A Distributed Greedy Heuristic for Computing Voronoi Tessellations With Applications Towards Peer-to-Peer Networks

Brendan Benshoof Andrew Rosen Anu G. Bourgeois Robert W. Harrison
Department of Computer Science, Georgia State University
bbenshoof@cs.gsu.edu rosen@cs.gsu.edu anu@cs.gsu.edu rharrison@cs.gsu.edu

***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, energy, or hops on an overlay network. Being able to do so will allow for tighter control over the network behavior and enable higher performance DHT applications.**

This paper presents VHash, a DHT protocol to construct an overlay optimized for such metrics. VHash exploits a fast and efficient Delaunay Triangulation heuristic to approximate Voronoi regions 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 approach outperforms overlays generated by the Chord DHT protocol in terms of lookup time. VHash provides a robust, scalable, and efficient distributed lookup service.

I. Introduction

Voronoi diagrams [1] have been used in distributed and peer-to-peer (P2P) applications for some time. They have a wide variety of applications. Voronoi diagrams can be used as part of distributed hash tables[2]. They can be used in coverage detection for wireless networks [3]. Massively Multiplayer Online games (MMOs) can use them to distribute game states and events between players at a large scale [4] [5] [6].

Computing the Voronoi tessellation along with its coprime problem, Delaunay Triangulation, is a well analyzed problem.[1] There are many algorithms to efficiently compute a Voronoi tessellation given all the points on a plane, such as Fortune's sweep line algorithm[7]. However, many network applications are distributed and many of the algorithms to compute Voronoi tessellations are unsuited to a distributed environment.

In addition, trouble occurs when points are located in spaces with more than two dimensions. Computing the Voronoi tessellation of n points in a space with d dimensions takes $O(n^{\frac{2d-1}{d}})$ time [8]. Distributed computations often have to resort to costly Monte-Carlo calculations [9] in order to handle more than two dimensions.

Rather than exactly solving the Voronoi tessellation, we instead a fast and accurate heuristic to approximate each of the regions of a Voronoi tessellation This enables fast and efficient formation of P2P networks. A P2P network built using this heuristic would be able to take advantage of it's available fault-tolerant architecture to route along any inaccuracies that arise. Our paper presents the following contributions:

- We present our Distributed Greedy Voronoi Heuristic (DGVH). The DGVH is a fast, distributed, and highly accurate method whereby nodes calculate their individual regions described by a Voronoi tessellation using the positions of nearby nodes. DGVH can work in an arbitrary number of dimensions and can handle non-euclidean distance metrics. Our heuristic can also handle toroidal spaces. In addition, DGVH can accommodate the calculation of nodes moving their positions and adjust their region accordingly, while maintaining a high degree of accuracy (Section **??**). Even where small inaccuracies exist, DVGH will create a fully connected graph.
- We discuss what P2P and distributed applications can use DGVH and how. In particular, we show how we can use DGVH to build a distributed hash table with embedded minimal latency.
- We present simulations demonstrating DGVH's efficacy in quickly converging to the correct Voronoi tessellation. We simulated our heuristic in networks

[1] More citations

ranging between size 500 to 10000. Our simulations show that DGVH accuracy reaches 90% within 20 cycles and converges near 100% accuracy by cycle 30.

- We present the previous work we have built upon to create our heuristic and what improvements we made with DGVH.

II. Distributed Greedy Voronoi Heuristic

A. The Heuristic

The Distributed Greedy Voronoi Heuristic (DGVH) is a fast method for nodes to define their individual Voronoi region (Algorithm 1). This is done by selecting the nearby nodes that would correspond to the points connected to it by a Delaunay triangulation. The rationale for this heuristic is that, in the majority of cases, the midpoint between to nodes falls on the common boundary of their Voronoi regions.

Each cycle, nodes exchange their peer lists with a current neighbor and then recalculate their neighbors. A node combines their neighbor's peer list with its own to create a list of candidate neighbors. This combined list is sorted from closest to furthest. A new peer list is then created starting with the closest candidate. 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 peer list are closer to the midpoint than the candidate, the candidate is set aside. Otherwise the candidate is added to the new peer list.

The following paragraphs may need reordering: DGVH never actually solves for the actual polytopes that describe a node's Voronoi region. This is unnecessary and prohibitively expensive [9]. Rather, once the heuristic has been run, nodes can determine whether a given point would fall in it's region.

Nodes do this by calculating the distance of the given point to itself and othe nodes it knows about. The point falls into a particular node's Voronoi region if it is the node to which it has the shortest distance. This process continues recursively until a node determines that itself to be the closest node to the point. Thus, a node defines its Voronoi region by keeping a list of the peers that bound it.

This heuristic has the benefit of being fast and scalable into any geometric space where a distance function and midpoint can be defined. The distance metric used for this paper is the minimum distance in a multidimensional unit toroidal space. Where $\vec{a}$ and $\vec{b}$ are locations in a d-dimensional unit toroidal space:

$$distance = \sqrt{\sum_{i \in d} (\min(|\vec{a}_i - \vec{b}_i|, 1 - |\vec{a}_i - \vec{b}_i|))^2}$$

In two-dimensional spaces, there is quick hack to achieve 100% accuracy distributed.

Our heuristic can be overaggressive in removing candidate nodes. For example, if a node is located between two other nodes, such that their midpoint does not fall upon the shared face of their Voronoi regions, then this heuristic will not link the blocked peers. This is demonstrated in Figure 1. Our algorithm handles these cases via our method of peer management (Section II-B).

Algorithm 1 Distributed Greedy Voronoi Heuristic

```
Given node n and its list of candidates.
Given the minimum table_size
short_peers ← empty set that will contain n's one-hop peers
long_peers ← empty set that will contain n's two-hop peers
Sort candidates in ascending order by each node's distance to n
Remove the first member of candidates and add it to short_peers
for all c in candidates do
    m is the midpoint between n and c
    if Any node in short_peers is closer to m than n then
        Reject c as a peer
    else
        Remove c from candidates
        Add c to short_peers
    end if
end for
while |short_peers| < table_size and |candidates| > 0 do
    Remove the first entry c from candidates
    Add c to short_peers
end while
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
```

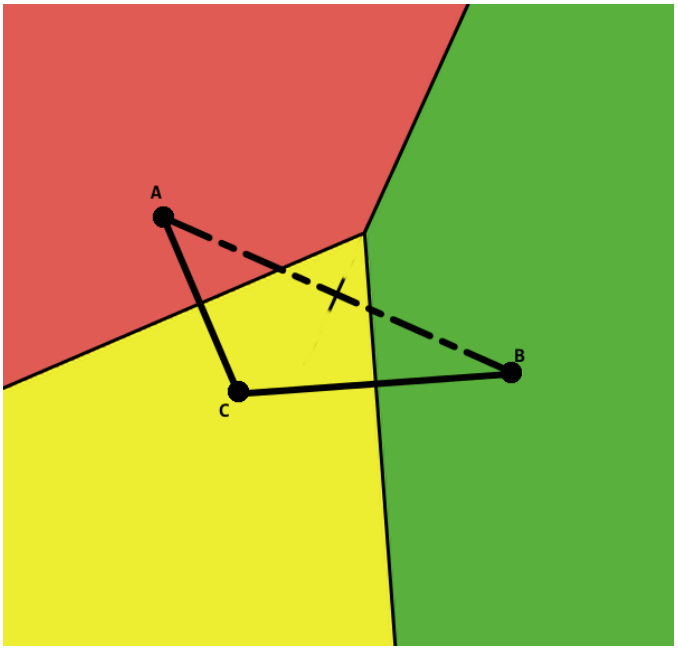


Fig. 1: The edge between A and B is not detected by DGVH, as node C is closer to the midpoint than B is. This is mitigated by VHash's peer management polices.

B. Peer Management

Nodes running the heuristic maintain two peer lists: *Short Peers* and *Long Peers*. This is done to mitigate the error induced by DGVH and providing robustness against churn[2] in a distributed system.

Short Peers are the set of peers DGVH judged to have Voronoi regions adjacent to the node's own. Using a lower bound on the length of *Short Peers* corrects for errors in the approximation as it force nodes to include peers in that would otherwise be omitted. Previous work by Beaumont *et al*. [9] has found a useful lower bound on short peers to be $3d+1$. Should the number of short peers generated by DGVH be less than the lower bound, the nearest peers not already included in *Short Peers* are added to it, until *Short Peers* is of sufficient size.

There is no upper bound to the number of short peers a node can have. This means in contrived cases, such as a single node surrounded by other nodes forming a hypersphere, this number can grow quite high. Bern *et al.* [18] found that the expected maximum degree of a vertex in a Delaunay Triangulation is

$$\Theta(\frac{\log n}{\log \log n})$$

where n is the number of nodes in the Delaunay Triangulation. This bound applies to a Delaunay Triangulation in any number of dimensions. Thus, the maximum expected size of *Short Peers* is bounded by $\Theta(\frac{\log n}{\log \log n})$, which is a highly desirable number in many distributed systems [12] [11].

Long Peers is the list of two-hop neighbors 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 $(3d+1)^2$. For example, if the short peer list has a minimum size of 8, the long peer list has a maximum size of 64 entries. We recommend that members of *Long Peers* are not actively probed during maintenance to minimize the cost of maintanence.

How nodes learn about peers is up to the application. We experimented using a gossip protocol, whereby a node selects peer from *Short Peers* at random to "gossip" with. When two nodes gossip with each other, they exchange their *Short Peers* with each other. The node combines the lists of short peers[3] and uses DGVH 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 Algorithm 3. This maintenance through gossip process is very similar to the gossip protocol used in [9].

Algorithm 2 Gossiping

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

C. Algorithm Analysis

DVGH is very efficient in both terms of space and time. Suppose a node n is creating its short peer list from k candidates in an overlay network of N nodes. The candidates must be sorted, which takes $O(k \cdot \lg(k))$

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

[3]Nodes remove themselves and repetitions from the candidates they receive.

operations. Node n must then compute the midpoint between itself and each of the k candidates. Node n then compares distances to the midpoints between itself and all the candidates. This results in a cost of

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

Since k is bounded by $\Theta(\frac{\log N}{\log \log N})$ [18] (the expected maximum degree of a node?), we can translate the above to

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

In the vast majority of cases, the number of peers is equal to the constant minimum table size. This yields $k = (3d+1)^2 + 3d + 1$ in the expected case, where the lower bound and expected complexities are $\Omega(1)$.

Previous work [9] claims constant time approximation. The reality is that Raynet's leading constant is in the order of thousands. Our algorithm has a greater asymptotic worst case cost, but for all current realistic network sizes it will be more time efficient then RayNet's approximation.

III. Applications

IV. Related Work

Unlike ACE, we handle unbounded simultaneous join and leave operations.

V. Conclusion

VI. Introduction

A Distributed Hash Table (DHT) protocol is used to provide an overlay network for many P2P applications. A DHT protocol allows peers to self organize and allocate responsibility for file distribution between each other [10] [11] [12] [13]. Each peer in the network maintains a routing table of other peers in the overlay. The configuration and rules of the routing table vary from one protocol to another. However, they all have the common goal to minimize the number of overlay hops that a distributed lookup requires. A routing table created to minimize overlay hops does not necessarily create routes with minimized overall network latency.

We have designed a DHT protocol, VHash, that enables more sophisticated applications of DHTs to networking problems. This paper shows that VHash generates an overlay network and routing method that not only reduces overlay hops, but also reduces actual latency across the underlying network as compared to existing DHTs.

DHTs are built on the premise of assigning locations in a geometric space to peers and files. Peers are assigned areas in this space and are responsible for files and maintain connections with other peers accordingly. Reducing latency in a DHT requires embedding a graph of inter-peer latency into such a geometric space. This allows for efficient routing in the geometric space model and yields efficient routing on the network.

This requires solutions for two problems: How do we create and maintain a DHT based on an arbitrary geometric space? How do we usefully embed inter-node latency into the geometric space's coordinate system?

VHash provides a solution to both of these problems. VHash works in arbitrary geometric spaces by creating an approximate Voronoi partition and Delaunay Triangulation based on peer locations. The Delaunay Triangulation defines the routing tables and the Voronoi partition dictates where content is stored in the network. We accomplish this by assigning each node d coordinates, rather than a single key. We use a basic force directed model to embed the inter-node latency graph in the overlay and assign peer locations to reduce latency. A result of this is that peers in VHash can *move* through the metric space over the lifetime of the peer; their position is not necessarily fixed and are expected to move as the network latency behavior changes.

Our paper presents the following:

- We describe the VHash protocol (Section VII) and the underlying approximation algorithm for Delaunay Triangulation and Voronoi regions. Our approximation is distributed, greedy, efficient, and accurate in an arbitrary number of dimensions. It 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 force directed graph model for embedding nodes in the overlay with latency information (Section VIII).
- We present simulations (Section IX) to demonstrate that the overlays created by VHash enable routing messages from arbitrary source nodes to random destination locations efficiently. We also show that by embedding the latency graph, our routing dramatically outperforms DHTs that rely on $O(\lg(n))$ size routing tables, such as Chord [12].
- We present related work upon which VHash is built and describe the improvements provided by VHash (Section X).

- We discuss future work, including how VHash is capable of optimizing other network metrics beyond latency (Section XI).

VII. VHash

This section presents the VHash DHT protocol to generate an overlay network topology. VHash differentiates from other DHTs primarily in its method of peer selection. Nodes in the VHash network periodically gossip with other nodes, exchanging locations of theirs peers and use an approximation algorithm to refine its list of neighbors. These neighbors approximate the node's Voronoi region and its corresponding responsibilities. A node is responsible for all data that is assigned to locations within its Voronoi region. This approximation algorithm (Algorithm **??**) is fast and can be used in spaces with an arbitrary number of dimensions.

A. Voronoi Regions in DHTs

A Voronoi diagram is the partition 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 Voronoi diagram and Delaunay Triangulation are dual problems, as an edge between two objects in a Delaunay Triangulation exists if and only if those object's Voronoi regions border each other. This means that solving either problem will yield the solution to both. An example Voronoi diagram is shown in Figure 2. A formal and thorough description of Voronoi diagrams as well as their applications can be found in [14].

In our network, the nodes are the objects of the Voronoi diagram and are responsible for keys that fall in their region. The edges created by the Delaunay Triangulation correspond to the connections between neighboring nodes. Computing Voronoi diagrams in a distributed and generalized fashion is prohibitively time expensive and previous work [9] shows that an approximation is sufficient for peer selection and routing. With VHash, we have created a new greedy, online algorithm that approximates and maintains the set of peers defining the node's Voronoi region and Delaunay Triangulation.

Arguably all DHTs are built on the concept of Voronoi regions. In all DHTs a node is responsible for all points in its 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 itself and its predecessor. VHash generalizes these same 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 used by Chord [12], Symphony [15], and others into additional dimensions.

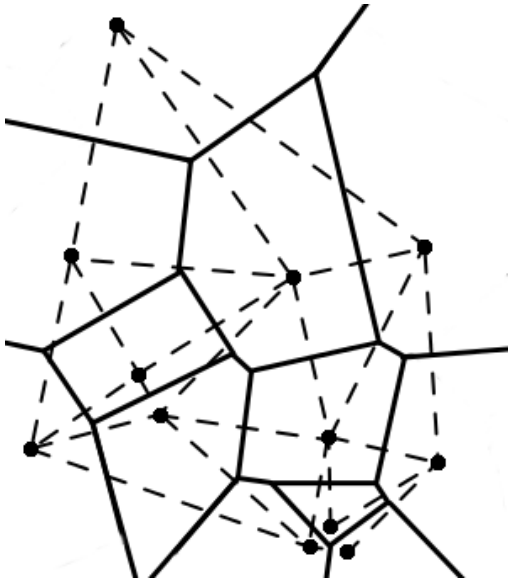


Fig. 2: An example Voronoi diagram for objects on a 2-dimensional space. The black lines correspond to the borders of the Voronoi region, while the dashed lines correspond to the edges of the Delaunay Triangulation

.

The cost of generalizing VHash to utilize any geometric 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 geometric space in a distributed fashion [9]. Previous explorations into distributed approximations of a node's Voronoi region offer constant time approximations. The greedy heuristic we use is approximately as fast as a single sample calculation in the Monte-Carlo approach used in RayNet [9]. As a heuristic, there exist edge cases where it is incorrect. We show in the experimental section that this loss in accuracy has no practical impact.

B. Distributed Greedy Voronoi Heuristic

C. Peer Management

VHash maintains two peer lists: *Short Peers* and *Long Peers*. This is motivated by mitigating the error induced by DGVH and providing robustness against churn[4].

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

Short Peers are the subset of the Delaunay Peers generated by DGVH. 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 has found a useful lower bound on peers to be $3d + 1$ [9]. Using a lower bound on the length of the short peer list corrects for errors in the approximation processes by including peers that would otherwise be omitted. $3d + 1$ is used as the lower bound for the size of the short peer list. Should the number of short peers generated by DGVH 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. Members of *Short Peers* are analogous to the predecessors/successors in other DHTs.

Long Peers is the list of two-hop neighbors 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 $(3d + 1)^2$. For example, if the short peer list has a minimum size of 8, the long peer list has a maximum size of 64 entries. Members of *Long Peers* are not actively probed during maintenance and the cost of managing them is minimal.

We next discuss how nodes learn about short and long peers.

D. 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[5] and uses DGVH 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 Algorithm 3. This maintenance through gossip process is very similar to the gossip protocol used in [9].

E. Handling Churn

Churn refers to the effects caused by the continuous joining and exiting of nodes to and from the overlay.

[5]Nodes remove themselves and repetitions from the candidates.

Algorithm 3 Gossiping

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

DHTs must have some mechanisms to handle this process to maintain fault tolerance and accurate routing as their members change over time.

Joining the VHash 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 VIII and uses the patron to find the node responsible for its assigned location, which we call the parent. The prospective node sets the parent node as the lone member of *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" method of leaving the network. VHash assumes nodes will fail often and abruptly and that the distinction between an intended failure and unintended failure is unnecessary. Should a node wish to leave the network, it only needs to cease sending and receiving messages. Suppose a node f fails or leaves the network; we assume it does so 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 then 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.

F. Routing

The proper forwarding peer for routing a message extends from the Voronoi regions of *Short Peers*and *Long Peers*. For the purposes of routing, each node assumes itself and its short and long peers are the only nodes that exist in the geometric space. When a node receives a message destined for a particular location, that node determines whether or not it is closest to that location. This is the same as determining whether or not the specified location falls into the node's Voronoi region. If this is the case, the node handles the message accordingly.

Otherwise, one of its peers must be responsible for that location. It forwards the message to the peer it deems closest to the location, which repeats the same decision process, but with different knowledge.

This process is equivalent to a precomputed, cached, and efficient routing algorithm (Algorithm 4), shown to be polylogarithmic by [9] [16] [17]. In the case that our approximation of the Voronoi region of a peer is incorrect due to each nodes incomplete knowledge of the network, our approximation describes the most efficient forwarding path of a message to a particular location.

Algorithm 4 VHash Lookup

1: Given node n
2: Given 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**

G. Algorithm Analysis

DVGH is very efficient in both terms of space and time. Suppose a node n is creating its short peer list from k candidates in an overlay network of N nodes. The candidates must be sorted, which takes $O(k \cdot \lg(k))$ operations. Node n must then compute the midpoint between itself and each of the k candidates. Node n then compares distances to the midpoints between itself and all the candidates. This results in a cost of

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

Since k is bounded by $\Theta(\frac{\log N}{\log \log N})$ [18] (the expected maximum number of peers stored between two neighbors), we can translate the above to

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

In the vast majority of cases, the number of peers is equal to the constant minimum table size. This yields $k = (3d+1)^2 + 3d + 1$ in the expected case, where the lower bound and expected complexities are $\Omega(1)$.

Previous work [9] claims constant time approximation, the reality is that Raynet's leading constant is in the order of thousands as Monte-Carlo samples. Our algorithm has a greater asymptotic worst case cost, for all current realistic network sizes it will be more time efficient then RayNet's approximation.

VIII. Network Metric Embedding

During previous research in applications of DHTs [19] to parallel processing, Rosen et al. tested DHT network behavior under churn. They came to the conclusion that high levels of churn improved the performance of the system under certain circumstances. The built-in mechanisms for managing responsibility in a DHT are very effective and efficient at handling nodes changing locations in the network. In this work, Rosen et al. found that it was more effective to move the node to a location in the DHT where work or data was stored rather than attempt to remap data efficiently over the nodes.

VHash is designed to leverage this discovery and allow a nodes' location in the network to provide meaningful information about the node and to allow nodes to change location in the network to suit the networks needs. In this paper we propose a method that enables peers 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. Each node is given a random initial location and, over time, migrates to a location which optimizes latency.

We utilize a distributed, online 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 network performs a periodic update step to its location to accurately represent inter-node latency on 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 [20].

To perform the embedding, we designed an algorithm for nodes to utilize based on force-directed graph drawing [21] The algorithm periodically updates the node locations to lower the latency with their peers. The

approximate distributed embedding algorithm employs VHash's maintenance and join behaviors. This algorithm is described in Algorithm 5. Each node periodically measures the latencies to its peers and generates a change in position that minimizes the error in distance to its peers. Once a node finds peers with low latency to it, these error approximations become small and the network configuration becomes stable. If a node has high latency with its current peers, the high latency connection causes the nodes to repel each other and move to new peers.

Algorithm 5 Decentralized Peer-to-Peer Force-Directed Model

Given 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

IX. Simulations and Results

We implemented two simulations to prove that VHash acts as as a stable DHT and is able to reduce the latency of lookup time. Our first simulation examined successful lookup rates (hit rate) 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 reduce latency. We wanted to demonstrate the benefits to latency that followed from optimizing VHash's overlay topology for minimum latency according to our force directed model (Algorithm 5).

A. Convergence Simulation

This simulation demonstrates that the VHash protocol converges to a stable overlay from a chaotic starting topology after a sufficient number of gossip cycles. We do this by showing that the rate of successful lookups approaches 1.0. We compare these results to RayNet [9], 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 each cycles, the nodes perform a gossip with a random node from their short peers to recompute both their peer lists. We then calculate the hit rate of successful lookups by simulating 2000 lookups from random nodes to random locations, as described in Algorithm 6.

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 hit rate at each cycle is $\frac{hits}{2000}$, where hits are the number of successful lookups.

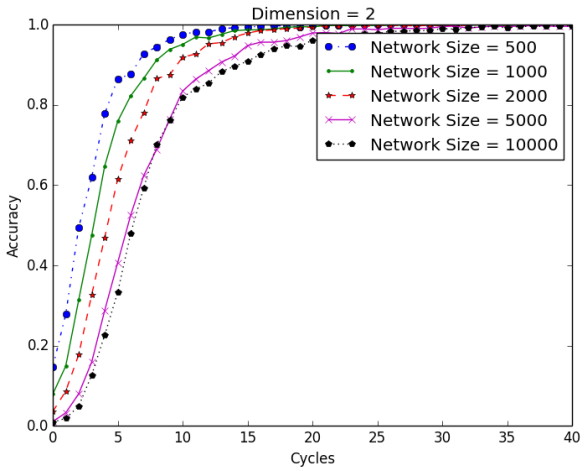
Algorithm 6 Routing Simulation Sample

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

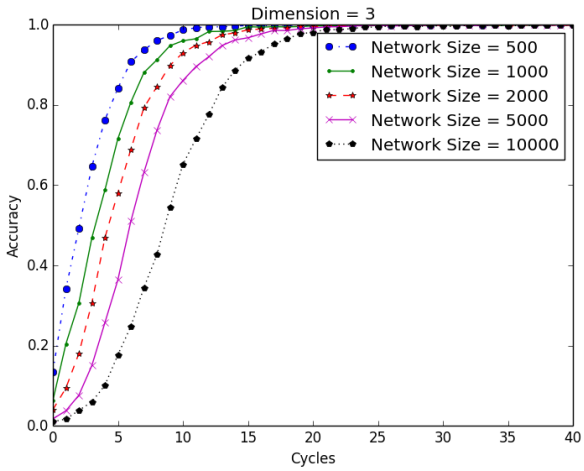
Our results are shown in Figures 3a, 3b, 3c, and 3d. Our graphs show that VHash's overlay quickly constructs itself from a random configuration and that our hit rate reached 90% by cycle 20, regardless of dimension. VHash consistently approached a hit rate of 100% by cycle 30. In comparison, RayNet's routing converged to a perfect hit rate at around cycle 30 to 35 [9] As the network size and number of dimensions each increase, convergence slows, but not to a significant degree.

B. Latency Distribution Test

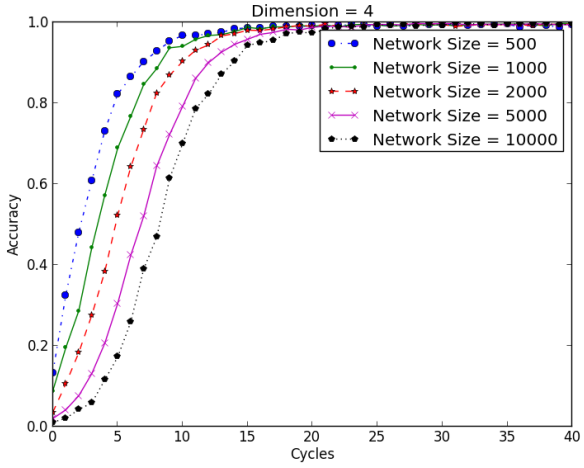
The goal of our second set of experiments is to demonstrate VHash's ability to optimize a selected network metric: latency in this case. In our simulation, we use the number of hops on the underlying network as an approximation of latency. We compared VHash's performance to that of a more traditional DHT, Chord [12]. 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 [11] [12] [13] [9] [17], we are concerned with the actual latency lookups



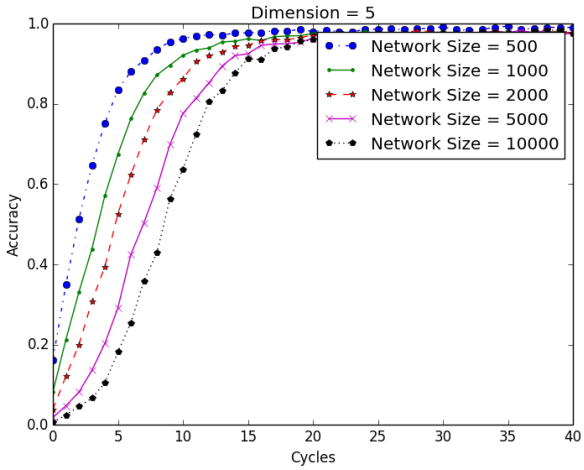
(a) This plot shows the accuracy rate of lookups on a 2-dimensional VHash network as it self-organizes.



(b) This plot shows the accuracy rate of lookups on a 3-dimensional VHash network as it self-organizes.



(c) This plot shows the accuracy rate of lookups on a 4-dimensional VHash network as it self-organizes.



(d) This plot shows the accuracy rate of lookups on a 5-dimensional VHash network as it self-organizes.

Fig. 3: These figures show that, starting from a randomized network, VHash forms a stable and consistent network topology. The Y axis shows the success rate of lookups and the X axis show the number of gossips that have occurred. Each point shows the fraction of 2000 lookups that successfully found the correct destination.

experience traveling through the *underlay* network, the network the overlay is built upon.

Overlay hops are used in most DHT evaluations as the primary measure of latency. It is the best approach available when there are no means of evaluating characteristics of the underlying network. VHash is designed with a capability to exploit the characteristics of the underlying network. 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) as an underlay network [22] [23] [24]. We used a scale-free network as the underlay, as it is a simplified model of the Internet's topology [22] [23]. 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 random source-destination nodes from the overlay. VHash generates an embedding of the latency graph utilizing the distributed force directed model algorithm described in Algorithm 5, with the latency function defined as the number of underlay hops between it and its peers.

Our simulation created 100, 500, and 1000 node over-

lays for both VHash and Chord. We used 4 dimensions in VHash and a standard 160 bit identifier for Chord.

Figures 4a, 4b, and 4c 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 significantly reduced the underlay path lengths. Besides having a much lower average path length, the variance was also considerably lower.

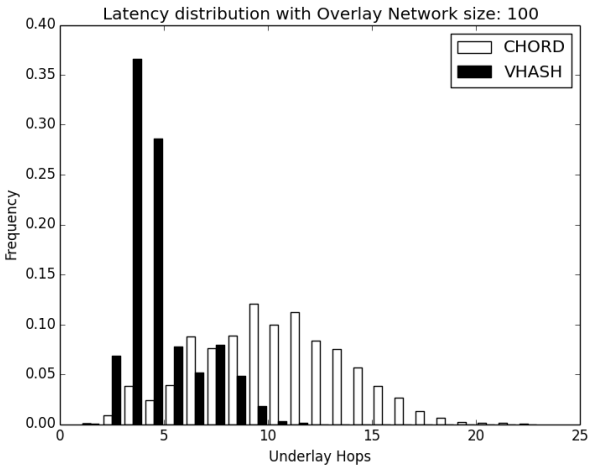
For comparison, we also sampled the lookup length measured in overlay hops for a 1000 sized Chord and VHash network. As seen in Figure 5, the paths in VHash's overlay were significantly shorter than those in Chord. In comparing the overlay and underlay hops, we find that for each overlay hop in Chord, 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. Recall that this work is based on scale free networks, where latency improvements are difficult. An improvement of 0.4 hops over a diameter of 3 hops is significant. VHash has on average less overlay hops per lookup than Chord, and for each of these overlay hops we consistently traverse more efficiently across the underlay network.

X. Related Work

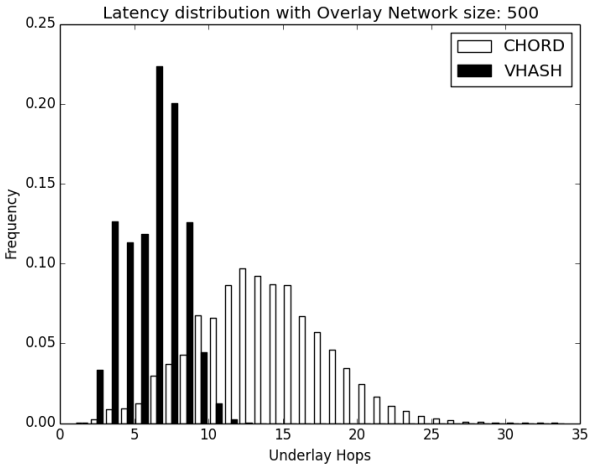
While there has been previous work on applying Voronoi regions to DHTs and peer-to-peer (P2P) applications, we have found no prior work on how to perform embedding of an inter-node latency graph.

Backhaus et al.'s VAST [6] is a Voronoi-based P2P protocol designed for handling event messages in a massively multiplayer online video game. Each node finds its neighbors by constructing a Voronoi diagram using Fortune's sweepline algorithm [7]. VAST demonstrated that Voronoi diagrams could be used as the backbone to large-scale applications, 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 expensive to solve.

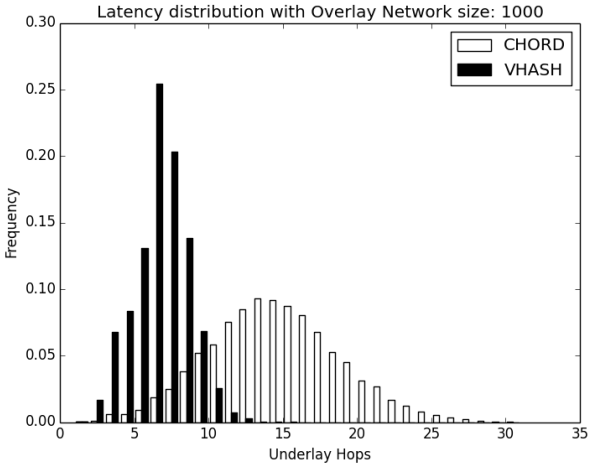
The two DHT protocols developed by Beumont et al., VoroNet [17] and RayNet [9] are the closest comparisons to VHash. VoroNet is based off Kleinberg's small world model [16] and achieves polylogarithmic lookup time. Each node in Voronet solves its Voronoi region to determine its neighbors and also maintains a link to a randomly chosen distant node. Voronet focused specifically on the two-dimensional Voronoi computations and the techniques used would be too expensive in higher dimensions and were not resilient to churn [9].



(a) Frequency of path lengths on Chord and VHash in a 100 node overlay.



(b) Frequency of path lengths on Chord and VHash in a 500 node overlay.



(c) Frequency of path lengths on Chord and VHash in a 1000 node overlay.

Fig. 4: Figures 4a, 4b, and 4c show the difference in the performance of Chord and VHash for 10,000 routing samples on a 10,000 node underlay network for differently sized overlays. The Y axis shows the observed frequencies and the X axis shows the number of hops traversed on the underlay network. VHash consistently requires fewer hops for routing than Chord.

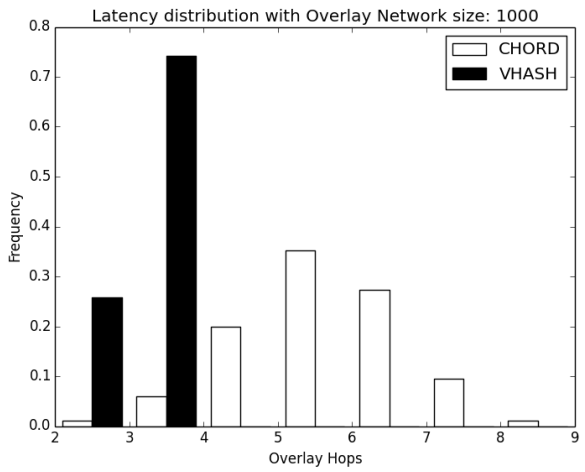


Fig. 5: Comparison of Chord and VHash in terms of overlay hops. Each overlay has 1000 nodes. The Y axis denotes the observed frequencies of overlay hops and the X axis corresponds to the path lengths in overlay hops.

RayNet [9] 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. RayNet uses a Monte-Carlo method to approximate the volume of a node's Voronoi region. While effective at 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.

XI. Conclusions and Future Work

Our experiments show that VHash outperforms Chord in overlay-hop latency, underlay-hop latency, and traverses the underlying network more effectively. For each overlay hop in Chord, the lookup must travel an average of 2.719 underlay hops per overlay hop. For each overlay hop in VHash, the lookup instead travels only 2.291 underlay hops per overlay hop on average. In the context of a scale free graph with an approximate diameter of 3, a difference of 0.4 hops is a significant improvement. Further gains in overlay-hop efficiency can be expected from applying VHash with a hyperbolic plane geometric space [20]. This would allow for construction of optimally minimal latency networks in VHash.

While we only focus on network latency in this paper as an initial exploration, VHash may be used to optimize multiple network metrics. These additional metrics can be implemented by mapping them to additional dimensions. Much of our future work will look at applying VHash to network metrics such as energy or transmission range.

Another venue for exploration is application of caching and replication strategies to a functional distributed file system running on top of VHash. Such extensions seek to improve upon existing work done on file replication and caching schemes [25].

References

[1] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, pp. 345–405, Sept. 1991.

[2] W. Wang, G. Yang, N. Xiong, X. He, and W. Guo, "A general p2p scheme for constructing large-scale virtual environments," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pp. 1648–1655, May 2014.

[3] B. Carbunar, A. Grama, and J. Vitek, "Distributed and dynamic voronoi overlays for coverage detection and distributed hash tables in ad-hoc networks," in *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on*, pp. 549–556, IEEE, 2004.

[4] S.-Y. Hu and G.-M. Liao, "Scalable peer-to-peer networked virtual environment," 2004.

[5] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang, "Voronoi state management for peer-to-peer massively multiplayer online games," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pp. 1134–1138, IEEE, 2008.

[6] 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.

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

[8] D. F. Watson, "Computing the n-dimensional delaunay tessellation with application to voronoi polytopes," *The computer journal*, vol. 24, no. 2, pp. 167–172, 1981.

[9] 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.

[10] B. Cohen, "The bittorrent protocol specification," 2008.

[11] 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.

[12] 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.

[13] 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.

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

[15] G. S. Manku, M. Bawa, P. Raghavan, *et al.*, "Symphony: Distributed hashing in a small world.," in *USENIX Symposium on Internet Technologies and Systems*, p. 10, 2003.

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

[17] 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.

[18] 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.

[19] A. Rosen, B. Benshoof, M. Erwin, R. W. Harrison, and A. G. Bourgeois, "Chronus: Organizing Mapreduce with Chord DHT."

[20] 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.

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

[22] 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.

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

[24] 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.

[25] H. Shen, "Irm: integrated file replication and consistency maintenance in p2p systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 100–113, 2010.