VHash: A Voronoi-Based Multidimensional Distributed Hash Table

Double Blind

***Abstract*—Distributed Hash Tables are used as a tool to generate overlay networks for P2P networks. Current DHT techniques are not designed to take the nature of the underlying network into account when organizing the overlay network. Current DHT networks assign nodes locations in a ring or tree, limiting the ability of these networks to be more efficient. A DHT technique that allows for efficient construction of an overlay network that takes into account the real underlying network would allow for higher performance and faster P2P networks. We present VHash as a spacial DHT based on approximate Delaunay Triangulation to integrate distance information between nodes into overlay network topology. VHash allows for the creation of P2P networks with faster record lookup time, storage, and maintenance with a geographically diverse set of nodes.**

I. Introduction

A Distributed Hash Table is used provide an overlay network for many P2P applications. State of the art DHT techniques are built on trees or $\log$-ring structures to ensure that the routing distance is $O(lg(n))$ hops between nodes.

In the vast majority of Distributed Hash Tables, such as Chord [1], Kademlia[2], Pastry [3], a node is mapped key on a 1-dimensional keyspace. This key is chosen via a hash function, such as SHA-1, ensuring that nodes are randomly and uniformly the overlay network. This provides the network with fault tolerance; if all the nodes located in a real geographic region were suddenly taken offline, the damage to the network would be spread uniformly throughout the network and maintenance would repair the damage. These topologies, while sufficient in reasonably local networks, do not take embed the lengths or latencies of routes defined by the topology and assume that every hop has similar latency and throughput. For a global network, a more intelligent means of generating a dynamic overlay network with efficient routing, storage, and backups is needed for future P2P applications.

We present VHash as a DHT designed to take inter-node latency information into account when generating an overlay on a massive scale. VHash creates an approximation of a Voronoi network 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. The naive method of doing so is to assign coordinates to servers based on the geographic location of nodes. More complex approaches approximate a minimum latency space based on internode latency.

Our paper presents the following:

- We present the algorithms that are used to approximate the Voronoi region and demonstrate that these approximations are accurate and sufficient enough to efficiently route between arbitrary nodes.
- We show how VHash can be used to create a robust, fault-tolerant file-sharing service.
- We created a simulation of our protocol and compared it to previous Voronoi based algorithms.
- We present the related work and how VHash improves upon the previous work and identify future areas of fruitful research.

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 and Delaunay Triangulations

Voronoi diagrams define ownership of a space, with each object in the space owning all the points closest to it. We denote the region owned by an object as a Voronoi region, or simply that object's region. In VHash, these objects are nodes mapped to a d-dimensional toroidal space and the regions they own define the range of keys they are responsible for.

By assigning nodes and files a set of coordinates composed of not only the hashkey, but information that can be used as a metric for routing, allows us to optimize routing in the the network along that metric without jeopardizing the distributed quality of the network. We are particularly interested in embedding latency as a measurement of true distance between nodes. Algorithm 1 describes the process for performing a minimum latency embedding using VHash.

A Voronoi diagram is the division of a d-dimensional 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 the object. We refer to the region owned by an object as that object's Voronoi region. The Delaunay Trangulation 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 iff those object's Voronoi region border each other. This means that solving either problem will yield the solution to both.

In our network, the nodes are the objects of the Voronoi diagram and their regions define the keyspace they are responsible for. The edges created by the Deluanay Triangulation correspond to the connections between neighboring nodes.

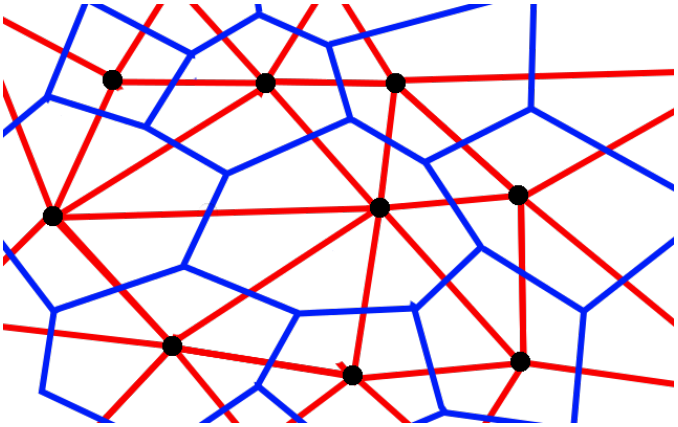

Fig. 1: The starting network topology. The blue lines demark the Voronoi edges, while the red lines connecting the nodes correspond to the Delaunay Triangulation edges and one-hop connections.

However, computing Voronoi diagrams is expensive, but a greedy approximation of the Voronoi regions is sufficient for the 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].

Algorithm 1 VHash Minimum Latency Embedding via P2P Spring Model

1: This needs to be written

B. Approximation Algorithm

VHash maps nodes to a d dimension toroidal unit space overlay. This is essentially a hypercube with wrapping edges. The toroidal property makes visualization difficult but allows for a space without a sparse edge, as all nodes can translate the space such that they are at the center of the space. In effect, each node views itself at the center of the graph.

As VHash uses multiple dimensions, responsibility for a key is assigned to the node closest to that key. Given two vector locations $\vec{a}$ and $\vec{b}$ on a d dimensional unit toroidal hypercube, the distance between them is:

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

[1]

VHash does not strictly solve Voronoi diagrams [6] for two reasons. First, the toroidal nature of the space preclude the traditional means of solving for Voronoi regions. Second, computing a Voronoi diagram in spaces where $d \geq 3$ is prohibitively expensive [7]and most of the efficient algorithms concern themselves with obtaining a Voronoi diagram of the entire space, rather than from the perspective of a distributed network, with each member calculating its own region. Rather than attempt to calculate the Voronoi region of each node, our algorithm simply filters locations, assigning responsibility to the nearest node.

Each cycle, nodes exchange their peerlists with their current neighbors and then recalculate their neighbors. The calculation is straightforward. After a node receives it's neighbor's peerlists, it combines their peerlists and its own into a list of candidate neighbors, sorted the nodes by distance from closest to furthest (using the distance metric from REF). A new peerlist is then created starting with the first candidate from the list of candidates. The node then looks at each of the remaining candidates and calculates the midpoint between the node and the candidate. If any of the nodes in the new peerlist are closer to the candidate, the candidate is set aside. Otherwise the candidate is added to the new peerlist.

To reduce the effects of nodes occluding one another when they are clustered together, each node maintains at minimum $3d + 1$ neighbors. If the node has remaining slots left over after creating a new peerlist, the remaining slots are filled up the closest remaining candidates.

Algorithm 2 Approximation using Greedy Peer Selection

1: $Candiates$ is the set of candidate peers
2: $Peers$ is the set of this node's peers
3: $Canidates$ is sorted by each node's closeness to this node
4: The closest member of $Canidates$ is popped and added to $Peers$
5: **for all** n in $Canidates$ **do**
6: c is the midpoint between this node and n
7: **if** Any node in $Peers$ is closer to c than this node **then**
8: reject n as a peer
9: **else**
10: Add n to $Peers$
11: **end if**
12: **end for**

By finding the correct (enough) neighbors, a node is able to approximate its local Voronoi region close enough for routing If a node can figure out its Delaunay neighbors, it can extrapolate the Voronoi region (ie the region it's responsible for) from that.

C. Algorithm Analysis

VHash's Voronoi approximation is extremely Efficient, in both terms of space and time. Suppose a node is creating a peerlist from k candidates to choose from. 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 the largest k can be is $6d$ (the maximum number of unique stored neighbors allowed between two neighbors), we can translate the above to

[1] check

$$6d \cdot \lg(6d) + 6d \text{ midpoint computations}$$

$$+(6d)^2 \text{ distance computations}$$

Since k is dependent only on the number of dimensions, d, k itself is constant and the algorithm runs in constant time with a small coefficient. This is a very large improvement over the volume approximation used by Raynet [7], which required 1000 distance calculates to do a Monte-Carlo approximation of the Voronoi region. The routing tables will have a minimum size of $3d + 1$ but have an unbounded maximum size.

It is theoretically possible to create a network in which a single node is the neighbor of every other node in the network, but it is highly unlikely. The *expected maximum degree*[2] of an object in a d-dimensional Delaunay Triangulation in a network of size N is $\Theta(\frac{\log N}{\log \log N})$ [8].

Accuracy: The key to the greedy approximation working is that it just needs to be accurate to route. A node will not falsely believe that it owns a particular key; a key not belonging to it will always fall into the domain of some peer and be closer to that peer or another node that the peer knows of.

The routing tables in VHash are $O(1)$ space, while providing polylogarithmic average lookup time measured in hops?[3] average lookup time in hops with a minimized latency.

Consider a d-dimensional toroidal space. Let some arbitrary point A be the center of this space[4]. A hypercube that surrounds the space would be defined by $3^d - 1$ points citation [5]

Greedy peer selection is predicated on the assumption that when two nodes' voronoi region share an edge that the midpoint between them also falls on this segment. This is mitigated by the usage of long-peers. Should two nodes share an edge on thier vornoi region while the midpoint between them is occuluded by the region of another peer they will not form an accurate vonoi region or peer list. Should the occluding node have both occuluded nodes as peers then this will be solved by keeping long peers. Should chains of these occlusions occur, then the hit rate for those regions will suffer. Such chains are highly unlikely to exist without the intent to create them and likely will not occur during the normal operation of the network. The only solution to this flaw is to fall back on methods significantly slower than the greedy voronoi approximation (see volume based approximation and exact calculation).

D. Routing

1) Messages: Maintenance and joining are handled by a simple periodic mechanism. A notification message consisting of a node's information and active peers is the only maintenance message. All messages have a destination hash location which is used to route them to the proper server. This destination can be the hash location of a particular node or the location of a desired record or service. The message is received by the node responsible for the location. Services running on the DHT define their own message contents, such as commands to store and retrieve data.

2) Message Routing: Messages are routed over the overlay network using a simple algorithm (Algorithm 3). When routing a message to an arbitrary location, a node calculates who's Voronoi region the message's destination is in amongst the itself and its peers. If the destination falls within its own region, then it is responsible and handles the message accordingly. Otherwise, the node forwards the message to the closest peer to the destination location. This process describes[6] a pre-computed and cached efficient routing algorithm. Even if our approximation for the Voronoi region of a peer is incorrect due to incomplete knowledge of the network, our approximation describes the efficient forwarding path of a message destined for that space.

Algorithm 3 Vhash Routing

```
P_0 is this node's set of peers
N is this node
m is a message addressed for L
Forwards is the set P_0 ∪ N
find C: member of Forwards which has the shortest distance to L
if C is N then
    N is the responsible party.
    Handle m
else
    Forward m to C for handling or further routing
end if
```

3) Provided Routing Time:

E. Maintenance

This section is actually spread around the document at the moment

The layout should be maintenance, routing, approximation

Peerlist: A node maintains fairly limited information about the state of the network, as it only needs to hold information about its immediate vicinity to calculate the approximation of the Voronoi region. Specifically, the node maintains two lists of peers.

1) The *short peers* or *neighbors*. These are the peers sharing an edge with the node from the computed Delaunay Triangulation. *Short Peers* has a limited size of FORMULA. There is no maximum to the size that the list of Short Peers can have, but the expected maximum size in a network with N nodes is $\Theta(\frac{\log N}{\log \log N})$ [8].
2) The *long peers* correspond to the two hop peers of the node. This list, which can have anywhere between 0 and FORMULA entries, helps the approximation algorithm avoid making errors due to occlusion.

[7]

[2]This is maximum number of peers that a node would have in a non-contrived example. While the degree for a node can theoretically be $O(n)$ in a Delaunay triangulation, it is extremely unlikely.

[3]Our initial assessment was d-root, but other papers say Polylogarithmic routing [9] [7]

[4]Any point in a toroidal space can view itself as it's center

[5]Is this rigorous enough?

[6]Wording. Is equivalent a better word?

[7]WE NEED TO GO THRU AND CLARIFY EACH INSTANCE OF PEER AND NEIGHBOR. I'VE BEEN USING PEERLIST FOR SHORT PEERS

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 peers* with each other. The node combines the lists of *short peers* and uses the approximation algorithm determine which of these candidates correspond to its neighbors along the Delaunay Triangulation. The candidates determined not to be neighbors become *long peers*[8]. The formal algorithm for this process is described by Algorithms 5 and 6

This maintenance through gossip process is very similar to the gossip protocol used in [7].

When messages sent to a peer fail, it is assumed the peer has left the network. The leaving peer is removed from the peer list and candidates from the set of 2-hop peers provided by other peers move in to replace it. . Figures 2, 3, and 4 illustrate the joining processing.

Joining: Joining the network is a straightforward process. A new node first learns the location of at least one member of the network to join. The joining node then chooses a location in the hash space either at random or based on a problem formulation (for example, based on geographic location or latency information).

After choosing a location, the joining node sends a "join" message to its own location via the known node. The message is forwarded to the current owner of that location who can be considered the "parent" node. The parent node immediately replies with a maintenance message containing its full peer list. This message is sent to the joining node, who then uses this to begin defining the space it is responsible for.

The joining node's initial peers are a subset of the parent and the parent's peers. The parent adds the new node to its own peer list and removes all his peers occluded by the new node. Then regular maintenance propagates the new node's information and repairs the overlay topology. This process is enumerated by Algorithm 4.

Algorithm 4 Vhash Join

1: new node N wishes to join and has location L
2: N knows node x to be a member of the network
3: N sends a request to join, addressed to L via x
4: node $Parent$ is responsible for location L and receives the join message
5: $Parent$ sends to N its own location and list of peers
6: $Parent$ integrates N into its peer set
7: N builds its peer list from N and its peers
8: regular maintenance updates other peers

There is no function for a "polite" exit from the network. VHash assumes nodes will fail and the difference between an intended failure and unintended failure is unnecessary. The only issue this causes is that node software should be designed to fail totally when issues arise rather then attempt to fulfill only part of its responsibilities.

[8]which is prevented from becoming to large by...

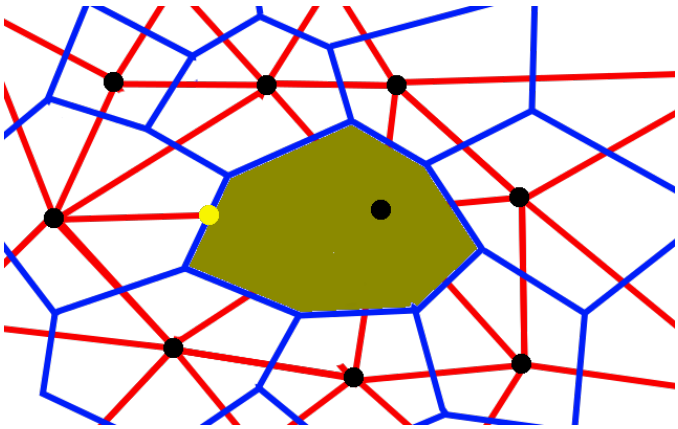

Fig. 2: Here, a new node is joining the networks and has established that his position falls in the the yellow shaded Voronoi region.

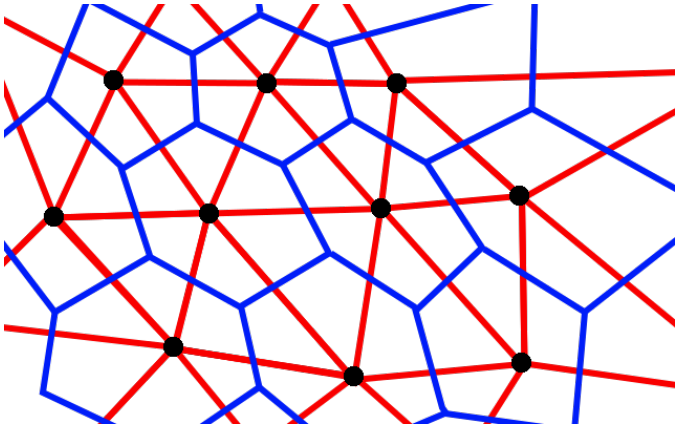

Fig. 3: The network topology after the new node has finished joining.

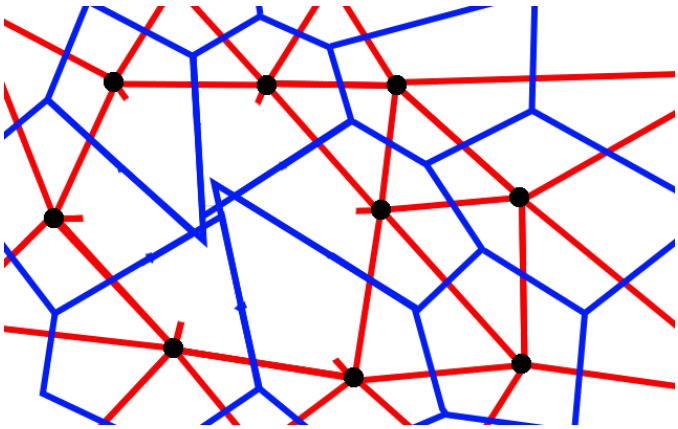

Fig. 4: The topology immediately after the new node leaves the network. After maintenance takes place, the topology repairs itself back to the configuration shown in Figure 1.

Algorithm 5 VHash Maintenance Cycle

```
1: P_0 is this node's set of peers
2: T is the maintenance period
3: while Node is running do
4:    for all node n in P_0 do
5:       Send a Maintenance Message containing P_0 to n
6:    end for
7:    Wait T seconds
8: end while
```

Algorithm 6 VHash Handle Maintenance Message

```
1: P_0 is this node's set of peers
2: Receive a Maintenance Message from peer n containing
   its set of peers:P_n
3: for all Peers p in P_n do
4:    Consider p as a member of P_0
5:    if p should join P_0 then
6:       Add p to P_0
7:       for all Other peers i in p do
8:          if i is occluded by p then
9:             remove i from P_0
10:         end if
11:      end for
12:   end if
13: end for
```

F. Data Storage and Backups

The primary goal of a DHT is to provide a distributed storage medium. We extend this idea to distribute work and information among nodes using the same paradigm. Resources in the network, be it raw data or assigned tasks, are assigned hash locations. The node responsible for a given hash location is responsible for the maintenance of that resource. When a node fails, its peers take responsibility of its space. Thus it is important to provide peers with frequent backups of a node's assigned resources. That way, when a node fails, its peers can immediately assume its responsibilities.

When a resource is to be stored on the network, it is assigned a hash location. The hash locations assigned could be random, a hash of an identifier, or have specific meaning for an embedding problem. The node responsible for that resource's hash location stores the resource.

A resource is accessed by contacting the node responsible for the resource. However, the requester generally has no idea which node is responsible for any particular resource. The data request messsage is addressed to the location corresponding to the resource, rather than the node responsible for that location. The message is forwarded over the overlay network, each hop bringing the node closer until it reaches the responsible node, who sends the resource or an error if the resource does not exist.

Some options are immediately apparent for dealing with wasted storage space. A system that is primarily read driven can record the time of the last read or a frequency of reads such that resources that are not read often enough are deleted after a certain period of time. If a system is write driven, allow the resource to be assigned a time to live, which can be updated as needed.

A node periodically sends a message containing backups of the resources for which it became newly responsible for to each of its peers. To minimize bandwidth and time wasted by backups, the node should only send the records changed since last backup.

G. Fault Tolerance

Fault tolerance is a necessary part of any Distributed Hash Table. VHash can easily deal with churn, the constant inaccuracies of nodes joining and leaving the network, using an extremely lightweight and reactive pruning of peers.

Suppose a node f fails and 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, in the case of routing. The network can rely on the incessant gossiping of nodes to create new links among the nodes.

1) Files: They get stored in the short peers, so no worries. Lookup would be successful for any retreive. Stores on the other hand, can be more volatile. If a node has just dropped out of the network, it's neighbors will appropriately recalculate the region within a few cycles. If a store command is issued during that time, routing may direct it to the wrong node temporarily; the node will believe it is responsible for that file until the Delaunay neighbors have been recalculated.

H. Error Analyis

As our construction of the Voronoi Diagram and the Delaunay Triangulation is an approximation, it is not completely immune to error. Error in the graph results from occlusion (example). We allivate this with a minimum short peer size and using long peers.

I. Key Generation - Mapping a Node to Coordinates

We suggest using the following spaces as our dimensions

This is a new Topic. We're going to talk about the options for this in the section , but taking advantage of these is it's own work.

1) Cryptographic Keyspace:

2) Geographic Coordinate:

3) Latency - Spring Based Model:

4) Security/Trust Space:

5) Social Network Influence as an attribute.:

6) Memory Overhead: In order to route, a node maintains a routing table consisting of the nodes it borders a region with. Unlike other DHTs, which keep routing tables of a set size, VHash's routing table depends on the nodes that it shares a border with, which is on average $3^d - 1$???

III. Simulations

We simulated VHash to ascertain the protocol's performance.

A. Underlay Routing

Our first set of simulations tested VHash's ability to converge towards a topology that would correctly route messages

What we didn't show and why: A non chaotic starting point because that was no challenge, chatty join was immediately convergent

What we didn't test and why:

B. Latency

We began by creating a large underlay network. Then n random nodes were chosen to join the overlay Nodes in the suimulation would route a message to one another and we plotted the latency and successful deliveries for VHash and Raynet. Latency was measured by the number of hops in the *underlay* network. For comparison we also compared the performance if messages were routed perfectally and optimally to see how close to the ideal our performance was.

C. Hop Details

IV. Related Work And Future

Voronet stuff here.

Beaumont *et al* [7] argues that a loose structure enough for searching. Assume a d-dimension space, each dimension tied to some attribute of an object and each object identified by a unique set of values. Objects should be linked to other objects that are close in the space.

The key insight that Beaumont *et al* had was that nodes only needed to calculate their regions locally and this can be done via goosip-based protocol.

Each node maintains a *view*, the closest $3d+1$ neighbors it knows of and periodically exchanges information with them.

Pastry also tried to address this problem.

Apply this to MANET.

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 SIGCOMM Computer Communication Review*, vol. 31, pp. 149–160, ACM, 2001.

[2] 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.

[3] 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.

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

[7] 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.

[8] 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.

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