UrDHT: A Unified Model for Distributed Hash Tables

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

***Abstract*—UrDHT is an abstracted Distributed Hash Table (DHT) that maps the topologies of DHTs to the primal-dual problem of Voronoi Tessellation and Delaunay Triangulation. Many of the DHTs have an inherent set of the same qualities, such as greedy routing, maintaining lists of peers which define the topology, and forming an overlay network. By completing a few simple functions, a developer can implement the topology of any DHT in any arbitrary space.**

One topology of particular interest we created is a DHT operating within a Poincaré disk model. This DHT has latency embedded within the overlay and capable of responding to changes in latency. The consequence of this is that, unlike other DHTs, the routing algorithm always uses the shortest latency path to a given destination.

I. Introduction

UrDHT is an abstract implementation of a DHT which solves a number of problems. First, it is a unified and cohesive model for creating distributed hash tables and P2P applications based on DHTs. Second, it provides a single network for bootstrapping distributed applications. Third, we show that using the abstraction features of UrDHT, we can embed latency into the DHT's overlay.

1) Abstraction: Distributed Hash Tables have been the catalyst for the creation of many P2P applications. Among these are Redis [1], Freenet [3], and, most notably, BitTorrent [4]. All DHTs use roughly the same protocol to perform lookup, storage, and retrieval operations. Despite this, no one has created a cohesive formal specification for building a DHT.

Our primary motivation for this project was to create an abstracted Distributed Hash Table based on observations we made during previous research [2]. In that paper, we discovered that all DHTs can cleanly be mapped to the primal-dual problems of Voronoi Tessellation and Delaunay Triangulation.

UrDHT directly builds its topology using this insight, using a greedy distributed heuristic for approximating Deluanay triangulations. It is our specification of an abstract DHT, which can be used to build of DHTs. We found that we could reproduce the topology of different DHTs by defining the necessary distance and midpoint functions for the space.

The end result is not only do we have an abstracted DHT, we have a simple framework that developers can use to quickly create new distributed applications and frameworks.

2) Bootstrapping: Another ambiguous issue with DHTs and DHT-based P2P applications we wish to address with UrDHT is the what we have termed the *bootstrapping problem*. Simply put, a node can only join the network if it knows another node that is already a member of the network it is trying to join.

The general way this works is by having a potential user manually look up the bootstrapping information needed at a centralized source, such as the project or application's website. There is a philosophical conflict here: a distributed application using a centralized source of information to build a distributed network.

UrDHT exists to simplify this process, minimizing the distributed application development time and making it easier to adopt by creating a network to bootstrap *other networks*. UrDHT does this by making it easy to add other networks as a service.

3) Embedding: One of the other features shared by nearly every DHT is that routing works by minimizing the number of hops across the overlay network, with all hops treated as the same length. This is done because it is assumed that DHTs know nothing about the state of actual infrastructure the overlay is built upon.

However, this means that most DHTs will happily route a message from one continent to another and back. This is obviously undesirable, but it is the status quo in DHTs. This stems from the generation of node IDs in DHTs. Nodes are typically assigned a point in the range of a cryptographic hash funtion, called the keyspace. This ID corresponds to the hash of some identifier or given a point randomly. This is done for purposes of load balancing and fault tolerance.

We asked ourselves if there was a means of embedding latency into the DHT, while still maintaining the system's fault tolerance. Doing so would mean that the hops traversed to a destination are, in fact, the shortest path to the destination.

We found that we could embed a latency graph in a hyperbolic space and defined UrDHT such that it operates within this space. The end result was a DHT with latency embedded into the overlay. Nodes can respond to changes in latency and the network by effectively moving their positions.

Accomplishments: To summarize our accomplishments:

- We give a formal specification for what needs to be defined in order to create a functioning DHT. While there has long existed a well known protocol for distributed hash tables, these define what a DHT needs to be able to do. It does not describe what a DHT is.
  We show that DHTs cleanly map to the primal-dual problem of Delaunay triangulations and Voronoi tessellations. We list a set of simple functions that, once defined, allow our Distributed Greedy Voronoi Heuristic (DGVH) to be run in any space, creating a DHT overlay for that space (Section II).
- We present UrDHT as an abstract DHT and show how a developer can tweak the functions we defined to create an arbitrary new DHT topology (Section III).
- We show how to reproduce the topology of Chord and Kademlia using UrDHT, which we call UrChord and UrKademlia (Section IV).
- We show how we can embed latency into the overlay of a DHT by placing the DHT into and defining the necessary methods so that DGVH can run in that space (Section V).
- We conduct experiments showing that UrChord sufficiently approximates a correct implementation of Chord (Section VI).
- We discuss the ramifications of our work and what future work is available.

II. What Defines a DHT

A distributed hash table is usually defined by its protocol; in other words, what it can do. Nodes and data in a distributed hash table are assigned unique[1] keys via a consistent hashing algorithm. To make it easier to grok the context, we will call the key associated with a node its ID and refer to nodes and their IDs interchangeably.

A DHT can perform the `lookup(key)`, `get(key)`, and `store(key, value)` operations.[2] The `lookup` operation returns the node responsible for a queried key, `get` returns the value stored with that key with the `store` function.

However, this is what a DHT *does*, viewing the DHT as a black box, not what a DHT *is* and needs to be implemented. Here, we open that black box for the first time and present those components. We show that Distributed Hash Tables are just Voronoi tessellations and Delaunay triangulation.

A. DHT Components

The following functions need to be defined in order for nodes to perform lookup operations and determine responsibility.

- **A `distance` function** - This measures distance in the overlay formed by the Distributed Hash Table. In most DHTs, the distance in the overlay has no correlation with real-world attributes. This is not necessarily the case with UrDHT (see Section V).
- **A `midpoint` function** - This calculates the minimally equidistant point between two given point. The midpoint is required for Delaunay triangulation calculation. In some spaces, such as Kademlia's XOR metric space, this can be tricky to calculate.
- **An `responsibility` definition** This defines the range of keys a node is responsible for. Not every DHT defines which node is responsible for particular keys in the same way. For example, nodes in Kademlia are responsible for the keys closest to themselves, while in Chord, nodes are responsible for the keys falling between themselves and the preceding node.

A DHT also needs a strategy to organize and maintain two lists of of other nodes in the network: *short peers* and *long peers*. Short peers are the set of peers that define the topology of the network and guarantee that greedy routing works.

Long peers allow the DHT to achieve a better than linear lookup time, typically $\log(n)$, where n is the size of the network.

[1]Unique with astronomically high probability, given a large enough consistent hash algorithm.

[2]There is typically a *delete(key)* operation defined too, but it is not strictly necessary.

Interestingly, despite the diversity of DHT topologies, all DHTs use the relatively the greedy routing algorithm (Algorithm 1):

Algorithm 1 The DHT Generic Routing algorithm

Given node n and a message being sent to key
function n.lookup(key)
if If $key \in n$'s range of responsibility **then**
    return n
end if
if One of n's short peers are responsible for key **then**
    return the responsible node
end if
$candidates$ = $short\_peers$ + $long\_peers$
$next \leftarrow \min(n.\text{distance}(candidates, key))$
return $next$.lookup(key)

If I, the node, am responsible for the key, I return myself. Otherwise, if I know who is responsible for this key, I return that node. Finally, if that is not the case, I forward this query to the node I know with shortest distance from the node to the desired key.[3]

Between individual DHTs, this algorithm might be implemented either recursively or iteratively. It will certainly have differences in how a node handles errors, such as how to handle connecting to a failed node which no longer exists. This algorithm may possibly be run in parallel, such as in Kademlia [7]. Despite this, the base greedy algorithm is always the same between implementations.

The final component is a consistent hashing function. This function must generate keys large enough to make the chances of a hash collision nigh impossible.

B. DHTs, Delaunay Triangulation, and Voronoi Tesselation

With the following components of a DHT defined above we can now show the relationship between DHTs and the primal-dual problems of Delaunay Triangulation and Voronoi Tessellation.

We can map a given node's ID to a point in a space, the range of keys a node is responsible for to that node's Voronoi region, and the set of short peers to the Delaunay triangulation. Thus, if we can calculate the Delaunay triangulation between nodes in a DHT, we have a generalized means of created the overlay network.

[3]This order matters, as some DHTs such as Chord are unidirectional.

This can be done with any algorithm that calculates the Delaunay Triangulation. However there is a plethora of DHTs with highly diverse spaces. The majority of well-known algorithms for solving this problem, such as Fortune's Sweepline [5], are designed to be run centralized.

Is there an algorithm we can use to efficiently calculate Delaunay Triangulations for a distributed system in an arbitrary space? We created an algorithm call the Distributed Greedy Voronoi Heuristic (DGVH), shown in Algorithm 2 and explained below [2].

Algorithm 2 Distributed Greedy Voronoi Heuristic

Given node n and its list of $candidates$.
Given the minimum $table\_size$
$short\_peers \leftarrow$ empty set that will contain n's one-hop peers
$long\_peers \leftarrow$ empty set that will contain n's peers further than one hop.
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**
    $m \leftarrow \texttt{midpoint}(n, c)$
    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 $long\_peers$
`handleLongPeers`($long\_peers$)

DGVH uses the midpoint to gauge which other nodes to use as its Delaunay triangulation [2]. Every maintenance cycle, nodes exchange their peer lists with **a current neighbor and then recalculate their neighbors.** A node creates a list of candidates by combining their peer list with their neighbor's peer list. This list of peers is then sorted from closest to furthest distance. The node then initializes a new peer list with the closest candidate. For each of the remaining candidates, the node calculates the midpoint between itself and the candidate. If new peer list does not contain any nodes closer to the midpoint than the candidate, the candidate is added to the new peer list. Otherwise, the candidate is set aside.

III. UrDHT

UrDHT is sectioned off into 3 broad components: database, network, and logic. Database handles file storage and network dictates the protocol for how nodes communicate. These components deal with the lower level mechanics of how files are stored on the network and how bits are transmitted through the network. The specifics are outside the scope of the paper, but can be found on the UrDHT Project [8].

The Logic component is what dictates the behavior of the DHT and the construction of the overlay network. It is composed of two parts: the DHT protocol and the space math.

The DHT protocol is the canonical operations that a DHT performs, while the space math is what effectively distinguishes one DHT from another. We discuss each in further detail below.

A. The DHT Protocol

The DHT protocol is the shared functionality between every single DHT. It consists of the node's information, the short peer list to define the overlay, the long peers that make efficient routing possible, and a bunch of other stuff.

B. The Space Math

The space math consists of the functions which define the DHTs topology.

The distance and midpoints functions are defined and discussed in detail in Section V

Put and Poll: The functions of `store` and `get` can be further abstracted otu to `put` and `poll`

This does X

IV. Implementing other DHTs

A. Implementing Chord and Ring-Based Topology

Ring topologies are fairly straightforward since they act as are one dimensional Voronoi Tesselations, splitting up what is effectively a modular number line among multiple nodes.

B. Implementing Kademlia and Other Tree Based Topologies

Trees are easy to embed in a hyperbolic space.

The largest complication in implementing UrKademlia is defining the exclusive or, or XOR, metric which is used for distance. This metric, while non-euclidean, is perfectly acceptable for calculating distance in Kademlia [7] However, XOR does not have an intuitive midpoint we could use for DGVH.

We found that

C. ZHT

ZHT [6] leads to an extremely trivial implementation in UrDHT.

V. DHTs in a Hyperbolic Topology

A. Okay, this is interesting, but why bother?

Hyperbolic spaces allow us to cleanly embed scale free graphs

B. Wait, Nodes can move in DHTs??

Yes, they can. There's no rule against it. In fact, it helps.

VI. Experiments

We are doing simulations. This is standard practice in both whitepapers and applications Citations are all major DHTs Citations for analysis works that use simulations Citations for applications of DHTs that were peer reviewed and accepted uing simulations

A. UrDHT Cohesion Euclidean Space

B. Cohesion in hyperbolic space

We showed in worked in DGVH, it should work in a hyperbolic space.

C. Performance of Chord on our network module vs UrChord

D. UrKademlia Works

VII. Future Work and Conclusions

References

[1] Redis. http://redis.io.

[2] Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, and Robert W Harrison. A distributed greedy heuristic for computing voronoi tessellations with applications towards peer-to-peer networks. In *Dependable Parallel, Distributed and Network-Centric Systems, 20th IEEE Workshop on*.

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[4] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[5] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153–174, 1987.

[6] Tonglin Li, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, and Ioan Raicu. Zht: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 775–787. IEEE, 2013.

[7] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[8] Andrew Rosen, Brendan Benshoof, Robert W Harrison, and Anu G. Bourgeois. Urdht. https://github.com/UrDHT/.