VHash: An Optimized Voronoi-Based Distributed Hash Table

***Abstract*—Distributed Hash Tables (DHT) provide a fast and robust decentralized means of key-value storage and retrieval and are typically used in Peer-to-Peer applications. DHTs assign nodes a single identifier derived from the hash of their IP address and port, which results in a random overlay network. A random overlay network is not explicitly optimized for certain metrics, such as latency, trust, energy, or hops on an overlay network. It is often desirable to generate an overlay network that is optimized for one or more specific metrics.**

This paper presents VHash, a DHT protocol to construct an overlay optimized for such metrics. VHash exploits a fast and efficient Delaunay Triangulation heuristic in a geometric space. We used VHash to generate an overlay with edges that minimize latency. While we focused on latency in this paper, VHash optimizes on any defined metrics. This overlay outperformed an overlay generated by the Chord DHT protocol in terms of lookup time. VHash provides a robust, scalable, and efficient distributed lookup service.

I. Introduction

A Distributed Hash Table is used provide an overlay network for many P2P applications. Each node in the overlay network maintains a routing table of a subset of other nodes in the overlay. The configuration and rules of the routing table vary from one protocol to another. In a DHT the entries of the routing tables may be separated by powers of 2 [1], be determined by shared prefixes [2], or be chosen according to a probabilistic distribution [3], but the goal is to minimize the number of overlay hops the distributed lookup needs to make.

A routing table created to minimize overlay hops does not necessarily create routes with minimized latency. What if more information about the nodes could be encoded as part of the overlay, such as the node's latency? Problems like latency require embedding a graph into coordinates in the metric space such as to minimize errors in inter-node distance. If each node is defined as object in this space we can then use a distance function to choose the shortest path over these metrics and approximate a desired behavior, in this case minimum latency.

This opens up two problems: How to we create and maintain a DHT based on an arbitrary metric space? How do we usefully embed inter-node latency and other important routing information into that metric space's coordinate system?

VHash is a DHT designed to take inter-node latency information into account when generating an overlay. Other network metrics could easily be applied to VHash. VHash creates an approximation of a Voronoi network and Delaunay Triangulation to define the routing tables and dictate where content is stored in the network. We accomplish this by assigning each node d coordinates, rather than than a single key. We use a basic spring model to embed inter-node latency graph in the overlay. A result of this is that nodes in VHash can *move* through the metric space over the lifetime of the node; their position is not necessarily fixed. Our paper presents the following:

- We describe the VHash protocol (Section II) and the underlying approximation algorithm of overlay's Delaunay Trinangulation and the corresponding Voronoi diagrams. Our approximation is distributed, greedy, efficient, and accurate in arbitrary number of dimensions and creates an overlay with edges that minimize distance over network metrics while maintaining robustness, scalability, and polylogarithmic lookup time in overlay hops.
- We use VHash to provide us with an overlay for embedding network metrics. We present our basic spring model for embedding nodes in the overlay with latency information and discuss other network metrics that can be used with VHash (Section III).
- We created simulations (Section IV) to prove that the overlays created by VHash are accurate enough for routing messages from arbitrary source nodes to random destination locations. We also show that by embedding the latency graph, our routing dramatically outperforms Chord (and by extension, other overlays with $\lg(n)$-sized routing tables) in terms of latency.
- We compare the VHash protocol to the other protocols that are based off of Voronoi region approximation (Section V).
- We discuss future work that follows from our what plans we have for embedding different problems using VHash (Section VI).

II. VHash

Nodes in the VHash network periodically gossip with other nodes, exchanging information about theirs peers and use the approximation algorithm to refine its list of neighbors. These neighbors approximate the node's Voronoi region and it's corresponding responsibilities. This approximation algorithm is fast and can be used for spaces with an arbitrary number of dimensions.

A. Voronoi Regions in DHTs

A Voronoi diagram is the division of a space into cells or regions along a set of objects O such that all the points in a particular region are closer to one object than any other object. We refer to the region owned by an object as that object's Voronoi region. The Delaunay Triangulation of this same space along the same set of objects is defined by the edges such that no object is inside the circumcircle of any triangle formed by the edges [4]. The Voronoi diagram and Delaunay Triangulation are dual problems, as an edge between two objects in a Delaunay Triangulation exists if and only if

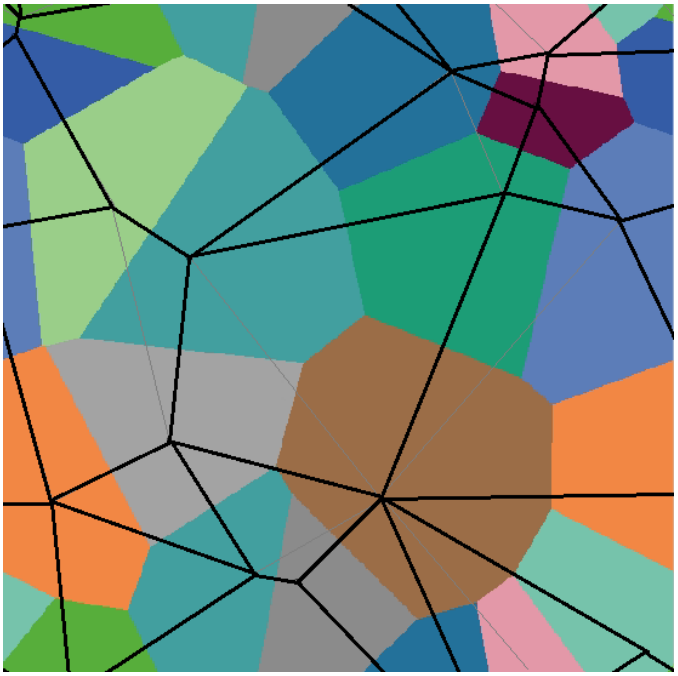

Fig. 1: An example Voronoi diagram for objects on a 2-dimensional toroidal space. The black lines correspond to the edges of the Delaunay Triangulation

those object's Voronoi region border each other. This means that solving either problem will yield the solution to both. An example Voronoi diagram is shown in Figure 1

In our network, the nodes are the objects of the Voronoi diagram and their regions define the regions they are responsible for. The edges created by the Deluanay Triangulation correspond to the connections between neighboring nodes. Computing Voronoi diagrams in a distributed and generalized fashion is prohibitively time expensive. A greedy approximation of the Voronoi regions is sufficient for the VHash protocol. We created a new greedy, online algorithm that approximates and maintains the set of peers defining the node's Voronoi region and Delaunay Triangulation.

A formal and thorough description of Voronoi diagrams as well as their applications can be found in [5].

Arguably all DHTs are built on the concept of Vornoi regions. In all DHT's currently a node is responsible for all points in it's hash space to which it is the "closest" node. These DHTs have carefully chosen metric spaces such that these regions are very simple to calculate. For example Chord and similar ring based DHTs utilize a unidirectional one dimensional ring as their metric space, such that the region for which a node is responsible is the region between it and it's predecessor. VHash generalizes these behaviors: By choosing a particular metric space VHash can approximate other DHTs. The toroidal metric space utilized in this paper is an extension of the ring topology to additional dimensions.

The cost of generalizing VHash to utilize any metric space is the efficiency of calculating Voronoi regions. It is prohibitively expensive to generate an exact calculation of the Deluanay peers and Voronoi regions of a metric space in a distributed fashion [6]. Previous explorations into distributed approximations of node's Voronoi regions (RayNet [6]) offer constant time approximations. We utilize a greedy heuristic that requires a similar computation cost of a single sample from RayNet's Monte-Carlo approximation. As a heuristic, there exist edge cases where it is incorrect. We show in the experimental section that this loss in accuracy has negligible impact on use.

B. Distributed Greedy Voronoi Heuristic

The Distributed Greedy Voronoi Heuristic (Algorithm 1) is a fast method for nodes to select peers from their Deluanay Triangulation. It is predicated on the assumption that the midpoint between two nodes falls on the line segment that they share on the edges of their Voronoi regions. This holds true in the vast majority of cases.

Algorithm 1 Distributed Greedy Voronoi Heuristic

```
candidates is the set of candidate peers obtained from gossiping.
short_peers is the set of this node's one-hop peers
table_size is the minimum size of short_peers
candidates is sorted ascending by each node's distance to this node
The closest member of Candidates is popped and added to short_peers
for all n in Candidates do
    c is the midpoint between this node and n
    if Any node in Peers is closer to c than this node then
        reject n as a peer
    else
        Add n to short_peers
    end if
end for
while |candidates| > 0 do
    for all candidates do
        add candidates to the set of long_peers
        if |long_peers| > table_size^2 then
            long_peers ← random subset of long_peers of size table_size^2
        end if
    end for
end while
```

Each cycle, nodes exchange their peerlists with a current neighbors and then recalculate their neighbors. A node combines the neighbor's peerlists and its own into a list of candidate neighbors. This combined list is sorted from closest to furthest. A new peerlist is then created starting with the first candidate from the list of candidates (which must always be included). The node then examines each of the remaining candidates in the sorted list and calculates the midpoint between the node and the candidate. If any of the nodes in the new peerlist are closer to the midpoint than the candidate, the candidate is set aside. Otherwise the candidate is added to the new peerlist.

This heuristic has the benefit of being fast and scalable into any metric space where a distance function and midpoint

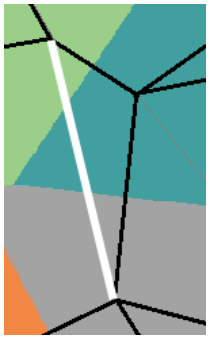

Fig. 2: The white edge is an example of where the Distributed Greedy Voronoi Heuristic can fail to find an edge without peer management preventing this case.

can be defined. The distance metric used for this paper is the minimum distance in a multidimensional toroidal space. Where $\vec{a}$ and $\vec{b}$ are locations in a d-dimensional toroidal space:

$$Distance = \sqrt{\sum_{i \in d} (\min(|\vec{a}_i - \vec{b}_i|, 1.0 - |\vec{a}_i - \vec{b}_i|))^2}$$

This heuristic may fail. Should a node be positioned between two other nodes such that the midpoint between those two nodes does not fall upon the shared face of their Voronoi regions then this heuristic will not link the blocked peers (Figure 2).

This is mitigated by our method of peer management described in subsection D and already present in Algorithm 1.

C. Fast Vornoi Classifier

VHash never actually calculates the polytopes that describe a node's Voronoi region. This is unnecessary and prohibitively expensive [6]. Rather VHash only ever assigns a given point to a Voronoi region. It does this by calculating the distance from that point to all candidate nodes. The point falls into a node's Voronoi region if it is the node to which it has the shortest distance. Thus a node defines it's Voronoi region by keeping a list of the peers that bound it.

D. Peer Management

VHash maintains two peer lists: Short Peers and Long Peers. This is motivated by mitigating the error induced by the Distributed Greedy Voronoi Heuristic and providing robustness during churn[1].

Short Peers are the subset of the Delaunay Peers generated by the Distributed Greedy Voronoi Heuristic. The number of short peers has no upper bound and in contrived cases, such as a single node surrounded by other nodes forming a hypersphere, it can grow quite high. Previous work as found a useful lower bound on peers to be $3d + 1$[6] Using a lower bound on the length of the short peer list corrects for errors in the approximation processes by including peers that otherwise would not be. An approximation of the number of nodes in the network is sufficient to generate an accurate expected number of short peers. This expected number of short peers is used as the lower bound for the size of short peer list. Should the number of short peers generated by the Distributed Greedy Voronoi Heuristic be less than the lower bound for that set's size, the nearest peers not already included in the short peer list are added to the short peer list until it is of sufficient size. Short peers are analogous to the predecessors/successors in other DHTs.

Long Peers correspond to the two hop peers of the node. When a node learns about potential neighbors, but are not included in the short peer list, they may be included in the long peer list. The long peer list has a maximum size of the base length of the short peer list squared. For example, if the short peer list has a minimum size of 8, the long peer list has a maximum size of 64 entries. Long peers are not actively probed during maintenance and the cost of managing them is minimal.

We discuss how nodes learn about short and long peers in the Gossiping subsection.

E. Maintenance via Gossiping

Each node in the network performs maintenance periodically by 'gossiping' with a randomly chosen neighbor. When two nodes gossip with each other, they exchange their short peer lists with each other. The node combines the lists of short peers[2] and uses the Distributed Greedy Voronoi Heuristic to determine which of these candidates correspond to its neighbors along the Delaunay Triangulation. The candidates determined not to be short peers become long peers. If resulting number of long peers exceeds the maximum size of the long peer list, a random subset of the maximum size is kept.

The formal algorithm for this process is described in 2. This maintenance through gossip process is very similar to the gossip protocol used in [6].

Algorithm 2 Gossip Process

1: Node n initiates the gossip.
2: $neighbor \leftarrow$ random node from $n.short\_peers$
3: $candidates \leftarrow n.short\_peers \cup neighbor.short\_peers$.
4: n and $neighbor$ each run Distributed Greedy Voronoi Heuristic using their own copy of $candidates$

F. Handling Churn

Churn is effects of the continuous joining and exiting of nodes in the overlay. DHTs must have some mechanisms to handle this process to maintain fault tolerance and routing.

Joining the network is a straightforward process. A prospective node must be able to communicate with at least one patron member of the DHT. The prospective node is assigned a location using the method described in Section III and uses the patron it knows to find the node responsible for it's assigned

[1]The disruption caused to the overlay by the continuous joining, leaving, and failing of nodes.

[2]Naturally, nodes remove themselves and repetitions from the candidates.

location, which we call the parent. The prospective node sets the parent node as the lone member of the short peers and immediately gossips with the parent. Subsequent gossips will refine the peer lists of the nodes affected by the join.

There is no "polite" exit from the network. VHash assumes nodes will fail and the distinction between an intended failure and unintended failure is unnecessary. Should a node wish to crease participating in the network, it need only to crease sending and receiving messages. Suppose a node f fails or leaves the network; we assume it does so abruptly and without warning. When one of f's neighbors attempts to contact f for gossiping or routing, failure to communicate with f will prompt the neighbor to remove f from its peers. The node the selects a different neighbor to gossip with or recomputes the peer closest to the location it was looking for. Should a short peer fail, routing around its failure is trivial due to knowledge about the long peers.

G. Routing

The proper forwarding peer for routing a message extends from the voronoi regions of the short peers and long peers. Rather than attempt to know the true voronoi region of those peers we approximate the voronoi regions of those peers as if there where no other nodes in the network. The resulting voronoi regions describe both the subset of the network for which that peer is responsible and the subset of the network for which it is the most efficient forwarding node. The routing node determines the voronoi region into which the message's destination falls. If this is itself, it handles the message accordingly; otherwise the routing node forwards the message to the responsible node. This process is equivalent to a pre-computed and cached efficient routing algorithm (Algorrihm 3). Even if our approximation for the Voronoi region of a peer is incorrect due to incomplete knowledge of the network, our approximation describes the most efficient forwarding path of a message destined for that space available.

Algorithm 3 Vhash Lookup

1: n is this node
2: m is a message addressed for loc
3: $potential\_dests \leftarrow n \cup n.short\_peers$
4: $c \leftarrow$ node in $potential\_dests$ with shortest distance to loc
5: **if** $c == n$ **then**
6: **return** n
7: **else**
8: **return** c.lookup(loc)
9: **end if**

H. Algorithm Analysis

The Distributed Greedy Voronoi Heuristic is very efficient in both terms of space and time. Suppose a node is creating its short peer list from k candidates. The candidates must be sorted, which takes $O(k \cdot \lg(k))$ operations. Then for each candidate, that the node must compute the midpoint between itself and the current candidate and then compare distances to that point between itself and all of the peers it has found so far. This comes down to a cost of

$k \cdot \lg(k) + k$ midpoint computations$+k^2$ distance computations

Since k is bounded by $\Theta(\frac{\log N}{\log \log N})$ [7] (the maximum number of peers stored neighbors allowed between two neighbors), we can translate the above to

$$O(\frac{\log^2 N}{\log^2 \log N})$$

Because in the vast majority of cases, the number of peers is equal to the constant minimum table size, which gives $k = O(1)$ in the expected case, the lower bound and expected complexities are $\Omega(1)$.

While previous work [6] claims constant time approximation, the reality is that Raynet's leading constant is in the order of thousands as Monte-Carlo samples. While our algorithm has a greater asymptotic worst case cost, for all current realistic network sizes it will be more time efficent then raynet's approximation.

III. Network Metric Embedding

During previous research in applications of DHTs to parallel processing [**?**] we tested our network's behavior under churn. We came to the unexpected conclusion that high levels of churn improved the performance of the system rather than being detrimental under the right circumstances. The built-in mechanisms for managing responsibility in a DHT are very effective and efficient at handling servers changing locations in the network. We found in this previous work that it was more effective to move the server to a location in the DHT where work or data was stored rather than attempt to remap data efficiently over the servers. VHash is designed to leverage this discovery and allow a server's location in the network to provide meaningful information about the server and to allow servers to change location in the network to suit the networks needs. In this paper we propose a method that peers can use to periodically change their location in the network such that routes taken on the resulting overlay network have lower latency than would otherwise possible in a DHT.[3]

We utilize a distributed graph embedding algorithm to find locations in the coordinate system for each node such that the latency between nodes can be accurately represented. Once inter-node latency is effectively represented as distance in the coordinate space, routing for shortest distance across this space also routes along the shortest latency path on the overlay network.

For our simulations we measure latency in terms of hops on a large scale free graph (10,000 nodes) that acts as an underlying network. We simulate a small fraction of these nodes to act as members of a DHT and simulate VHash's and Chord's distributions of latency for recursive lookups. Each node in the VHash DHT preforms a periodic update step to it's location to accurately represent inter-node latency on

[3]Is this better suited for Section I?

the coordinate system. We utilize a toroidal metric space for these experiments as generalization of Chord's ring topology. Other research shows we have potential for even more effective results on a hyperbolic plane based coordinate system [8].

To preform the embedding we designed an algorithm based on force-directed graph drawing [9] for servers to utilize and periodically update their locations to lower the latency with their peers. The approximate distributed embedding algorithm utilizes VHash's maintenance and join behaviors. This algorithm is described in Algorithm 4. Each node periodically measures the latencies to it's peers and generates a change in position that minimizes the error in distance to it's peers. Once a node finds peers of low latency with it, these error approximations become small and the network configuration becomes stable. If a node has high latency with it's current peers the high latency connection causes the nodes to repel each other and move to new peers.

Algorithm 4 Decentralized Peer-to-Peer Spring Model

n is a node at location $\overrightarrow{loc}$
$total\_dist \leftarrow \sum distance(n,p), \forall p \in n.short\_peers$
$total\_lat \leftarrow \sum latency(n,p), \forall p \in n.short\_peers$
$unit\_lat \leftarrow total\_dist \div total\_lat$
for all $p \in n.short\_peers$ **do**
  $ideal\_distance \leftarrow latency(n,p) \div unit\_lat$
  $error\_distance \leftarrow ideal\_distance - dist(n,P)$
  $\overrightarrow{loc} \leftarrow \overrightarrow{loc} + error\_distance \cdot \overrightarrow{Unit\_vector(n,p)}$
end for

IV. Simulations

We implemented two simulations to prove that VHash acts as as a stable DHT and is able to minimize the latency on lookup time. Our first simulation was of successful lookup rates (hitrate) over time as a VHash overlay converged on the proper topology from a random graph. This simulated a large network bootstrapping itself from a series of essentially random connections and into an ordered DHT.

Our other simulation compared the distributions of latency in both the VHash and Chord DHTs to examine VHash's capability to minimize latency. We wanted to demonstrate the benefits to latency that followed from optimizing VHash's overlay topology for minimum latency according to our spring model (Algorithm 4).

A. Convergence Simulation

This simulation demonstrates that the VHash protocol converges to a stable overlay from an chaotic starting topology after a sufficient number of gossip cycles. We do this by showing that rate of successful lookups converges to 1.0. We compare these results to RayNet [6], which proposed that a random k-connected graph would be a good, challenging starting configuration for demonstrating convergence of a DHT to a stable network topology.

During the first two cycles of the simulation, each node bootstraps its short peer list by appending 10 randomly selected nodes. In all the cycles, the nodes perform a gossip with a random node from their short peers to recompute both their peer lists. We then calculate the hitrate of successful lookups by simulating 2000 lookups from random nodes to random locations, as described in Algorithm 5.

Our experimental variables for this simulation were the number of nodes in VHash overlay and the number of dimensions. We tested network sizes of 500, 1000, 2000, 5000, and 10000 nodes each in 2, 3, 4, and 5 dimensions. The hitrate at each cycle is $\frac{hits}{2000}$, where hits are the number of successful lookups.

Algorithm 5 Routing Simulation Sample

$start \leftarrow$ random node
$dest \leftarrow$ random set of coordinates
$ans \leftarrow$ node closest to $dest$
if $ans == start.lookup(dest)$ **then**
  increment $hits$
end if

Our results are shown in Figures 3, 4, 5, and 6. Our graphs show that VHash's overlay quickly constructs itself from a random configuration and that our hitrate approached 90% by cycle 20, regardless of dimension. VHash consistently converged to a hitrate of 1.0 by cycle 30. In comparison, Raynet's routing converged to a perfect hitrate at around cycle 30 to 35 [6] As the network size and number of dimensions each increase, convergence slows, but not to a significant degree.

B. Latency Distribution Test

The goal of our second set of experiments is to demonstrate VHash's ability to optimize a selected network metric: latency in this case. We compared VHash's performance to that of a more traditional DHT, Chord [1]. Chord is a well established DHT with an $O(log(n))$ sized routing table and $O(log(n))$ lookup time measured in overlay hops. Rather than examine the number of hops on the overlay network as our primary metric as done most other analyses of lookup time [1] [2] [6] [10] [11], we are concerned with the actual latency lookups experience traveling through the *underlay* network, the network the overlay is built upon.

Overlay hops are used in most DHT papers as the primary measure of latency under the rationale that as the number of nodes increases towards maximum network capacity, the majority of nodes will have average latency with the majority of other nodes because the nodes will be more uniformly distributed. This rationale is only valid where it's assumption of massive and diverse networks is held. For most realistic network sizes and structures, there is dramatic room for latency reduction in DHTs.

For this experiment we constructed a 10000 node random scale free network (which has an approximate diameter of 3 hops) to act as an underlay [12] [13] [14]. We used a scale-free network as the underlay as it is a simplified model of the Internet's topology [12] [13]. From this underlay, we chose a random subset to be members of the overlay network. We then measured the distance in underlay hops between 10000

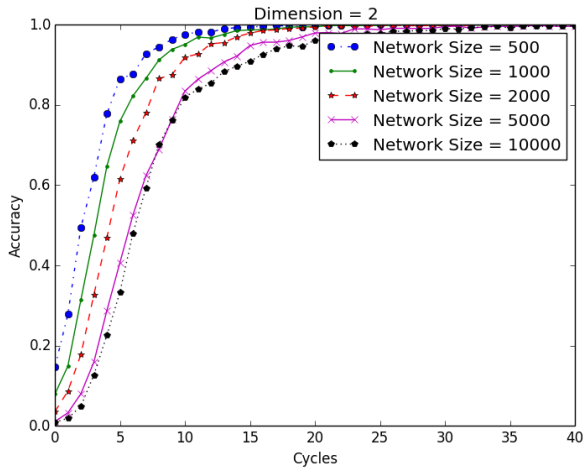

Fig. 3

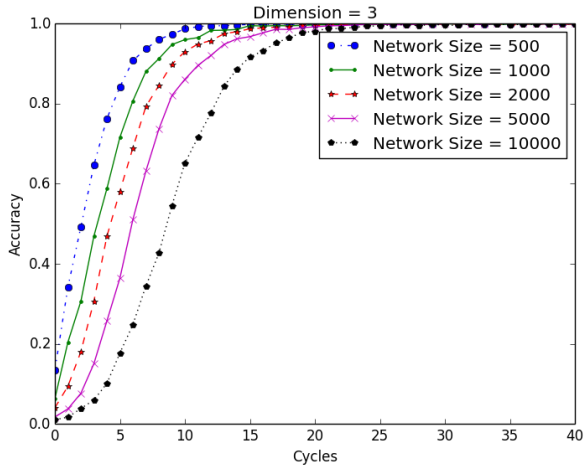

Fig. 4

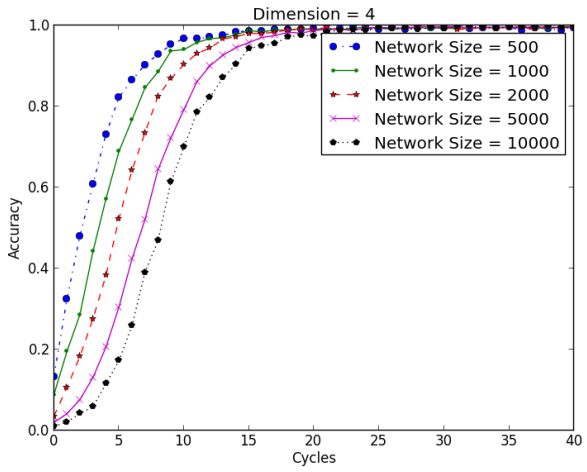

Fig. 5

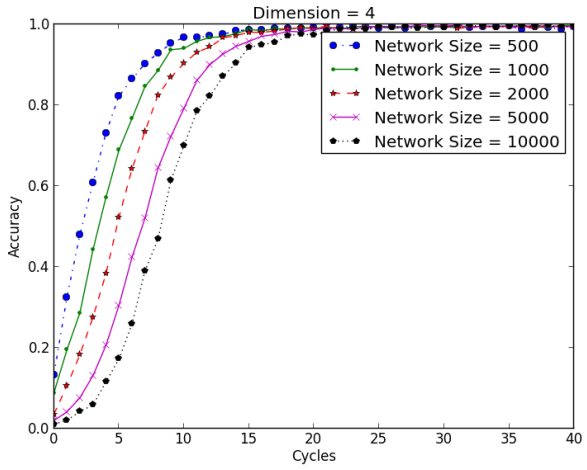

Fig. 6

random source-destination nodes from the overlay. VHash generates an embedding of the latency graph utilizing the distributed spring model algorithm described in Algorithm 4, with the latency function is defined as the number of underlay hops between it and its peers.

Our simulation created 100, 500, and 1000 node overlays for both VHash and Chord. We used 4 dimensions in VHash and a 160 bit identifier for Chord[4].

Figures 8, 9, and 10 show the distribution of path lengths measured in underlay hops in both Chord and VHash. In all three network sizes, VHash dramatically outperformed Chord and minimized the underlay path lengths. Besides having a much lower average path length, the variance was also significantly lower.

For comparison, we also sampled the lookup length in overlay hops for a 1000 sized Chord and VHash network. As seen in Figure 7, the paths in VHash's overlay were significantly shorter than those in Chord.

In comparing the overlay and underlay hops, we find that for each overlay hop in Chord the lookup must travel 2.719 underlay hops on average; in VHash, lookups must travel 2.291 underlay hops on average for every overlay hop traversed. This shows that the latency embedding process has seem some measurable benefit.

[4]Using 160 bits for keys and identifiers is standard for Chord.

V. Related Work

While there has been previous work on applying Voronoi regions to DHTs and P2P applications, we have found no prior work on how to perform embedding of an inter-node latency graph.

Backhaus et. al's VAST [15] is a Voronoi-based P2P protocol designed for handling event messages in a massively multiplayer online videogame. Each node finds its neighbors by constructing a Voronoi diagram using Fortune's sweepline algorithm [16]. VAST demonstrated that Voronoi diagrams could be used as the backbone large-scale, although their work focused specifically on using 2-dimensional Voronoi diagrams. VHash approximates the Voronoi region rather than solving it, as higher dimension Voronoi regions are computationally

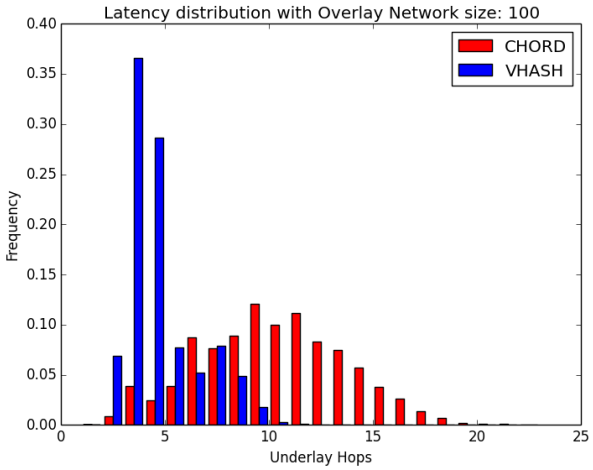

Fig. 7: Distribution of path lengths for Chord and VHash in a size 100 overlay.

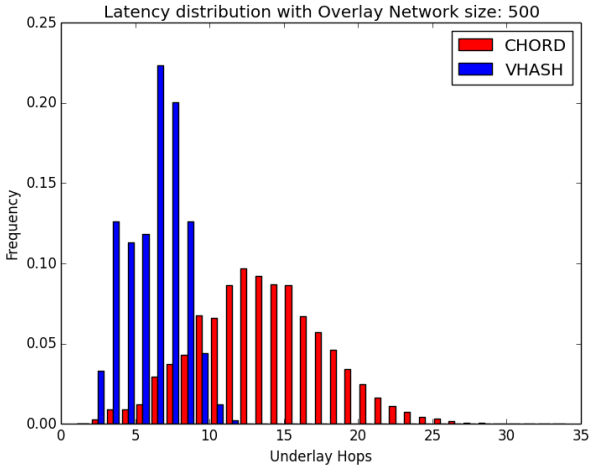

Fig. 8: Distribution of path lengths for Chord and VHash in a size 500 overlay.

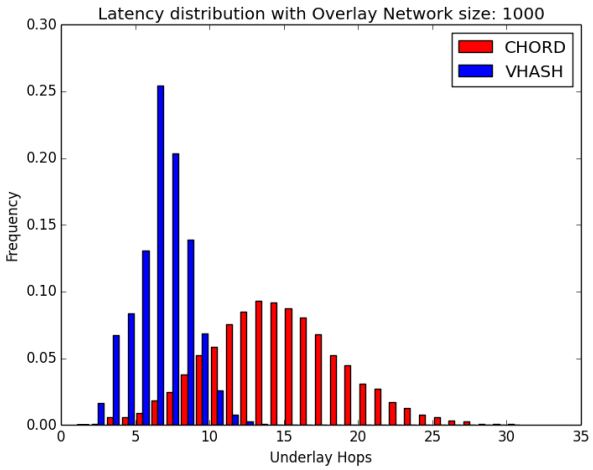

Fig. 9: Distribution of path lengths for Chord and VHash in a size 1000 overlay.

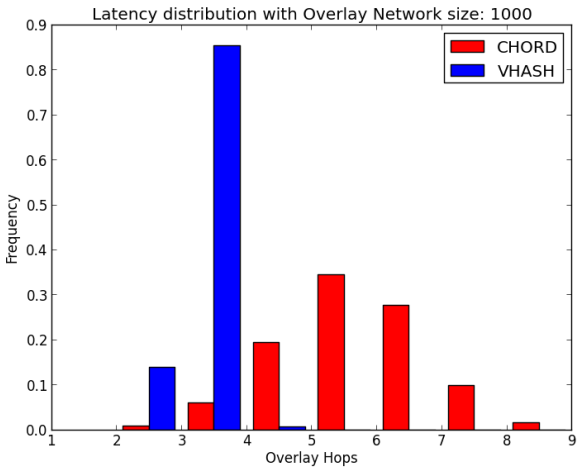

Fig. 10: Comparison of Chord and VHash in terms of overlay hops. Each overlay has 1000 nodes.

expensive to solve. In addition, VHash is designed to exploit network metrics.

The two DHT protocols developed by Beumont et al., VoroNet [11] and RayNet [6] are the closest comparisons to VHash. VoroNet is based off Kleinberg's small world model [3] and achieves polylogarithmic lookup time. Each node in Voronet solves its Voronoi region to determine its neighbors and also maintains a link to a randomly chosen distant node. Voronet focused specifically on the two-dimensional Vornoi computations and the techniques used would be too expensive in higher dimensions and were not resilient to churn [6].

RayNet [6] was based on the work done on Voronet and is by far the most similar to VHash. Like VHash, RayNet does not solve for Voronoi regions, as that is prohibitively expensive. Unlike VHash, RayNet uses a Monte-Carlo method to approximate the volume of a node's Voronoi region. While effective and estimating the Voronoi region, the volume-based Monte-Carlo approximation is expensive and requires multiple samples. RayNet does mention the idea of mapping attributes to each axis, but how this can be exploited is left as future work.

VI. Future Work

Our experiments show that VHash can significantly reduce latency in a network compared to traditional DHTs by using the latency spring model. VHash can be applied to not just latency, but other network metrics as well. Much of our future work will look at applying VHash to network metrics such as energy or transmission range. We also plan on making adjustments to the spring model itself; the model is a basic prototype to demonstrate that latency minimization works. Other work has demonstrated that it may be possible to embed scale free networks into hyperbolic planes [8]. This would allow for construction of optimally minimal latency networks in VHash.

Another venue for exploration is application of caching and replication strategies to a functional distributed file system running on top of VHash.

REFERENCES

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCO*, no. 4, pp. 149–160, ACM, 2001.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*, pp. 329–350, Springer, 2001.

[3] J. M. Kleinberg, "Navigation in a small world," *Nature*, vol. 406, no. 6798, pp. 845–845, 2000.

[4] "Geometric algorithms." http://www.cs.princeton.edu/~rs/AlgsDS07/16Geometric.pdf.

[5] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[6] O. Beaumont, A.-M. Kermarrec, and É. Rivière, "Peer to peer multidimensional overlays: Approximating complex structures," in *Principles of Distributed Systems*, pp. 315–328, Springer, 2007.

[7] M. Bern, D. Eppstein, and F. Yao, "The expected extremes in a delaunay triangulation," *International Journal of Computational Geometry & Applications*, vol. 1, no. 01, pp. 79–91, 1991.

[8] F. Papadopoulos, D. Krioukov, M. Bogua, and A. Vahdat, "Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[9] S. G. Kobourov, "Spring embedders and force directed graph drawing algorithms," *CoRR*, vol. abs/1201.3011, 2012.

[10] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*, pp. 53–65, Springer, 2002.

[11] O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Rivière, "Voronet: A scalable object network based on voronoi tessellations," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–10, IEEE, 2007.

[12] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin, "Resilience of the internet to random breakdowns," *Physical review letters*, vol. 85, no. 21, p. 4626, 2000.

[13] R. Pastor-Satorras and A. Vespignani, "Epidemic spreading in scale-free networks," *Physical review letters*, vol. 86, no. 14, p. 3200, 2001.

[14] A. Hagberg, D. Schult, P. Swart, D. Conway, L. Séguin-Charbonneau, C. Ellison, B. Edwards, and J. Torrents, "Networkx. high productivity software for complex networks," *Webová strá nka https://networkx. lanl. gov/wiki*, 2004.

[15] H. Backhaus and S. Krause, "Voronoi-based adaptive scalable transfer revisited: Gain and loss of a voronoi-based peer-to-peer approach for mmog," in *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '07, (New York, NY, USA), pp. 49–54, ACM, 2007.

[16] S. Fortune, "A sweepline algorithm for voronoi diagrams," *Algorithmica*, vol. 2, no. 1-4, pp. 153–174, 1987.