neighbor which is the closest to the destination. If no neighbor is closer than the peer itself, and the peer is not the destination, the packet has reached a local minima and is dropped. This can happen when the addressing tree is broken by a failed peer or link *and* when no shortcut link exists to bypass this failed peer or link. Of course, this failure is expected to be temporary as the failed peer may restart or a new peer may take its place in the meantime. In a real network environment, link and peer failures are expected to happen often. If the addressing tree is broken and can not be restored as described above in a reasonable amount of time, a partial readdressing can occur. To perform the readdressing, the peers located beyond the failed peer or link (i.e., having addresses derived from the failed or unreachable peer's address) will flush their addresses and attempt to obtain new addresses from peers still inside the tree. Some peers may have first to reconnect to other peers in order to restore connectivity.

III. HYPERBOLIC DHT

In this section we explain how our overlay system stores and retrieves the (key, value) pairs that are used by the applications. Our solution is a structured DHT system that uses the distributed addressing and the greedy routing algorithms presented in section II.

A (key, value) pair is called a *binding*. Figure 1 shows how and where a given binding is stored in the overlay. A binder is any peer that stores these pairs. The depth of a peer in the addressing tree is defined as the number of parent peers to go through for reaching the root of the tree (including the root itself). When the overlay is created, a maximum depth for the potential binders is chosen. This value is defined as the *binding tree depth*. All the peers that have a depth less or equal to the *binding tree depth* in the addressing tree may hold bindings and thus be binders.

When a peer wants to store an entry in the DHT, it first creates a key by hashing the key string with the SHA-1 algorithm. It then divides the resulting 160-bit key into r equally sized $160/r$-bit subkeys (for redundancy storage). This r factor is chosen arbitrarily and can be set to whatever value depending on the amount of redundancy required. In absence of redundancy, the peer selects the whole key. Then the (sub)key is mapped to an angle by a linear transformation.

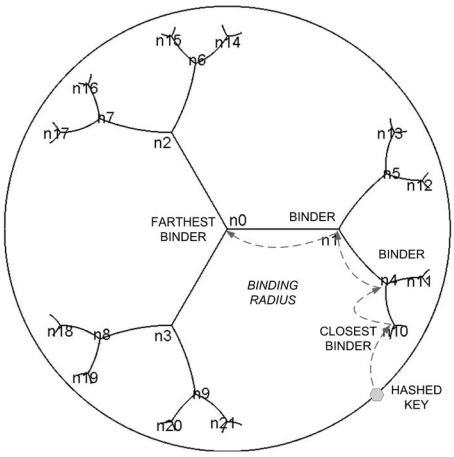

Fig. 1. Hyperbolic DHT system.

The angle is given by:

$$\alpha = 2\pi \times \frac{160/r\text{-bit subkey}}{\underbrace{111...11}_{r}} \quad (2)$$

The peer then computes a virtual point v on the unit circle by using this angle:

$$v(x,y) \text{ with } \begin{cases} x = cos(\alpha) \\ y = sin(\alpha) \end{cases} \quad (3)$$

Next the peer determines the coordinates of the closest binder, given its Euclidean distance, to the computed virtual point above by using the given *binding tree depth*. In the figure we set the *binding tree depth* to three to avoid cluttering the figure. It's important to note that this closest binder may not exist in reality if no peer is currently owning this address. The peer then sends a PUT request to this closest peer. This request is routed inside the overlay by using the greedy algorithm of the section II. If the query fails because the binder does not exist or because of node/link failures, it is redirected to the next closest binder which is the father of the computed binder. This process continues until the query reaches an existing binder peer which can be any peer on the path from the computed closest binder to the center peer. Upon reaching an existing binder, the pair is stored in that binder. The query can thus go up the addressing tree to the center peer having the address (0;0) which is the farthest binder. The path from the computed closest binder to the farthest binder is defined as the *binding radius*.

This process ensures that the pairs are always stored first in the binders closer to the unit circle and last in the binders closer to the disk center. If the addressing tree is imbalanced, many pairs may be stored in peers close to the center thus overloading them. In order to solve this issue any binder peer will be able to set a maximum number of stored pairs and any new pair to store will be rejected and the query redirected as above. Furthermore, to provide redundancy, the peer can repeat the storing process described above for each of the other $r-1$ subkeys. Thus r different binding radiuses can be used and this will improve the evenly distribution of the pairs. In addition, and still for redundancy purposes, a pair may be stored in more than one peer of the binding radius. A binder could store a pair and still redirect its query for storing it in its other ancestor binders. The number of stored copies of a pair along the binding radius may be an arbitrary value set at the overlay creation. These mechanisms enable our DHT system to cope with an non-uniform growth of the overlay and they ensure that a pair will be stored in a redundant way that will maximize the success rate of its retrieval. The number r of subkeys and the number of copies in a given radius are parameters that can be set at the creation of the overlay. Increasing them leads to a tradeoff between improved reliability and storage space cost in binders.

Our solution has the property of consistent hashing: if one peer fails, only its pairs are lost but the other binders are not impacted and the whole system remains coherent. However, this property does not hold true when a partial readdressing takes place as explained in Section II. In this case, all the pairs stored in the peers having addresses derived from the failed or unreachable peer's address are lost. Hopefully this is not a big issue because, as in many existing systems, pairs will be stored by following a hybrid soft and hard state strategy. Thus a pair will have to be stored by its creator every δt period of time otherwise it will be flushed by the binders that store it. These periodic store messages will ensure that pairs lost by a partial readdressing will be restored after at most δt period of time. A delete message may be sent by the creator to remove the pair before the end of the period. We analyze the influence of the degree of the addressing tree on the query success rate and the query path length in Section V.

IV. Scalability analysis

We provide in this section a brief complexity analysis of our proposal and compare the results with other existing DHT systems. We first define the four metrics that we use in our analysis. These metrics were defined and used in [6] and [10].

1) Hops: this metric counts the average number of peers to go through to reach the destination.
2) Paths: this metric counts the average number of paths that are crossing any given peer.
3) States: this metric counts the number of states that must be stored in a peer for the routing to work, it is typically equal to the number of entries found in the routing table of the peer.
4) Churn messages: this metric counts the average number of messages that are exchanged when a peer joins or leaves the overlay.

In our system, the peers in the overlay connect to each others as they wish, thus no strict topology is enforced. Any peer can have as many links as it can with other peers and one link is of course a minimum to connect to the overlay. The only requirement is that the embedded addressing tree which is a spanning tree of the overlay shall remain valid for the greedy routing to work.

Because any overlay will be at least (when no redundant links exist) composed of its addressing tree, the distances between any two nodes are expected to be of the order of $O(\log(n))$ hops. If the peers have a large number of redundant links (i.e., links not belonging to the addressing tree), the distances will be much shorter. If the overlay topology takes the form of a scale free network, the distances will be the order of $O(\log(\log(n)))$ as shown in [11]. Whatever the topology, the number of paths crossing any one peer (its congestion level) will have an expected probability of at most $O(\log(n)/n)$.

When a peer joins the overlay, only its neighbors (i.e., those having setup a link with the new peer) need to update their state information which bears a message cost complexity independent of n. Similarly, when a peer leaves the overlay, only its neighbors need to update their state information also giving a message complexity cost being of the order of $O(1)$. However, if the addressing tree is broken and can not be restored in a reasonable amount of time as explained in section II, a partial readdressing can occur for peers having addresses derived from the failed or unreachable peer's address. In this latter case, which is expected to be very uncommon, the message cost complexity is expected to be of the order of $O(n)$.

Readdressing is needed to provide to the peers the ability of connecting to whatever peers they want. If we force some peers to connect to some specific peers for restoring the addressing tree (as done by Chord, where a peer's IP address determines to which peers it must connect) then the message cost complexity is expected to be of the order of $O(1)$ for a leaving peer. Thus readdressing must be seen as a costly feature that can be opted out if performance is desired over flexibility.

Because we use a greedy routing, we do not construct and maintain routing tables and the number of states to maintain in any one peer is only equal to the number of its neighbor peers which does not grow with n thus giving a constant complexity cost being of the order of $O(1)$.

Table I compares the complexity costs of the four above defined metrics of various DHT systems including our solution. For CAN, d is an integer equal to or greater than 2 and thus $0 < 1/d < 1$. The results presented in this table have been gathered by using the data published in [6] and [10] as well as from our previous analysis. Because of the rules pertaining to the O notation, $\log$ functions with different constant bases are considered equivalent.

TABLE I
EXPECTED PERFORMANCE MEASURES OF VARIOUS DHT SYSTEMS

Lookup	Hops	Paths	States	Churn messages
CAN	$O(n^{(1/d)})$	$O(n^{(1/d)}/n)$	$O(1)$	$O(1)$
Chord	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log^2(n))$
Cloak	$O(\log(n))$	$O(\log(n)/n)$	$O(1)$	$O(1)$ / $O(n)$
Kademlia	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log(n))$
Pastry	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log(n))$
Tapestry	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log(n))$
Viceroy	$O(\log(n))$	$O(\log(n)/n)$	$O(\log(n))$	$O(\log(n))$

V. SIMULATIONS

In this section, we present the results of the simulations that we have carried out to assess the practicability of our addressing, routing and binding system based on hyperbolic coordinates. We have used our packet driven discrete event network simulator called *nem* for obtaining the results concerning the evaluation of the dynamic behavior of Cloak. In order to evaluate our overlay system on a realistic topology, we have used a 4k-node IP Internet map created from real data measurements.In all simulations, the first peer creating the overlay is always a randomly picked node of the map. We have considered that only some part of the nodes of a map at any given time are acting as overlay peers. The simulator's engine manages a simulation time and each overlay peer starts at a given time for a given duration on a random node of the map. The peer that creates the overlay remains active for all the duration of a simulation. The packets are delivered between the nodes by taking the transmission time of the links into account. Peers bootstrap by contacting the node that holds the peer that created the overlay, search for other peers to which they can connect, obtain an address from one of the peers they are connected to and send data or requests messages. This process models the birth, life and death of the overlay.

The number of new peers is set to 30 per minute with random inter-arrival times set with a probability following an exponential distribution. Each peer has a random lifetime set with a probability following an exponential distribution with $\lambda = 10e - 5$ which gives a median value of 300 seconds and a 90th percentile value of 1000 seconds. As each dynamic simulation lasts for 1 hour, this distribution of the peers' session lengths produces a lot of churn. The peers create overlay links with other peers by selecting those which are closer in terms of network hops. Finally, we collect measurements every 600 seconds. Each point shown on the following graphs is the average value of 20 runs, and the associated standard deviation values are plotted as error bars.

The frequency of the PUT requests generated by each peer is 1 every 30 seconds. The frequency of the GET requests generated by each peer is 1 every 5 seconds. We do not consider any redundancy parameters for now. Thus a pair is stored on one peer only. As explained in section III, we fix here the *binding tree depth* to 8.

We observe the influence of the addressing tree degree on the performances of the PUT and the GET requests. More precisely we measure the rate of success as well as the average overlay path length of both PUT and GET requests.

Figure 2 shows the percentage of successful PUT requests over the simulation duration. We assume here that only one copy of a binding is stored in the system. We do not yet study the replication strategies explained in section III of this paper. We can see that given the parameters of the simulation, the rate of success is very high despite the churn.

Figure 3 shows the average path length of the PUT requests in the overlay network over the simulation duration. The number of peers to go through including the destination

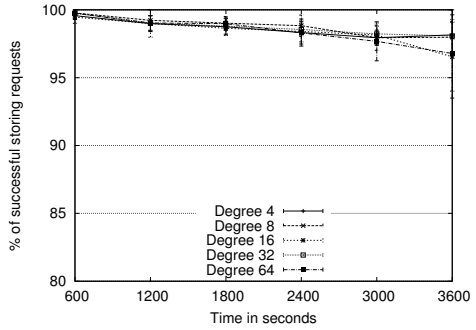

Fig. 2. Percentage of successful PUT requests.

before storing a binding varies from 6 to 9 depending on the addressing tree degree. This number is decreasing when the degree is increasing with a diminishing return effect that can be seen starting at degree 16.

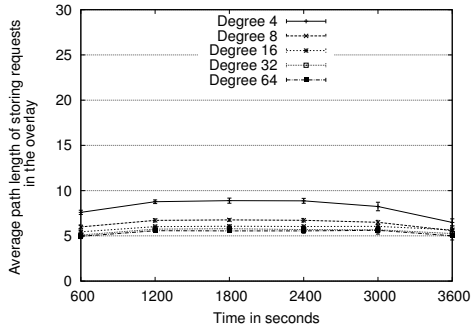

Fig. 3. Average path length of the PUT requests in the overlay network.

Figure 4 shows the percentage of successful GET requests over the simulation duration. As for the PUT request, we can see that given the parameters of the simulation, the rate of success is very high despite the churn.

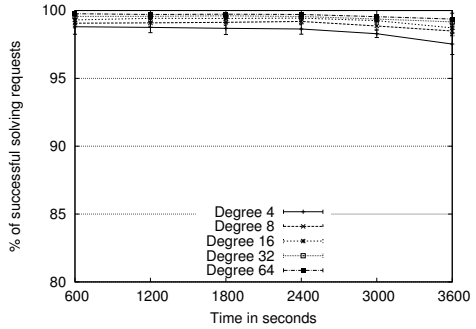

Fig. 4. Percentage of successful GET requests.

Figure 5 shows the average path length of the GET requests in the overlay network over the simulation duration. The number of peers to go through to reach the holder of the binding and including the return trip to the sender of the request varies roughly from 9 to 16 depending on the addressing tree degree. A degree of 4 yields a typical path length of 16, a degree of 8 reduces the path length to 12 and degree values above 8 all yield path lengths between 9 and 10. Thus the overlay hop number is decreasing when the degree is increasing with a diminishing return effect that can be seen starting at degree 16, similarly to the PUTrequests overlay path lengths shown in figure 3.

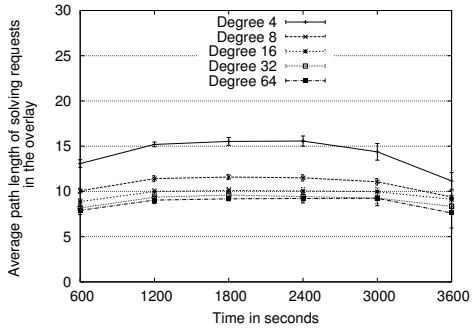

Fig. 5. Average path length of the GET requests in the overlay network.

We can conclude that, for the shown simulation results, our binding system has good performances as a structured DHT system. The rate of success of both the PUT and GET requests, for an overlay running for one hour with a total of 1800 peers, is encouraging. The average path lengths of the requests are also acceptable and show typical values for these kind of systems.

To validate the analysis shown in Section IV, we have also carried out simulations with the *PeerSim* simulator to compare our solution with Chord, Kademlia and Pastry regarding the average number of hops per request. We have used a degree of 24 for Cloak. We have tested various networks sizes ranging from 100 to 12800 by powers of 2 (hence the $\log$ scaled x-axis). We have carried out 50 runs per simulation although this seems not enough when looking at the Chord and Kademlia plots. Figure 6 shows that although the Cloak results are a bit above the other systems, they confirm the same expected $O(\log(n))$ hops per request complexity cost.

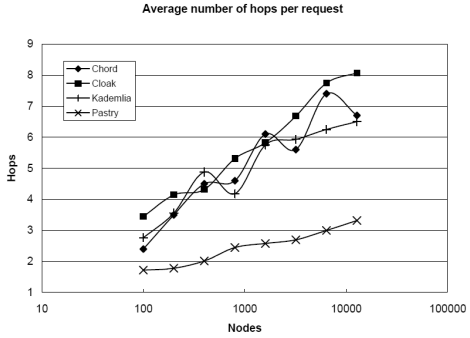

Fig. 6. Average number of hops for various DHT systems.

VI. Related work

Many existing DHTs have been extensively described and we refer the reader to the work of Lua *et al.* [6] for a detailed state of the art of these solutions. Our proposal borrows some elements from these well known DHTs. Our mapping mechanism for placing keys on the unit circle is similar to the one defined by Chord [1]. However, unlike Chord we do not place the peers themselves on this circle but inside the unit disk by using complex coordinates. Similarly to CAN [2], we use a multi-dimensional coordinate space, but instead of using a d-dimensional cartesian multi-torus, we use the 2-dimensional hyperbolic plane $\mathbb{H}^2$. Our greedy routing scheme is based on a properly defined distance metric as done in Kademlia [5]. But unlike Kademlia which is based on the XOR metric, we use the hyperbolic distance defined for the Poincaré disk model of the hyperbolic plane. Another advantage of our greedy routing algorithm as opposed to prefix routing algorithms such as those developed in Pastry [3] and Tapestry [4], is that it does not rely on routing tables. Only the coordinates of the neighbors of a peer are needed to forward a message. This is highly scalable as the peers do not need to build and maintain routing tables. The idea of using the hyperbolic plane as a virtual address space comes from the work of Kleinberg in [7]. We have extended his work by allowing the network to be dynamic and extensible. Indeed, as we setup an overlay network, we are able to fix the degree of the addressing tree to an arbitrary value and as such to avoid the discovery of the highest degree node. This property enables our algorithms to rely only on local information for distributing addresses and routing messages as shown in section II. Also we have defined a mapping function which is novel, whereas Kleinberg suggested using CAN for implementing a DHT based on hyperbolic coordinates. Our binding radius approach presented in section III can cope with the growth and shrink of the overlay automatically thus optimizing the storage of the keys in a dynamically evolving network. More details concerning the scalability and efficiency of our overlay solution based on hyperbolic addressing and routing can be found in our previous work [8].

VII. Conclusion

Providing storage and lookup services to networked applications by using a DHT system is always a challenging task. Because of its tessellation properties, the hyperbolic plane through the use of the Poincaré disk model is well suited for attributing virtual coordinates to the participating nodes of an overlay network. We have shown that given an appropriate mapping function, it is easy to setup and maintain a consistent DHT structure upon such an overlay. We have thus proposed a DHT system called Cloak that provides to all its members a scalable and reliable binding infrastructure. Our analysis has shown that our proposal is scalable with performances similar to other existing DHTs. Our simulation results have demonstrated that the success rate of the PUT and GET requests always remains above 95% in presence of churn. They have also shown that the number of hops per request is very similar to other existing DHTs. Given the fact that overlay nodes can connect to each other with a high degree of freedom, we are confident that our DHT can be an interesting alternative to current existing DHTs. Our future work will consist in implementing our hyperbolic DHT system in a library and comparing its performances to other existing implementations such as Kademlia and Pastry.

References

[1] 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 the ACM SIGCOMM Conference*, 2001, pp. 149–160.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the ACM SIGCOMM Conference*, 2001, pp. 161–172.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001, pp. 329–350.

[4] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 41–53, 2004.

[5] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer-Verlag, 2002, pp. 53–65.

[6] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, pp. 72–93, 2005.

[7] R. Kleinberg, "Geographic routing using hyperbolic space," in *Proc. of the 26th INFOCOM Conference*, 2007, pp. 1902–1909.

[8] C. Cassagnes, T. Tiendrebeogo, D. Bromberg, and D. Magoni, "Overlay addressing and routing system based on hyperbolic geometry," in *Proc. of the IEEE Symp. on Computers and Communications*, 2011.

[9] J. W. Anderson, *Hyperbolic Geometry*. Springer, 2005.

[10] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the butterfly," in *Proc. of the 21st Symposium on Principles of Distributed Computing*, 2002, pp. 183–192.

[11] R. Cohen and S. Havlin, "Scale-free networks are ultrasmall," *Phys. Rev. Lett.*, vol. 90, no. 5, p. 058701, Feb 2003.