OPTIMISING STRUCTURED P2P NETWORKS FOR COMPLEX QUERIES

JAMIE ROBERT FURNESS

For the Degree of Doctor of Philosophy

Institute of Computing Science and Mathematics
University of Stirling

June 2014

DECLARATION

I, Jamie Robert Furness, hereby declare that this work has not been submitted for any other degree at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original.

June 2014

ABSTRACT

With network enabled consumer devices becoming increasingly popular, the number of connected devices and available services is growing considerably - with the number of connected devices estimated to surpass 15 billion devices by 2015. In this increasingly large and dynamic environment it is important that users have a comprehensive, yet efficient, mechanism to discover services.

Many existing wide-area service discovery mechanisms are centralised and do not scale to large numbers of users. Additionally, centralised services suffer from issues such as a single point of failure, high maintenance costs, and difficulty of management. As such, this Thesis seeks a Peer to Peer (P2P) approach.

Distributed Hash Tables (DHTs) are well known for their high scalability, financially low barrier of entry, and ability to self manage. They can be used to provide not just a platform on which peers can offer and consume services, but also as a means for users to discover such services.

Traditionally DHTs provide a distributed key-value store, with no search functionality. In recent years many P2P systems have been proposed providing support for a sub-set of complex query types, such as keyword search, range queries, and semantic search.

This Thesis presents a novel algorithm for performing any type of complex query, from keyword search, to complex regular expressions, to full-text search, over any structured P2P overlay. This is achieved by efficiently broadcasting the search query, allowing each peer to process the query locally, and then efficiently routing responses back to the originating peer. Through experimentation, this technique is shown to be successful when the network is stable, however performance degrades under high levels of network churn.

To address the issue of network churn, this Thesis proposes a number of enhancements which can be made to existing P2P overlays in order to improve the performance of both the existing DHT and the proposed algorithm. Through two case studies these enhancements are shown to improve not only the performance of the proposed algorithm under churn, but also the performance of traditional lookup operations in these networks.

LIST OF PUBLICATIONS

- Jamie Furness and Mario Kolberg. A Survey of Blind Search Techniques in Structured P2P networks. In *Proceedings of The 11th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, Liverpool, UK, 2010
- Jamie Furness and Mario Kolberg. Considering complex search techniques in DHTs under churn. *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 559–564, January 2011
- Jamie Furness and Mario Kolberg. Improving Wide Area P2P Service Discovery Mechanisms using Complex Queries. In Anand R. Prasad, John F. Buford, and Vijay K. Gurbani, editors, *Future Internet Services and Service Architectures*, chapter 9, pages 183–203. River Publishers, 2011
- Dom Schlienger, David Irvine, James Irvine, Mario Kolberg, Jamie Furness, Swee Keow Goo, Jorge Eliécer Gómez Gómez, and Velssy Hernandez Riaño. Works in Progress-A Survey, Cloud File Sharing, and Object Augmentation. *IEEE Pervasive Computing*, 11(2):96, 2012
- Jamie Furness, Mario Kolberg, and Marwan Fayed. An Evaluation of Chord and Pastry Models in OverSim. In *Modelling Symposium (EMS), 2013 European*, pages 509–513, 2013
- Jamie Furness, Farida Chowdhury, and Mario Kolberg. An Evaluation of EpiChord in OverSim. In *5th International Conference on Networks and Communication*. ACM Press, 2013

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

API	Application Programming Interface
AVTree	Attribute-Value Tree
CAN	Content Addressable Network
DHT	Distributed Hash Table
DKS	Distributed K-ary Search
DST	Distributed Segment Tree
EDRA	Event Detection and Report Algorithm
GPRS	General Packet Radio Service
INS	International Naming System
IP	Internet Protocol
KBR	Key-Based Routing
K-D Tree	K-Dimensional Tree
KISS-W	Keytoken-based Index and Search Scheme for Wild-cards
KSS	Keyword-Set Search System
LSI	Latent Semantic Indexing
MAC	Machine Address Code
ODISSEA	Open DIStributed Search Engine Architecture
P2P	Peer to Peer
RPC	Remote Procedure Calls
RPS	Recursive Partitioning Search
SFC	Space Filling Curve
SHA-1	Secure Hash Algorithm 1
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TTL	Time-To-Live
UDP	User Datagram Protocol
VSM	Vector Space Model

TERMINOLOGY

CHURN

The act of peers joining and departing an overlay is referred to as network churn. The more stable the network, the lower the rate of churn is said to be.

COMPLEX QUERY

In this Thesis the term complex query is used to describe any form of search query more complex than an exact match. This encapsulates query types such as keyword search, range queries, and complex regular expressions. This idea is expanded upon in Chapter 3.

OVERLAY

An overlay network is a network built on top of another network. In a peer to peer context an overlay network refers to a network of all participating peers, built at the application layer, on top of another, usually Internet Protocol (IP), network.

PEER

Within a P2P network there is no notion of clients or servers - only peers. Peers are always treated equally, and act as both clients and servers simultaneously. Also known as a node.

UNDERLAY

The underlay network is the native IP network which sits underneath an overlay network. As the IP network sits on top of the link layer it can span over multiple physical protocols, such as Ethernet, WiFi, and General Packet Radio Service (GPRS).

1

INTRODUCTION

As network enabled consumer devices are becoming more and more popular, the number of connected devices is growing considerably. In this large and dynamic environment it is important to have a comprehensive yet efficient means of service discovery.

Most existing wide-area service discovery mechanisms are centralised, bringing the issues of scalability, single points of failure, maintenance costs, censorship, and dictatorship.

In industry Peer to Peer (P2P) networks are becoming more and more popular for providing services. These services include file sharing (e.g. Gnutella, Bittorrent), media streaming (e.g. Spotify, Joost), digital currency (e.g. Bitcoin). Service discovery, on the other hand, is yet to take this leap.

Structured P2P networks allow storage an retrieval of data using an exact key. The current state of the art builds upon this to provide systems which can index data by (sometimes multiple) keywords, add support for range queries, or wildcard - however there is no system to date that can efficiently locate a data item without placing restrictions on the query types or structure of the network in which it runs. In general existing systems which want to perform complex queries flood the query within the network, hoping that it will eventually be forwarded to a node that contains the desired data, resulting in high traffic and low success rates.

In my opinion this lack of a flexible yet efficient query system is one of the main issues holding back the use of P2P networks for service discovery. In this thesis a generic system for supporting complex queries in Structured P2P networks is designed, which allows for comprehensive, yet efficient, service discovery on top of a Structured P2P network through the use of efficient broadcasting. The generic aspect is important, as this system should not dictate the structure of the P2P network used - rather the system should independantly sit on top of the P2P network.

The system presented in this thesis is built on top of a Distributed Hash Table (DHT). A DHT is a data storage layer built on top of a Structured P2P network in which data items are hashed, and stored within the network using the same algorithm that places nodes within the network. As this system is based on a DHT it is scalable, resilient, avoids censorship, and has no single controlling authority.

Taking these properties in to consideration, and the requirements of supporting complex queries in a large networks of heterogeneous consumer devices, the objectives of the Thesis will now be described.

1.1 OBJECTIVES

The main objectives of this work are as follows:

- Design a generic approach for routing complex queries in different types of DHT, without storing extra information but that is also applicable to many DHT types. This approach should solve all complex query types, rather than just a subset.
- Design an optimal data replication strategy for use with the above approach which optimises placement of replica for success and performance of complex queries, without having a negative impact on regular DHT operations.
- Design optimisations that are applicable to many DHT types which optimise for success and performance of complex queries.

1.2 CONTRIBUTIONS

In order to understand the entire scope of the problems associated with complex queries within DHTs, this Thesis examines what is meant by the term complex query, and the requirements to support such a query type. In addition, a survey of existing solutions is performed, identifying which types of complex queries are supported by each, and where gaps exist in the state of the art.

The following list of contributions are made throughout this Thesis, as outcomes of the objectives described in Section 1.1:

- **A Blind-Search system**

  A Blind-Search system is designed, which brings together the state of the art in broadcast techniques from different overlay networks, and adapts them for use in a search system. This addresses the first objective of this Thesis, providing a system that works across different DHT types, and supports all complex query types.

- **Novel query response routing algorithm**

  To complete the Blind-Search system, a novel query response routing algorithm is presented. This algorithm uses reverse tree routing and message collation to reduce load, especially in searches with a large number of positive responses.

- **Advancements in DHT data replication**

  The state of the art in DHT data replication algorithms is advanced through the analysis of Symmetric replication and introduction of Replica Teams, the idea of grouping replica in a replication strategy such as Symmetric Replication, to get the advantages of Symmetric Replication, with the lower maintenance cost of Neighbour Replication.

- **Advancements in maintenance algorithms**

  Optimisations to the routing table in DHTs are explored and evaluated. Multiple generic optimisations are presented, that can be applied to many DHT routing table types, and it is shown that they can improve success rate, performance, and reduce maintenance traffic.

In summary, this thesis presents a Blind-Search system with efficient query response routing, that achieves a high success rate and high performance through use of an novel, more appropriate, choice of data replication, and optimised DHT routing table and maintenance algorithms.

1.3 STRUCTURE OF THE THESIS

This Thesis is composed of seven chapters:

- **Chapter 1 - Introduction**

  Introduces the thesis, providing context, objectives and achievements.

- **Chapter 2 - P2P Overlays**

  Introduces the background to P2P networks; unstructured, structured, hybrid, multi-hop, single-hop, and variable-hop. This background is necessary to understand the design of DHTs, and the challenges associated with complex queries.

- **Chapter 3 - Complex Queries**

  Introduces the background to Complex Queries. This includes a breakdown of what can be considered a complex query, and a full review of existing systems claiming to provide support for complex queries, or a subset of complex query types.

  The work presented in this chapter provides motivation and a base for the proposed system presented in Chapter 4.

- **Chapter 4 - Blind-Search**

  Describes the Blind-Search system designed. Starting with the design criteria and the related work that it is built upon, then following with a novel approach for response routing and observations of load balance and performance under churn.

  This work in this chapter provides the core of the proposed Blind-Search system, which the following chapter builds upon through optimisations to the DHT.

- **Chapter 5 - Optimising DHTs for Blind-Search**

  Explores a collection of different optimisations that can be applied to the DHT, to optimise for Blind-Search success and performance.

  Starting with increasing routing table size to increase branching factor, moving onto increasing routing table accuracy to decrease failures, and finishing with the choice of data replication strategy to best locate data such that failures in the broadcast tree have minimal effect. In this chapter the concept of Replica Teams is introduced, providing an adaption of the symmetric data replication strategy with a lower maintenance cost in the majority of DHT types.

- **Chapter 6 - Experimentation**

  Presents simulation results relating to the previous work discussed in this Thesis. This chapter starts by describing the simulation environment used, and validating the environment as well as the models in use.

  The Blind-Search system is then evaluated, including performance under high levels of churn. The optimised data replication, described in Chapter 5 is then applied, and results compared.

  Results for further optimisations to the DHT, including increasing routing table size and accuracy are also presented.

- **Chapter 7 - Conclusions**

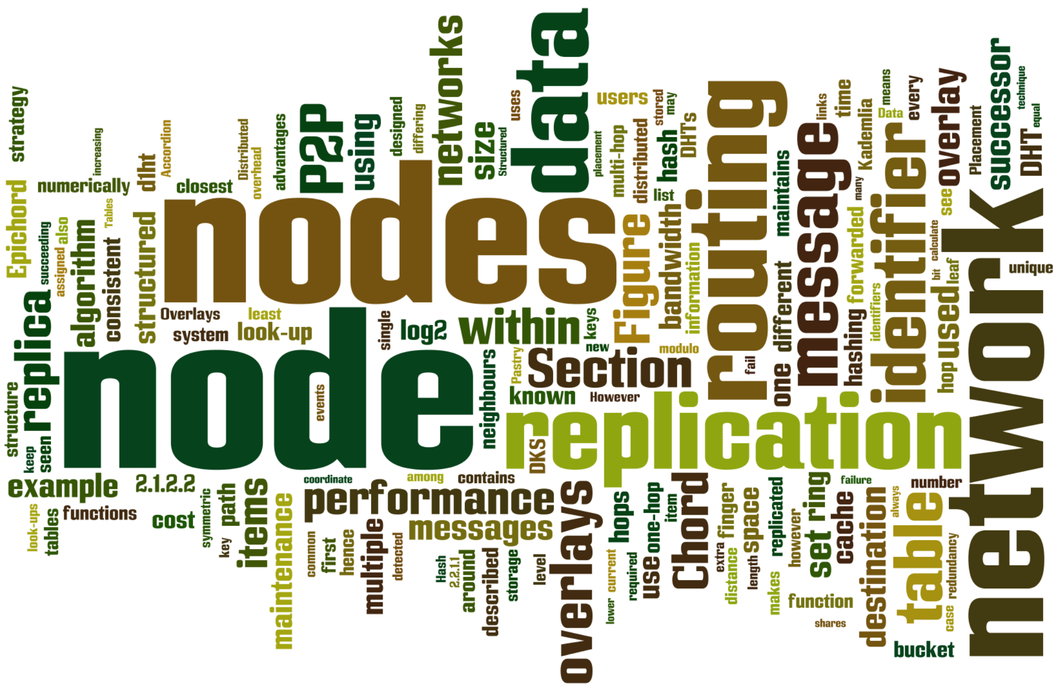
  Concludes the work presented in this Thesis. The strengths, and weaknesses, of the work are summarised, and the extent to which the original research objectives have been met is reviewed. Possible future work and open questions are discussed.

- **Appendix A**

  Includes two referenced papers validating the OverSim models for Chord, Pastry, and EpiChord.

2

P2P OVERLAYS



This chapter firstly discusses the background of P2P overlay networks, focusing on structured overlays, and in particular DHTs. Secondly this chapter looks at the current state of the art for structured P2P overlays.

2.1 BACKGROUND

The concept of P2P networks is simple - instead of all users requesting data from a designated set of nodes, the data is distributed among all users. All nodes within the network are treated as equal, and expected to fail or leave at any time.

A P2P network architecture has multiple advantages over traditional client-server architecture:

- Resource (cost) savings - As data and computation are distributed among users rather than concentrated on dedicated machines, the responsibility and cost of operation is distributed to the users.
- Scalability - Since data and computation are distributed to users of the network, as more users join the network so do more resources.
- Low barrier of entry - Due to the resource savings identified above, and the inherent scalability, the upfront setup cost of a P2P network is far lower than that of traditional architectures.
- Redundancy - As all nodes within a network are treated equally, there is no reliance on any single point. Due to the assumption that any node can fail or leave at any time, by design, data is automatically replicated at multiple locations. P2P networks have no single point of failure.
- Lack of authority - By having all nodes treated equally, and data distributed among all users, there is no single authority which can be considered in charge. This can be considered an advantage in certain domains, both for privacy and legality.

However, these advantages are not totally free - P2P networks have the following assumptions baked in to their design:

- Resource ownership - As mentioned above, data and computing is distributed among users of the network. This clearly complicates the question of trust considerably. Within a network there should be no assumption that a node is trustworthy.
- Node reliability - All machines can fail. Consumer machines generally fail more than dedicated servers, and even if they do not, it cannot be assumed they will shut down correctly or leave the network gracefully.
- Heterogeneity - P2P networks are heterogenous. Users have machines with different processing capabilities, amounts of storage, and bandwidth. Not only do the nodes differ from each other, but the resources on a single node are likely to differ over time - as other processes on the machine compete for them.

A large number of P2P overlays have been designed, with differing trade-offs and levels of performance. At the highest level P2P these can be divided in to two categories. These are now discussed.

2.1.1 *Unstructured Overlays*

An unstructured P2P overlay defines no, or at least a very loose set of, rules over how the overlay is formed. Links between nodes are established arbitrarily in an ad-hoc fashion. Such a network is easy to construct, as new nodes can simply copy existing routing tables from other nodes, and then over time form their own links within the network.

Because of the inherit lack of structure within unstructured overlays, they can only provide probabilistic routing and performance estimates.

In general this Thesis focuses primarily on structured overlays, except in Chapter 4 where inspiration in drawn from search in unstructured overlays.

2.1.2 *Structured Overlays*

Structured P2P overlays define a strict set of rules over how the overlay is formed and how nodes are addressed. The majority of structured P2P overlays make use of consistent hashing for node addressing. This strict organisation makes structured overlays more complex and increases the amount of maintenance required by nodes to continually conform to the rules while other nodes join, leave, and move around the overlay. However, this extra cost buys a few extra features:

- Deterministic routing - By making use of the structure inherit in the overlay, messages can always be forwarded closer to their destination on each hop. This provides efficient message delivery, with no duplicated or redundant messages generated.
- Guaranteed performance - Given the size of the network, it is possible to calculate the best, average, and worst case performance for message routing.

Generally most proposed overlays provide three distinct functions: message routing, maintenance, and data storage/retrieval. While conceptually very different functions, the boundary between them is often blurred in reality, with authors describing an overlay structure, routing algorithm, maintenance algorithm, and data storage algorithm as a single system. In this thesis the three functions are separated as much as possible, both for clarity, and because in the majority of cases there is no need for the data storage algorithm to be tied to a specific overlay - they are, and should be, interchangeable.

ROUTING

The routing algorithm is responsible for directing a message from a source node to a destination. This puts restrictions on the structure and minimum size of the routing table.

MAINTENANCE

> The maintenance algorithm is responsible for ensuring node failures are detected, and the routing tables are kept as full and up-to-date as possible.

STORAGE

> The storage algorithm is responsible for placing data items within the DHT, and determining which nodes should be contacted to find a specific result.

The majority of structured P2P networks which provides data storage and retrieval functionality fall within the category of DHTs.

2.1.2.1 *Distributed Hash Tables*

Distributed Hash Tables (DHTs) expose a hash table interface built on-top of a structured P2P overlay. In a DHT, key-value pairs are stored within the network, allowing any node to efficiently retrieve the value associated with a given key by routing a message to the storing node.

As previously noted, it is an assumption of P2P networks that nodes will fail. As such, it is important that data stored within a DHT is replicated and stored at multiple locations in the network.

2.1.2.2 *Data replication in Distributed Hash Tables*

There are multiple ways in which data can be replicated within a DHT and still allow for efficient storage and retrieval operations. This section describes the most popular three.

NEIGHBOUR REPLICATION Neighbour replication is the term used to describe different replication strategies, such as successor replication and leaf set replication, in which data is replicated at neighbours of the original node, as can be seen in Figure 2.1 and Figure 2.2.

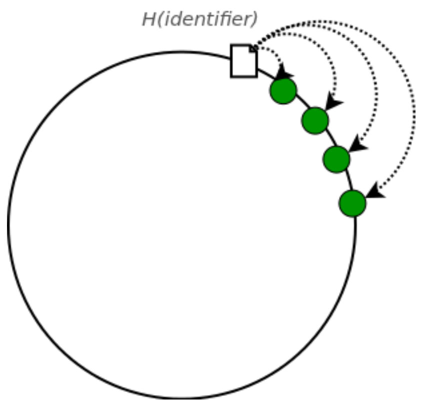


Figure 2.1: Placement of replica in the successor replication strategy.

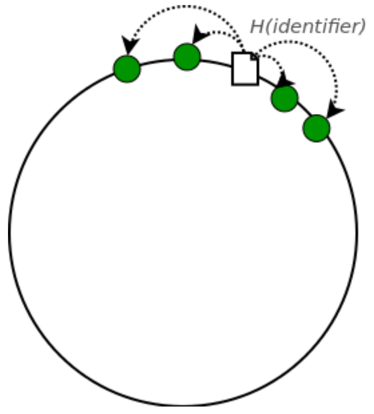


Figure 2.2: Placement of replica in the leaf replication strategy.

Neighbour replication is the default used by the majority of existing overlays. Neighbour replication is cheap to maintain, since in most DHTs nodes already store and maintain a list of their neighbours. The main advantage to neighbour replication is that data will usually be in the right location even post failure. For example, in many overlays when a node leaves the network the responsibility falls to its successor - since data is replicated at the nodes successors it will already be in the right place.

multi-publication replication Multi-publication replication is different from neighbour replication in that the data is published at multiple keys within the network, each of which is of equal importance. There are three proposed strategies for implementing multi-publication replication [39], described below.

Multiple hash functions To generate R unique keys R different hash functions are used. How replica are spread will depend on the hashing functions chosen. An example of replica placement when using symmetric replication can be seen in Figure 2.3.

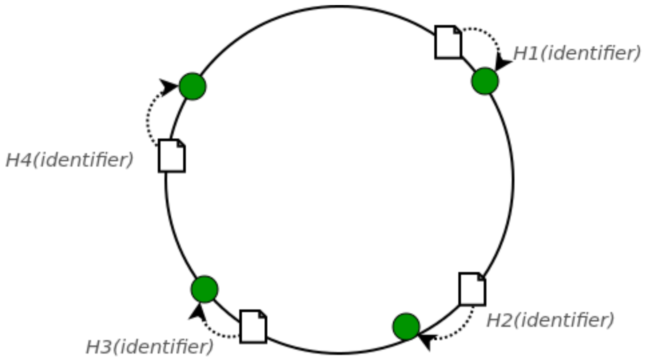


Figure 2.3: Placement of replica in the multiple hash function replication strategy.

Correlated hashing To generate R unique keys the numbers $r = 0 \ldots R-1$ are prepended to the original key before hashing. Assuming consistent hashing, replica should be spread evenly throughout the network. An example of replica placement when using symmetric replication can be seen in Figure 2.4.

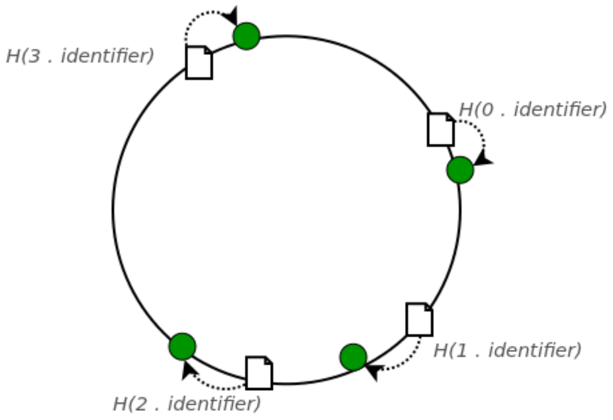


Figure 2.4: Placement of replica in the correlated hashing replication strategy.

Symmetric replication To generate R unique keys, first the hash h is calculated, then the keys are defined as $(h + (r * \frac{N}{R}))\%N$ where $r = 0 \ldots R$ and N is the size of the key space. Using this formula, replica will be spread perfectly evenly throughout the network. An example of replica placement when using symmetric replication can be seen in Figure 5.2.

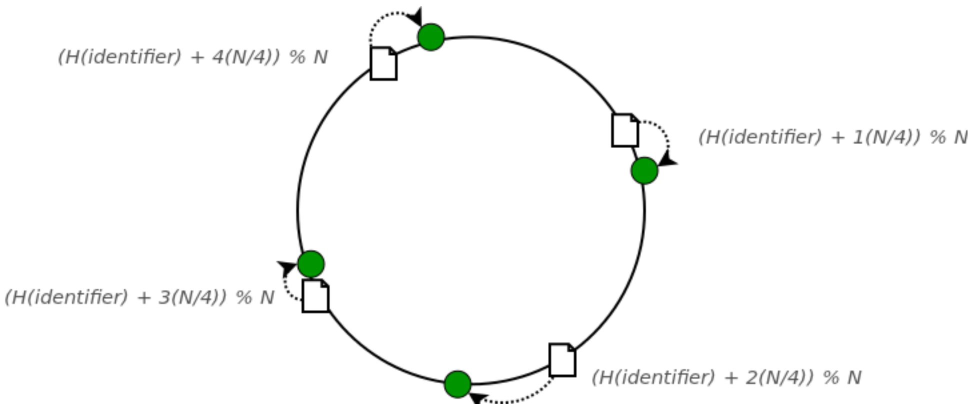

(H(identifier) + 4(N/4)) % N
(H(identifier) + 1(N/4)) % N
(H(identifier) + 3(N/4)) % N
(H(identifier) + 2(N/4)) % N

Figure 2.5: Placement of replica in the symmetric replication strategy.

The ability for nodes to calculate the location of all replica has various advantages. For example, the querying node then has the ability to send multiple parallel requests for the data, or even choose a replicating node in its local proximity by choosing the numerically closest replica key.

PATH REPLICATION Path replication replicates a data item along the path which the look-up message was forwarded. As messages in a DHT are routed towards the destination from different sources they are likely to converge at a node before the actual destination node, hence path replication shortens the routes and accelerates the look-ups. However, path replication is a lot less structured compared to the other approaches. For example, the number of replica depends on the route length and the placement depends on which nodes are requesting data. Removing or modifying data items is also problematic as it is not possible to calculate which nodes contain replica. A data items primary node would be required to maintain a list of all replicating nodes, requiring additional node state.

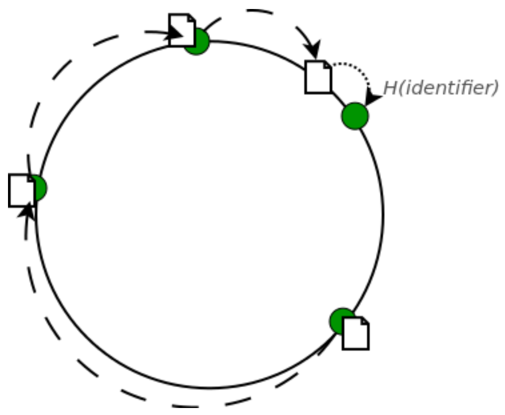


Figure 2.6: Placement of replica in the path replication strategy.

2.1.3 *Hybrid Overlays*

In hybrid P2P overlays the concept of equality between peers is sacrificed, and nodes are categorised in to multiple classes. There are a few common architectures for hybrid overlays:

- Central server - An example of this class is the BitTorrent protocol, in which a central server is used to mediate connections between peers. This simplifies the protocol considerably, removing all complexity associated with publishing and locating data. This simplification, however, comes at a huge cost, with the network losing its advantages of scalability, redundancy, and lack of authority.
- Super-node - In a super-node network, higher capacity nodes are designated as super-nodes and used to relay requests for lower capacity nodes. For example, a super-node may hold indexes for its connected children, or communication may be proxied through a super-node.

This thesis does not consider hybrid overlays a solution to complex search, as they sacrifice the main advantages of a P2P overlay.

2.2 STATE OF THE ART

Structured P2P overlays can be categorised in to three categories, based on the bound on the number of hops required to reach a desired node: Multi-hop (Section 2.2.1), Single-hop (Section 2.2.2), and Variable-hop (Section 2.2.3).

2.2.1 *Multi-hop Overlays*

Multi-hop overlays are, as the name suggests, structured overlays in which messages typically require multiple hops to reach their destination. Commonly multi-hop overlays have a routing table size of $O(\log(N))$ and can guarantee delivery of a message within a maximum of $O(\log(N))$ hops. Multi-hop overlays are designed to reduce the size of the routing table required by each node, and hence reduce the amount of maintenance traffic required to keep routing tables up-to-date. In other words, performance is traded off against maintenance overhead. The most relevant multi-hop networks are discussed below.

2.2.1.1 *Chord*

In Chord [59] a consistent hash function, such as Secure Hash Algorithm 1 (SHA-1), is used to assign each node and data item an m-bit identifier. Nodes are then ordered numerically to form a ring structure modulo 2^m. Data items are assigned to the first node whose identifier is equal to or follows in the ring.

In a network of size N each node maintains routing state information for $O(\log_2(N))$ other nodes, specifically the k nodes succeeding it, known as the successors, and a set of what are known as finger nodes. The finger nodes are chosen at logarithmically increasing distance around the ring, the i^{th} entry in the table at node n contains the identity of the first node that succeeds n by at least 2^{i-1} $(i \geqslant 1)$. This means that nodes have a more complete view of the area nearby, with less links to far away nodes in the network. As a message is forwarded around the network, the closer it becomes to the destination the more likely nodes are to have a link to the destination node. An example Chord network showing the choice of finger table nodes can be seen in Figure 2.7.

To route a message to the destination identifier d within a Chord network, the originating node selects the finger node whose identifier most immediately precedes d and asks that node for the identifier of the node it knows closest to d. By repeating this process the originating node learns the identifier of the node preceding d. Since every node maintains a link to its successor, the node preceding d must know the node whose identifier follows d, and hence the message can be delivered to the node responsible for d. Using this algorithm the maximum path length should be $O(\log_2(N))$ hops.

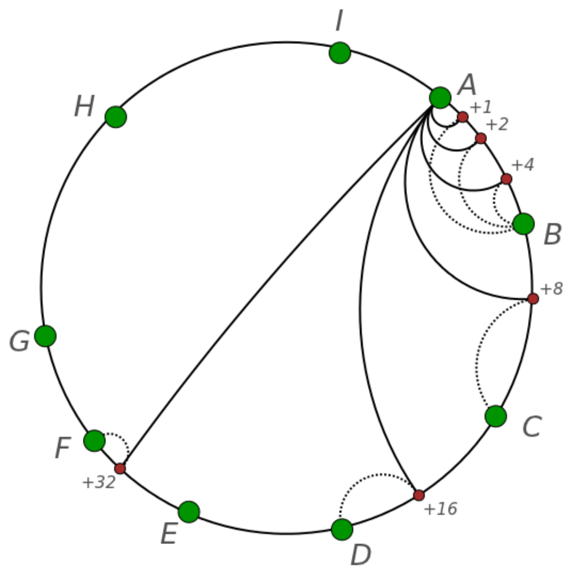


Figure 2.7: A Chord network, showing the choice of finger nodes for node A.

To keep the routing tables up-to-date when nodes join and depart from the network, Chord makes use of periodic maintenance. When a node failure is detected, the detecting node updates its successor list instantly, ensuring look-ups can always succeed eventually. However, to update the finger table, nodes probe their fingers every D seconds, and seek replacement nodes if necessary. When D is high, or the network is unstable, the finger table can become out-of-date for a period of time. This form of periodic maintenance cuts down on the maintenance cost, however results in less accurate routing tables and hence degraded performance when a network is unstable.

Chord recommends the use of successor replication (Section 2.1.2.2) for redundancy, in which data is replicated at the $\leqslant k$ nodes succeeding the data item.

Based on Chord, Koorde [37] is a P2P overlay which makes use of a De Bruijn graph.

2.2.1.2 *Kademlia*

Kademlia is another structured P2P overlay, designed by Petar Maymounkov and David Mazires [45]. Kademlia uses a $160 - \mathrm{bit}$ consistent hashing function to assign identifiers to nodes and data items, and an XOR (exclusive or) function to calculate distance between nodes, ensuring that during routing every hop moves a message closer to its destination.

Kademlias routing table is formed of b buckets, where b is the bit-length of the identifier used (i.e. 160). Every bucket contains nodes at a given distance, for example nodes in bucket d have a distance with the most significant bit d. This results in exponentially increasing numbers of candidate nodes for each bucket (half the nodes in a network will have a differing b bit, of those that match, half of them will have a differing $b - 1$ bit...).

As messages are received from nodes in the network they are added to the appropriate bucket, assuming the bucket isn't full. If the bucket is full, the older nodes in the bucket are preferred, and the new discarded. This decision was based on the observation that the longer a node exists in a network, the higher chance there is it will continue to exist in the network [60]. Kademlia specifies a parameter k, used to define the maximum size of a bucket. Because of this, Kademlia's buckets are often referred to as k-buckets.

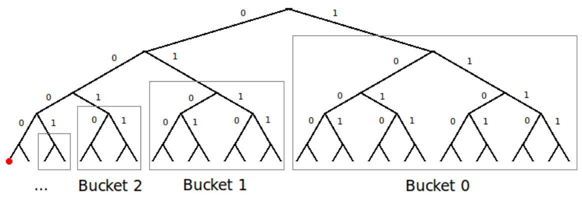


Figure 2.8: A Kademlia network, with 4 buckets.

Kademlia is one of the more widely used DHT in reality, with open-source implementations available in many major languages. It is used by the Kad Network, BitTorrent for its DHT-based implementation, and Gnutella (which moved away from flooding and towards a DHT-based approach is more recent versions).

2.2.1.3 *Pastry*

Pastry [51] also assumes a circular identifier space. Each node and data item is assigned a 128-bit identifier using either a consistent hash function or uniform random number generator, and ordered in a circular namespace modulo 2^{128}. Data items are stored at the node whose identifier is numerically closest to the data items identifier.

Each Pastry node maintains a routing table, a neighbourhood set, and a leaf set (otherwise known as the namespace set). In a network of size N, using identifiers with base 2^B, each nodes routing table is designed with $\log_B(N)$ rows, where each row holds $B - 1$ entries. All the entries at row r of the routing table each refer to a node whose identifier shares the current nodes identifier in the first r digits, but whose $(r+1)^{th}$ digit does not match that of the current node.

The neighbourhood set contains links to the m closest nodes, and the leaf set contains the k nodes whose identifiers are closest and centred around the local nodes identifier.

To route a message to the destination identifier d within a Pastry network, the originating node first checks if d falls in the range of the leaf set. If so, the message is forwarded directly to the destination. If not, then the routing table is used and the message is forwarded to a node who shares a common prefix with d by at least one more digit than the current node. If no such node exists then the message is forwarded to a node who shares a common prefix with d at least as long as the current node, but whose identifier is numerically closer. By repetition the message is eventually forwarded to the node whose identifier is numerically closest to d, in other words the node responsible for d. A message will always reach its destination in no more than $O(\log_{2^B}(N))$ hops.

When a node failure is detected, the detecting node updates its routing table, neighbourhood set, and leaf set instantly. This means routing tables are kept accurate, however as more failures are detected the maintenance overhead increases. In some cases this increased traffic could cause other nodes to become overloaded, and hence also detected as failed, further increasing the overhead.

Leaf set replication (Section 2.1.2.2) is performed by Pastry. A data item i is replicated with $\frac{k}{2}$ replica preceding i and $\frac{k}{2}$ replica succeeding i.

2.2.1.4 *CAN*

Content Addressable Network (CAN) [48] is a rather unique type of overlay, designed around a virtual d-dimensional Cartesian coordinate space. The entire coordinate space is dynamically partitioned among all the nodes in the system such that every node owns an individual zone. Data items are mapped onto a point in the coordinate space using a consistent hash function, and the node responsible is the node whose zone contains that coordinate.

In a CAN with dimension d, each node has at least $2d$ neighbours with which links must be maintained; one to move forward in dimension d and one to move backwards. An example CAN network can be seen in Figure 2.9.

Routing a message with a CAN network is simply a case of greedily forwarding the message to the neighbour with coordinates closest to the destination identifier. In a network of size N this results in a path length of $O((d/4)(n^{1/d}))$ hops.

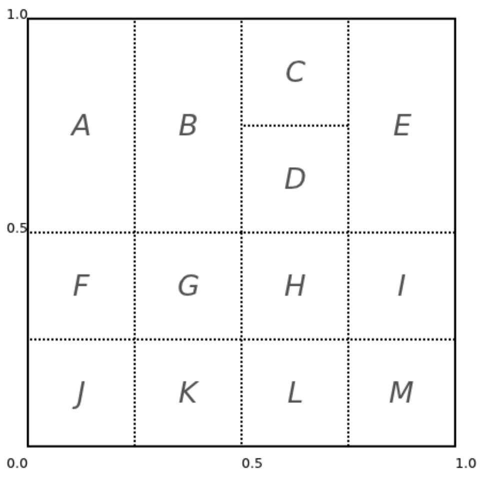


Figure 2.9: A CAN. As the zones are not all equal size it is said to be imperfectly partitioned.

In CAN, nodes periodically send updates to their neighbours indicating they are still within the network. If a node detects a failure it will take over the failed nodes zone, and update its routing table accordingly.

For redundancy CAN suggests the use of multiple hash functions, as described in Section 2.1.2.2.

2.2.1.5 *DKS*

In [1] a DHT known as Distributed K-ary Search (DKS) (sometimes referred to as $DKS(N, k, f)$) is presented. In a DKS network N is the maximum number of nodes that can be within the system, such that $N = k^L$ $(k \geqslant 2)$. f defines the fault tolerance of the system and is typically a small factor of k.

A DKS network is organised as a ring, modulo N. Like other DHTs, nodes and data items are assigned unique identifiers using a consistent hash function. Data items are stored at the first node succeeding the data items identifier. Each node in the network maintains a link to its predecessor p and a routing table with $O(\log_k(N))$ levels numbered from 1 to L. At each level l a node n has a view of the identifier space defined as $[n, n \oplus \frac{N}{k^{l-1}})$. This means that at level one, the view consists of the whole identifier space, and at any other level, one k^{th} of the previous level is considered. At every level the view is partitioned into k equally-sized intervals. This method for partitioning the identifier space is known as the k-ary principle, and is described fully in [23].

Using DKS a message can be routed in at most $O(\log_k(N))$ hops. To deliver a message d a node forwards it to the first node in its routing table that succeeds d on the identifier circle. The receiving node checks if they are responsible for d, and if so returns the data item. Otherwise the message is forwarded onwards.

To keep routing tables up-to-date DKS uses a correction-on-use technique. By embedding extra information in messages, when node n' receives a message from node n, node n' can determine whether the routing information used by node n when sending the message was correct or not. If n' finds out the routing information was incorrect, it will immediately inform node n. Using this technique any out-of-date routing information is eventually corrected.

For redundancy DKS makes use of symmetric replication, as described in Section 2.1.2.2.

2.2.2 *Single-hop Overlays*

Single-hop networks (sometimes referred to as one-hop networks) are structured peer-to-peer networks in which messages can usually be delivered directly from the source to the destination without the need to route via intermediate nodes. This means nodes must keep larger routing tables, and hence there is a higher maintenance cost than in multi-hop networks. However, as a single hop in a peer-to-peer network usually means a number of hops in the underlying IP network, the ability to transmit messages directly provides a much lower message latency.

2.2.2.1 *D1HT*

In [46] a one-hop DHT called D1HT is presented, arguing that by efficiently propagating events one-hop performance can be achieved with a reasonable amount of maintenance traffic. Similarly to Chord, D1HT uses a ring structure modulo 2^m, with both nodes and data items assigned an m-bit identifier using consistent hashing, then ordered numerically around the ring. Also like Chord, data items are assigned to the key's successor.

Every node in the D1HT network maintains a full routing table, containing links to every other node within the network. As such routing of messages is trivial, and can be achieved with one hop.

To keep the routing table up-to-date, D1HT uses a maintenance protocol known as Event Detection and Report Algorithm (EDRA), which is able to notify an event to the whole system in logarithmic time and yet have good load-balance properties and low bandwidth overhead.

To disseminate information about events, each node n sends up to ρ propagation messages at each θ seconds time interval, where $\rho = \log_2(N)$ and θ is a system parameter. Each message has a Time-To-Live (TTL) counter l in the range $[0 \ldots \rho - 1)$ and will be sent to $succ(p, 2^l)$. In other words the messages are sent to nodes at logarithmically increasing distance from n. In each message is both information about any events detected by n, as well as all events brought to n by messages with a higher TTL, received in the last θ seconds. To avoid messages wrapping around the ring, before sending a message to k, node n will discharge all events related to any node within the range $[n, k]$.

An example of event propagation using EDRA can be seen in Figure 2.10.

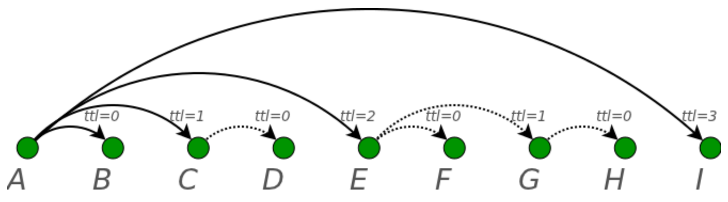


Figure 2.10: Event propagation using EDRA in a D1HT system. Note that a D1HT network uses a ring structure, but is shown linearly for ease of presentation.

D1HT makes use of successor replication, as described in Section 2.1.2.2, for redundancy, replicating data items on the following $O(\log_2(N))$ nodes around the ring.

2.2.2.2 *Epichord*

Epichord [41] is a DHT algorithm which can achieve one-hop look-up performance under look-up intensive workloads, and at worst case $O(\log_2(N))$ hop, such as offered in many multi-hop networks. As the name suggests, Epichord is based on the Chord DHT (see Section 2.2.1.1), and shares its ring structure as well as mapping of data items to the succeeding node. In addition to maintaining a successor list of k nodes, Epichord also maintains a predecessor list of k nodes. Instead of maintaining a finger table, as in Chord, Epichord maintains a cache of nodes. Nodes update their cache by observing look-up traffic, and add an entry any time they are queried by a node not already in the cache. Nodes in the cache each have a timeout, resulting in stale nodes being removed.

To guarantee worst case look-up performance of $O(\log_2(N))$ the address space is divided into two sets of exponentially smaller slices. Each node maintains their cache such that every slice contains at least $\frac{j}{1-\gamma}$ cache entries at all times, where j is a network parameter and γ is a local estimate of the probability that a cache entry is out-of-date. Nodes periodically check their cache slices to ensure that there are sufficient unexpired cache entries.

In general terms Epichord can be thought of as Chord with a cache of extra node addresses. As such the routing algorithm is the same as that in Chord (see Section 2.2.1.1). With a well populated cache this results in look-up performance of one hop. Under high churn the performance drops to that of Chord, $O(\log_2(N))$ hops in the worst case.

In [41] there is no indication as to what replication strategy Epichord uses, however as it is based on Chord common implementations use successor replication, described in Section 2.1.2.2.

2.2.3 *Variable-hop Overlays*

Variable-hop networks are a class in between multi-hop and one-hop networks, in which nodes may have single-hop performance or may have multi-hop performance. Such networks can have advantages in heterogeneous environments, allowing nodes which will benefit from decreased latency, and have the available bandwidth, to achieve one-hop performance, without imposing extra maintenance overhead on nodes which either do not require or can not afford single-hop performance.

2.2.3.1 *Accordion*

In [42] a variable-hop DHT is proposed which is designed to automatically tune routing table size to fit within a specified bandwidth budget. Accordion borrows Chords (see Section 2.2.1.1) protocols for maintaining a successor list, and like other DHTs, assigns unique m-bit identifiers to nodes and data items using consistent hashing, then orders them numerically in a ring modulo 2^m. Each node learns of new neighbours as a side effect of ordinary look-ups, but selects them so that the density of its neighbours is inversely proportional to their distance in the identifier space. In practise look-up keys are not always distributed uniformly, and thus Accordion devotes a small amount of its bandwidth budget to actively exploring for new neighbours. This allows Accordion to vary the routing table size while still providing the same worst-case look-up of $O(\log_2(N))$ as most multi-hop DHTs. The rate with which a node learns of new neighbours is determined by its bandwidth budget, and nodes are removed from the routing table after a timeout, hence the routing table size is determined by the equilibrium between the learning and removal processes.

The routing algorithm is also borrowed from Chord (see Section 2.2.1.1). Depending on the bandwidth budget and churn rate the path length can vary from $O(\log_2(N))$ to one hop.

There is no replication technique specified by Accordion, however as it borrows various protocols from Chord it is assumed that it makes use of successor replication, described in Section 2.1.2.2.

2.2.3.2 *Chameleon*

A two-tier variable-hop overlay known as Chameleon is presented in [8]. In Chameleon nodes assess their available bandwidth to determine whether they should be classed as a *high* or *low* bandwidth node. High bandwidth nodes, known as *H-nodes*, operate using the EDRA algorithm from D1HT (see Section 2.2.2.1), offering one-hop performance. Low bandwidth nodes, known as *L-nodes*, operate using the Epichord algorithm (see Section 2.2.2.2), offering lower performance at a reduced cost.

For *H-nodes* look-ups are trivial and are completed in one-hop. For *L-nodes* the look-up algorithm is borrowed from Epichord (see Section 2.2.2.2), as such look-ups are completed in between one hop and $O(\log_2(N))$ hops.

Chameleon does not specify which replication technique should be used, however as it borrows from both D1HT and Epichord, common implementations use successor replication, described in Section 2.1.2.2.

2.3 SUMMARY

In summary, this Chapter has shown that P2P networks come in three main types: Unstructured, Structured, and Hybrid. This Chapter (and rest of Thesis) focused on Structured P2P networks, of which there are many different types, such as ring-based, prefix-based, De Bruijn graph based, and Cartesian based. These different types of network have very different structures, and vary the tradeoff between routing table size and average lookup path length. This implies that designing a generic system to support Complex Queries, as one of the primary objectives of this Thesis, is a hard problem to solve, and will require a generic system that can make use of multiple algorithms, rather than a single generic algorithm.

This Chapter also introduced the use of DHTs as a data storage layer, along with the most common data replication strategies used and their main advantages: Neighbour replication supports automatic failover by clustering replica together, multi-publication replication allows any node within the network to locate all replica by publishing them with a uniform distribution using a well known algorithm, and path replication automatically replicates data where it is used the most.

It is noted that although a DHT consists of multiple different layers with distinct concerns, this boundary is often blurred, with some overlays dictating a specific storage algorithm and replication strategy, and others concentrating solely on the routing algorithm.

3

COMPLEX QUERIES

This chapter explores what is meant by the term *complex query*, and how such queries are supported in existing P2P systems.

3.1 BACKGROUND

Complex query is a vague term which is used to describe any form of search query more complex than an exact match. To properly understand the term the differing forms of search queries are described below (using a search for a BMW car as an example):

EXACT-MATCH

An exact-match lookup allows retrieval of a document only via the exact key under which it was initially indexed.

key: bmw320d

matching query: bmw320d

KEYWORD

Documents can be indexed using a limited set of keywords. A query for any of these keywords will return the original document.

keywords: bmw, 320, diesel

matching query: bmw

RANGE

Range queries are a type of query which look for any document, possibly indexed by keywords, which lies within a given range.

keywords: bmw, 320, diesel

matching query: 310-330

SEMANTIC

Semantic search is a content-based search, in which queries are expressed in natural language instead of keywords. The goal in semantic search is to use semantics, the meaning of words, to return results which are relevant to the search terms, without simply matching keywords.

keywords: bmw, 320, diesel

matching query: car

WILD-CARD

The term wild-card search is used to describe a search in which part of the search term is unknown. Note that wild-card search is related to range queries, but not exactly the same. For example, assuming keywords were restricted to A . . . Z, the wild-card search for ACM* can be converted to a range query for everything between ACM and ACN. However other forms, such as *ACM, A*M, or *ACM* are difficult to map to a range query [35].

keywords: bmw, 320, diesel

matching query: di*l

FULL-TEXT

Full text search refers to a technique for examining all words in all documents, to try match the search term supplied by the user. While this type of search may initially sound similar to the keyword search, keyword search solutions assume a limited number of keywords and do not scale well above a certain limit.

text: the bmw 320d touring is a car that removes ambiguity from life

matching query: car that removes ambiguity

REGULAR EXPRESSION

Although not often mentioned, in some circumstances it may be desirable to have the ability to search for data using regular expressions. A regular expression is an extremely powerful method for string matching.

text: bmw320d

matching query: bmw[0-9]+[di]?

If DHT based systems are to become more prevalent, they need to provide support for a much more comprehensive set of query types. Just now almost all real-world P2P systems are either unstructured, or hybrids of a DHT combined with a centralised search index.

3.1.1 *Unstructured Overlays*

In unstructured P2P overlays routing algorithms have no network structure to assist in request routing. Without a way for a message to answer "how far am I currently from my target", it has no way to determine if it is moving in the right direction or not. Because of this, querying in unstructured overlays tends to be probabilistic rather than deterministic, with most techniques looking at methods for improving the probability of finding the requested data, or reducing the cost of doing so.

The most basic approach to search in any P2P network is flooding. Flooding is the most simple approach imaginable to distributing a message within any network. Any origin node wishing to distribute a message does so by sending the message to every other node it is aware of within the network. In turn, every receiving node sends this message to every other node it is aware of, and the process repeats. The primary flaws with this approach is the lack of termination of the message (the protocol is stateless, nodes will forward a query every time they receive it), and high load on the network due to the huge number of redundant messages.

Flooding forms the basis of many real-world P2P networks in operation, such as Gnutella. To prevent queries from propagating infinitely and never terminating Gnutella limits the number of hops a query can take to a maximum of 7. While this solves the termination problem, it becomes impossible to search the entire network, and the probability of finding data becomes relative to its popularity.

3.1.2 *Structured Overlays*

Due to the hash-table nature of a DHT, the only form of search query supported (beyond that of unstructured overlays) without some form of extension is an exact match. All DHTs support, either directly or indirectly, a hash-table interface providing the methods `put(key,value)` and `get(key)`. Given the key corresponding to a service, a DHT can guarantee the return of results, usually within a specific number of hops, if any such results exist. DHTs and the lookup process are discussed in detail in Section 2.1.2.1.

There are many more complex search techniques which are built on top of structured P2P overlays, these are discussed below in Section 3.2.

3.1.3 *Hybrid Overlays*

As discussed in Section 2.1.3, Hybrid overlays are formed of both regular nodes and are what usually referred to as super-nodes (or super-peers). In this design flooding is still the most common technique, however regular nodes will send the query to its known super-nodes, which in turn will flood the query to a much larger number of nodes.

The alternative design sometimes used in Hybrid networks is a central server for coordination. In this design all nodes register the data they have with the central server, and any queries are sent directly to the central server. This makes searching for data a relatively trivial task, however introduces a single point of failure and authority.

3.2 STATE OF THE ART

In many cases the limited support for search provided by structured P2P overlays is not sufficient and clearly does not qualify as support for complex queries. This has resulted in many proposed systems which build support for more complex queries on top of the exact match facility.

3.2.1 *Token pointers in a DHT*

A simple approach to supporting keyword-based search, as used by International Naming System (INS)/Twine [5], is to index pointers to the service at multiple locations within the network, defined by the associated keywords.

The term resolvers is introduced to describe a service for resolving a resource to a location. In INS/Twine resolvers collaborate as peers. The system maps resources to resolvers by transforming descriptions in to numeric keys, which are then mapped on to the DHT.

The authors claim a resource discovery system should have three main goals:

- Handle complex resource descriptions and queries.
- Dynamically handle changes in resource state and location.
- Scale along with the network.

The INS/Twine system extracts which it calls strands, unique subsequences of attributes and values, from a resource description. Each strand is then hashed and stored within the DHT.

Strands are extracted by first converting resource descriptions in to a canonical form: an Attribute-Value Tree (AVTree).

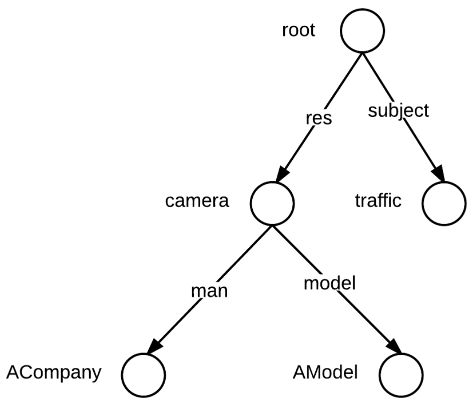


Figure 3.1: An example AVTree as used in INS/Twine.

To handle dynamic changes within the network, all resources are stored with soft-state and are refreshed periodically. Failure to refresh results in the resource description being removed from the network.

The beauty of this approach is its simplicity. It works over any existing DHT without modification to the routing or lookup algorithms, and can make use of existing algorithms for replication and redundancy. By republishing periodically and expiring entries it is possible to trade off maintenance traffic against data freshness depending on the use case.

A query in INS/Twine must only consist of a small number of keywords otherwise it may result in a large query fan-out, as each keyword requires a sub query to a different peer.
Supported query types: Exact-match, Keyword

Similarly, the Open DIStributed Search Engine Architecture (ODISSEA) system [62] maintains a distributed global inverted index, by hashing extracted tokens and storing pointers within the network. The primary difference between ODISSEA and INS/Twine is use of an inverted index rather than simply keywords.

This approach builds on the feature set of INS/Twine by adding full-text support, however this is arguable. The inverted index is distributed across the network, and practical query can only consist of a small set of keywords without resulting in a large query fan-out. Additionally, while the system can determine which documents contain a set of keywords, it cannot guarantee their ordering.
Supported query types: Exact-match, Keyword, Full-text

eSearch [63] is a P2P keyword search system based on a hybrid indexing structure.

Keywords are extracted and published within the DHT, however instead of simply publishing a keyword and pointer, the entire keyword set is published at each location. This adds storage overhead, but allows a multi-term query to be processed locally by a single node.

This approach solves the query fan-out problem of INS/Twine - a query with multiple keywords need only be routed to a single node responsible for any of the keywords. That node is then able to determine if a document exists that matches all keywords.
Supported query types: Exact-match, Keyword

Keyword-Set Search System (KSS) [27] is a system for supporting keyword search in P2P networks using distributed inverted indexes. These indexes are stored within the P2P network by hashing the set of keywords and storing at the appropriate peer. The novel approach in KSS is the use of keyword-sets rather than individual keywords, hence queries can be processed at a single node rather than combining results from many. The trade-off is a much higher insert and storage overhead due to more index entries per document.

Like eSearch solves the query fan-out problem of INS/Twine, KSS solves the query fan-out problem of ODISSEA. However, since KSS and ODISSEA make use of inverted indexes rather than extracting top keywords, the storage cost of storing the entire index at every responsible node is considerable, and grows with the number of unique terms in the document.
Supported query types: Exact-match, Keyword, Full-text

Keytoken-based Index and Search Scheme for Wild-cards (KISS-W) [34, 35, 36] is a system in which wild-card search is supported over a hypercube based network by extracting key-tokens from keywords. A key-token is a pair `{c,i}`, where c is a character in the keyword and i is the characters position. Key-tokens are also extracted counting backwards to allow queries where the wild-card is of unknown length. The keyword ACM would result in a key-token set of `{a,1},{c,2},{m,3},{a,-3},{c,-2},{m,-1}`, and the search term A*M would result in the key-token set `{a,1},{m,-1}`. After extraction the key-tokens are mapped to the key-token space then converted to an r-bit vector by applying a hash function, as shown in Figure 3.2. The document, or a pointer to the document, is then stored at the node with the identifier corresponding to the calculated r-bit vector.

To perform a search the set of nodes which may index a service matching the description need to be identified. It is easy to see that these nodes must all contain the key-token set extracted from the search term, and hence their identifiers must all have 1 at the same positions as the r-bit vector calculated from the search term key-token set. Using a spanning binomial tree [33] of the sub-hypercube matching the search terms r-bit vector, the appropriate nodes can be queried and matching services found.

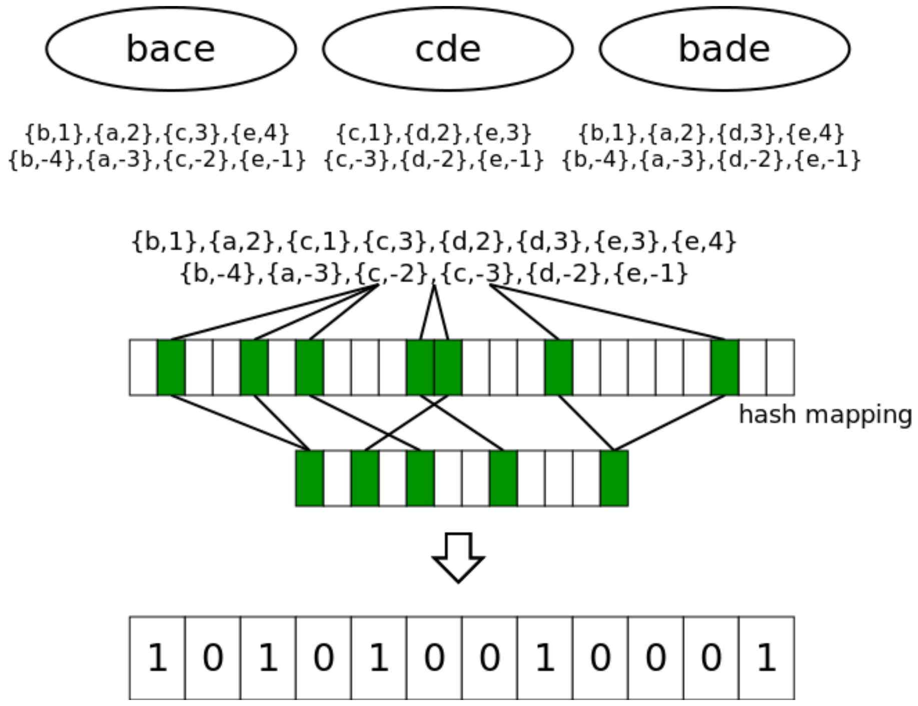

bace
cde
bade
{b,1},{a,2},{c,3},{e,4}
{b,-4},{a,-3},{c,-2},{e,-1}
{c,1},{d,2},{e,3}
{c,-3},{d,-2},{e,-1}
{b,1},{a,2},{d,3},{e,4}
{b,-4},{a,-3},{d,-2},{e,-1}
{b,1},{a,2},{c,1},{c,3},{d,2},{d,3},{e,3},{e,4}
{b,-4},{a,-3},{c,-2},{c,-3},{d,-2},{e,-1}
hash mapping
1 0 1 0 1 0 0 1 0 0 0 1

Figure 3.2: Example mapping of keywords to key tokens with application of a hash function to produce the r-bit identifier used by KISS-W.

KISS-W is a keytoken-based system which supports wild-card search. To solve this problem the simplicity and flexibility of the other keytoken-based techniques has been sacrificed - this system only works over hypercube-based networks
Supported query types: Exact-match, Keyword, Wild-card

3.2.2 d-*dimensional index space mapping using Space Filling Curves (SFCs)*

An alternative approach for keyword search, known as Squid [55, 57, 56], represents services in a d-dimensional coordinate space with their position defined by a set of keywords. For example using base-26 coordinates, a service providing the weather in Scotland may be stored at the coordinates $(scotland, weather)$, as shown in Figure 3.3. This d-dimensional coordinate space is then mapped to a 1-dimensional index space using a Hilbert SFC, allowing the query to be routed by the underlying DHT.

A SFC is a continuous mapping from a d-dimensional space to a 1-dimensional space. Using this mapping it is possible to describe a point in a cube either by its spatial coordinates or by its distance along the curve.

SFCs have 2 important properties that make them suitable:

- A SFC always has equal length of the curve contained in each sub-cube, which means it has $n^{(}k * d)$ segments.
- A SFC is locality preserving. Close points in the 1-dimensional space are guaranteed to be close points in the d-dimensional space. However the reverse is not true.

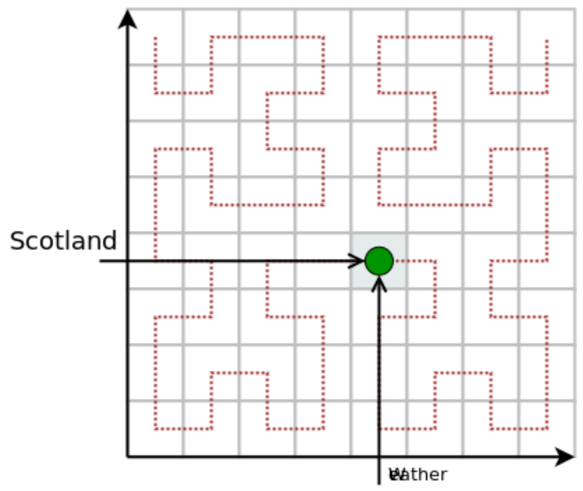


Figure 3.3: A Hilbert SFC on top of a 2-dimensional keyword space, showing location of a document indexed by keywords "Scotland" and "Weather".

Query processing in Squid is a two step process: translating the query to potential clusters of the SFC-based index space, and distributing the query to nodes within those clusters.

If the query consists of complete keywords (no wildcards or ranges) then it will be mapped to a single point in the index space, and hence an individual node. If the query contains partial keywords, wildcards, or ranges, the query will be mapped to a range of the index space, and hence a cluster of nodes.

The use of restricted base coordinate system obviously restricts the flexibility of such a system - while alphabetic keywords may be enough for some domains, others may need alphanumeric support. When languages other than English come in to the equation such a system becomes unusable. Similarly when using such a system a key-space must be defined which can then be mapped on to the 1-dimensional key-space. By placing bounds on the key-space, bounds are placed on the minimum and maximum length of keywords that can be supported.
Supported query types: Exact-match, Keyword, Range, Wildcard

A very similar approach which makes use of a Hilbert SFC on top of the CAN overlay is described by Andrzejak and Xu [2].

3.2.3 *Load balancing with Virtual Peers*

By removing the consistent hashing algorithm and hence preserving data locality, PRoBe [52] provides support for range queries. PRoBe organises peers into a multi-dimensional logical space, similar to that used by CAN [48]. Since the data items and queries are directly mapped onto the logical space, locality is preserved and the range queries can be processed by only visiting the relevant peers. To balance load virtual peers are used, allowing peers to manage multiple zones rather than a single zone. When a peer P finds that the ratio between the load of its most loaded neighbour and itself crosses a threshold, it will hand over its virtual peers to its least loaded neighbour. P then splits the load with the most loaded neighbour such that their loads are nearly equal.

The approach of removing consistent hashing is a simple solution to supporting range queries - with data locality preserved querying a range becomes a trivial exercise. The problem then becomes one of load balancing while maintaining the relative position of data within the network.

PRoBe solves this by decoupling the logical boundaries in the key-space with the physical boundaries between peers. This adds an additional layer of complexity to the system, as well as complexity via an algorithm to handle load calculation and zone handoff between neighbours.
Supported query types: Exact-match, Keyword, Range

A similar approach called P-Ring is proposed by Crainiceanu, Linga, and Machanavajjhala [12]. In P-Ring consistent hashing is removed, and load is based by adding a layer above the P2P network. This layer partitions the ring in to ranges, ranges are then mapped on to the underlying peers.

Karger and Ruhl [38] also propose a methods for load balancing a P2P system when consistent hashing is not in use, also through the use of virtual peers.

Gupta, Agrawal, and Abbadi [29] propose a method for supporting range queries through the use of locality sensitive hashing [43].

3.2.4 *Tree structured networks*

The Skip Graph system [3] changes traditional overlay structure by exploiting the underlying tree structure rather than using consistent hashing, to allow tree functionality rather than hash table functionality.

The Skip Graph system organises peers in to a skip list structure. Searching for a node with a particular key involves searching first in the highest level, and repeatedly dropping down a level when determining the node is not in the current level.

A similar approach, also making use of skip graphs is proposed by Gonzalez-Beltran, and Sage [28].

By changing the network structure to a tree the fundamental design of the network is changed. As discussed by Aspnes and Shah [3], a tree structure has single points of failure, and hence in fact a graph structure must be adopted. This adds further complexity, both in terms of the theory, as well as the maintenance required to balance.

Like removing the use of consistent hashing, using a tree-based structure rather than a hash-based structure introduces the problem of load balance.
Supported query types: Exact-match, Range

The baton system [30] describes a p2p network based on a balanced tree structure. Since the peers form a tree structure range queries are supported.
Supported query types: Exact-match, Range

Distributed Segment Trees (DSTs) [71] aims to solve the problem of range queries and cover queries through the use of segment trees. A segment tree is a data structure for storing ranges, and allows querying which of the stored segments contains a given point. A cover query is defined as a query in which the goal is to find all the ranges currently in the system that cover a given key.

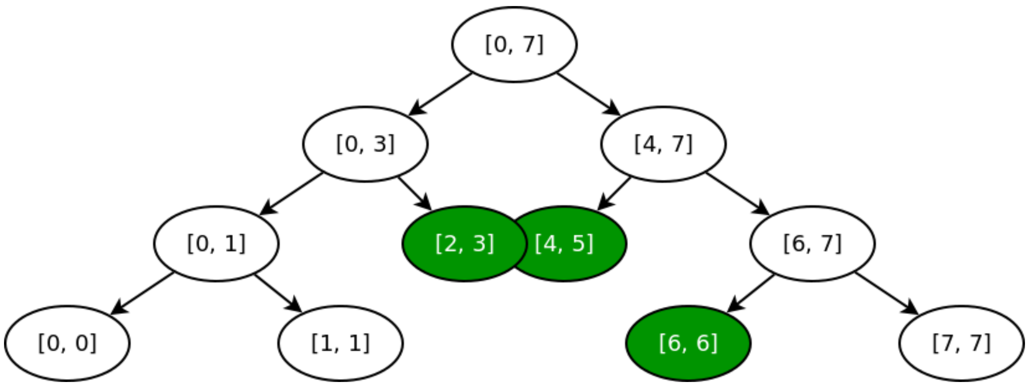


Figure 3.4: A segment tree with a range $[0, 7]$ and the range $[2, 6]$ via three subranges.

To address the issue of load balancing a mechanism termed downward load stripping is introduced. With downward load stripping each peer keeps track of the count of keys it has that could be covered by either it's left or right child. If either of the counters reaches a threshold the key is discarded and any query for such key will be split across both children instead.

Since the segment tree is a full binary tree the usual tree operations, and hence both range and cover queries, are supported.
Supported query types: Exact-match, Range

Wu, Gao, and Yu [68] extend the use of a tree structure, and make use of K-Dimensional Trees (K-D Trees) to support multi-dimensional range queries. Gao and Steenkiste [20] also make use of K-D Trees to support range queries.

A K-D Tree is a space partitioning data structure for distributing keys in a d-dimensional space. This builds on the other tree-based approaches by adding support for high dimensional range queries, such as required for image comparison.
Supported query types: Exact-match, Range

3.2.5 *Indexing by Semantic Vector*

An approach for semantic search is known as pSearch [64], built on top of a hierarchical version of CAN known as eCAN [69]. Given a document or service description, term vectors are computed using a Vector Space Model (VSM). From these term vectors, a semantic vector S is computed using Latent Semantic Indexing (LSI), and the document or service description is then stored in the DHT at the coordinates denoted by S, resulting in similar services being close together. During search the semantic vector Q of the search query is computed and the query is routed to the coordinates denoted by Q. Upon receiving the search query, the node responsible for Q floods the query to nodes within a defined radius based on the similarity threshold, as shown in Figure 3.5. Receiving nodes perform a local search using LSI, and return their results to the querying node.

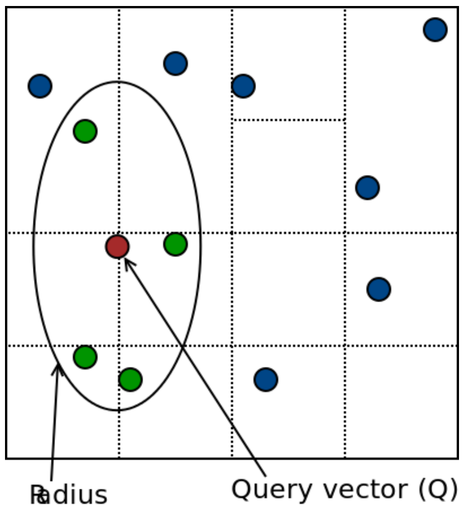


Figure 3.5: An example pSearch system, shwoing positions of services on top of a CAN network. Due to LSI, similar services are closer together, as indicated by the green nodes.

Load balancing is addressed by modifying the bootstrap process - when a node joins it randomly picks a document that it will publish and uses the semantic vector of that document as the point towards which the join request is routed. Assuming that nodes contents tend to be related, this adds capacity at the part of the network in which more capacity is about to be required. This also means that documents will tend to be stored on either the owning node, or it's neighbours.

While this approach efficiently solves the problem of semantic search, it does not address other query types. It also requires a specific key-space and structure in the network, and hence cannot be generalised to all DHTs.
Supported query types: Exact-match, Semantic

3.2.6 *Efficient Broadcasting*

Schlosser et al. propose a new network structure known as HyperCup [54] which allows for efficient broadcast and search. The structure proposed organises peers into a hypercube, or more generally a Cayley graph, topology. Edges in the graph are tagged based on their dimension. Figure 3.6 shows a hypercube topology with base $b = 2$, which in 3 dimensions turns out to be a hypercube with b nodes in each dimension. A complete hypercube graph consists of $N = b^{L+1}$ nodes, where all nodes have $(b-1) \times (L+1)$ neighbours, $(b-1)$ in each dimension - where $L+1$ is the number of dimensions.

A broadcast algorithm is described which is guaranteed to reach all nodes with exactly $N-1$ messages. A node invoking a broadcast sends the broadcast to all its neighbours, tagging it with the edge label on which the message was sent. Nodes receiving the message restrict the forwarding of the message to those links tagged with higher edge labels.

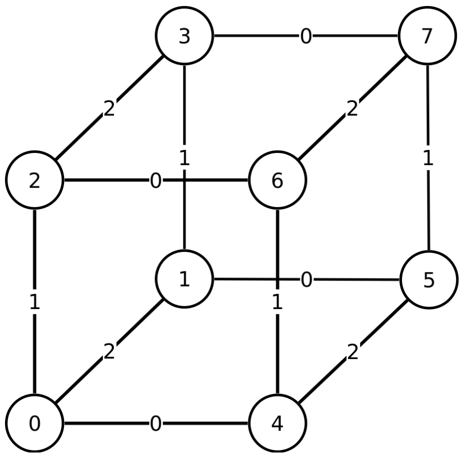


Figure 3.6: Structure of a 2-dimensional HyperCup network.

This approach is limited to a specific DHT structure, in this case HyperCup. It approach is designed primarily for broadcasting, but mentions search as a possible application. It does not however consider the fact that a query need not reach the entire network to be successful, and does not deal with the issue for replies when a matching document is discovered. Using a naive approach of messaging the query originator when a match is found is likely to result in the originator being overloaded when searching for a popular term.

Additionally, the issue of out-of-date routing tables in high-churn environments was not discussed. When broadcasting a broadcast tree is implicitly constructed, and breakages in that tree can result in large sections being missed.
Supported query types: Exact-match, Keyword, Range, Semantic, Wild-card, Full-text, Regex

In the Self-Correcting Broadcast algorithm [24] is an algorithm for broadcasting a search query over an entire DKS network. This allows the query to be processed locally, and hence provides the same flexibility and broad range of supported query types as flooding in an unstructured network.

A node issuing a search query iterates through all levels of the routing table, starting at the first level. At each level, the node moves in counter-clockwise direction through all of its intervals, broadcasting a message to each responsible node r. Each broadcast message carries with it the parameters l (level), i (interval) and the `limit`. The message first delivers the intended data to the receiving node. Secondly it serves as a request to cover all nodes in the interval $(r \oplus i * \frac{N}{k^l}, \mathrm{limit})$. Due to outdated routing tables some intervals might not seem to have any nodes even though they are populated. The responsibility of covering those intervals is delegated to the next interval. If node n gives another node the responsibility to cover other preceding intervals and other nodes exist in those intervals, the node will trigger correction-on-use, and the routing information will be corrected at node n.

Self-Correcting broadcast builds on the approach of Efficient Broadcasting [14], but is customised for use over DKS networks. It does not generalise to all DHTs.

While the correction-on-use technique allows discovery of extra nodes that are not in the routing table, though should be, it does not solve the issue of delegating sections of the tree to nodes which are no longer alive, and hence will not perform well in high churn environments.
Supported query types: Exact-match, Keyword, Range, Semantic, Wild-card, Full-text, Regex

A framework called Recursive Partitioning Search (RPS) for Blind-Search over structured peer-to-peer overlays is presented in [67], with a realisation for Chord. RPS is a version of the Efficient Broadcasting algorithm [14], however it includes a TTL value as well as a limit, and a node will only broadcast the query to its sub-tree if it cannot satisfy the query itself first. With enough data replication the whole network does not need to be searched to find the data, and once a single copy has been found there is no need to continue looking.

[66] consider ways to restrict the number of nodes visited but without reducing the query success rate. In a network of size N, if the TTL value is set to $\log_2(N)$ then in a stable network the algorithm results in a 100% success rate, so it is a maximum reasonable value for the TTL. In most cases a smaller TTL value can still achieve high success rates because most resources are replicated among multiple nodes.

A variation of the algorithm to control the query message traffic is as follows: The query originator selects a partial list of its fingers and sends the query to them first. The tag in the message now contains two values, the nodes local search limit and the global search limit. Receiving nodes update the local limit as before, but do not touch the global limit. This variation allows a node to divide the search space and perform RPS sector by sector sequentially, if required.

In [66] realisations of RPS for both Chord and Pastry are presented. To reduce network load the use of go-stop signals (RPS+G) and local indices (RPS+L) are suggested.

Typically Blind-Search algorithms generate multiple query messages that traverse the network concurrently, thus even when one of the query messages finds a result the others continue to search. To reduce the number of these messages some intermediate nodes enquire if the search originator has already found the target resource. The initiator either sends a GO signal if the search is still pending, or a STOP signal to terminate the query message. By using go-stop signals the scalability of Blind-Search algorithms can be enhanced.

RPS is a framework and selection of techniques for optimising Efficient Broadcast [14] in the context of search. These techniques are not DHT specific and build on the observation that a query need not reach every node in a network to be successful.
Supported query types: Exact-match, Keyword, Range, Semantic, Wild-card, Full-text, Regex

3.3 SUMMARY

In summary, search queries can be categorised in to a large number of types, ranging from keyword to regular expression. In this Chapter a review of the state of the art of Complex Queries was presented, which showed that while there are many systems which can handle certain types of search queries, only those based on efficient broadcasting can handle all types of search queries.

TOKEN POINTERS

Token pointers is a technique that involved extracting keywords from a document, and inserting pointers to the document in to the DHT at locations defined by the keywords. It supports indexing of a limited number of keywords per document, with larger numbers of keywords increasing the cost of updates and maintenance.

Supported: Exact-match, Keyword
Systems: INS/Twine [5], ODISSEA [62], eSearch [63], KSS [27], KISS-W [34, 35, 36]

SPACE FILLING CURVES

Systems that make use of space filling curves are able to reduce a multi-dimensional index space in to a single dimension. This is used to map from a collection of keywords to a single point in the DHT index space, and hence locate the node responsible for a document matching those keywords. Over token pointers, this approach retains the locality of documents, allowing for range queries. The downsides of this approach are even more restrictions on the number and length of keywords, as well as unbalanced load due to retention of data locality.

Supported: Exact-match, Keyword, Range, Wild-card
Systems: Squid [55, 57, 56], Andrzejak and Xu [2]

virtual peers

The use of virtual peers allows systems to avoid the use of consistent hashing, preserve data locality, and hence support range queries on keywords. Virtual peers are introduced to balance load. Like above, documents are indexed by keywords and have a restriction on the number of keywords per document.

Supported: Exact-match, Keyword, Range
Systems: PRoBe [52], P-Ring [12], Karger and Ruhl [38], Gupta et al. [29]

tree structured

Tree structured systems are yet another approach that removes the use of consistent hashing within DHTs, and preserves data locality. Like Space filling curves, systems based on tree structures can support range queries, but inherit problems with load balance and limitations of number of keywords.

Supported: Exact-match, Range
Systems: Skip graphs [3], González-Beltrán et al. [28], baton [30], DSTs [71], Wu et al. [68], Gao and Steenkiste [20]

semantic vectors

The use of semantic vectors allows storing a document within the DHT at a location determined by its semantic vector value rather than consistent hashing. Again this preserves data locality, meaning semantically similar documents are stored together, and it is possible to retrieve similar documents using range queries. This approach introduces the issue of load balance, and is only applicable to semantic search.

Supported: Exact-match, Semantic
Systems: pSearch [64]

EFFICIENT BROADCAST

Systems making use of efficient broadcast don't place any requirements on location of data during storage, instead moving the complexity to finding the data during query time. They exploit the network structure to build a broadcast tree at query time, distributing the search query to all nodes within the network. The search query is then processed locally by every node, meaning it is possible to support much more complex types of query. Efficient broadcast based systems have a much higher query cost to others, as each query has to reach $O(N)$ nodes.

Supported: Exact-match, Keyword, Range, Semantic, Wild-card, Full-text, Regex
Systems: HyperCup [54], Self-Correcting broadcast [24], RPS [67, 66]

Of the existing systems which can support all types of search queries, HyperCup and Self-Correcting broadcast are very prescriptive, and require a very specific network structure to support the search algorithm used. This ties together the search query processing layer, the data storage layer, and the request routing layer, making it difficult to modify any of these independantly. This rules them out as options, as one of the primary objectives of this thesis is a generic system which can be applied on top of an existing DHT.

The only system identified which meets the objectives of this thesis is RPS. RPS also provides multiple optimisations in the context of search, however does not consider the impact of high churn environments, low bandwidth environments, or how to efficiently respond to successful search queries.

The next chapter builds on this observation, introducing a novel search system based on efficient broadcasting, capable of supporting all types of complex queries.

4 A BLIND-SEARCH SYSTEM

As shown in Chapter 3 there are a plethora of systems adding support for certain types of complex queries to DHTs, but only one small family of systems which can support any type of query. This is the family of systems based on efficient broadcasting, which is termed blind-search. In this chapter the concept of blind-search is introduced, and a novel blind-search system capable of supporting complex queries is presented.

4.1 INTRODUCTION

The term Blind-Search is used to describe a search operation in which no information about the search space is known, other than to distinguish the goal state from all others. In other words, as a query traverses through the network it has either reached the goal or it has not. There is no concept of distance to the goal as with regular operations in a DHT. As such a Blind-Search query can be thought of as the structured equivalent to flooding (see Section 3.1.1).

4.1.1 *Design Criteria*

When designing A Blind-Search System I evaluated the system against the following criteria:

COVERAGE OF ALL NODES

The search query should be delivered to every node currently within the network (if required). This provides a deterministic search, guaranteeing that if a match exists, it will be discovered.

MINIMAL REDUNDANT MESSAGES

Redundant messages waste network resources, and should be avoided when possible. This reduces load on the network, and removes the requirement for nodes to store a history of processed queries for filtering duplicates. It is also important to note that redundant messages in the context of broadcasting are different from redundant messages in the context of searching - once a search has found a result it may, or may not, be desired that the query continues to all other nodes.

BALANCED LOAD

During a search the load generated should be reasonably balanced throughout the network. No single node should be required to contribute an unfair share of resources.

SUPPORT FOR HETEROGENEOUS NETWORKS

Taking the load balance requirement into account, a node should be allowed to contribute more or less resources than average depending on their free capacity.

EFFICIENT RESPONSE DELIVERY

Query responses should be delivered in an efficient manner which does not risk overloading the query originator when large numbers of matches are found.

GENERIC SOLUTION

A solution should be generalisable and not rely on a specific overlay structure. By decoupling the search algorithm from the network structure, implementers are free to choose whichever overlay best suits their needs.

SURVIVAL UNDER CHURN

High levels of churn should not impact the success of a search query any more than it impacts the success of routing in the underlying DHT.

4.1.2 *Related Work*

Looking back at the efficient broadcast based solutions in Section 3.2.6, it can be seen that these solutions all meet some of the requirements, but none meet all of them.

HYPERCUP

While the HyperCup system provides support for blind-search, it does not meet the majority of the requirements set out above. It is tied to a specific DHT structure, without consideration for high churn environments or heterogeneous networks. It also does not consider efficient response delivery, or the ability to halt a query once a result is obtained.

SELF-CORRECTING BROADCAST

Self-Correcting Broadcast attempts to improve network structure by detecting missing nodes within the broadcast tree. This will improve the general correctness of the routing tables and help in the long term. However, it does not consider heterogeneous networks, efficient response delivery, or the ability to halt a query once a result is obtained.

RECURSIVE PARTITIONING SEARCH

RPS is much closer to what can be considered a full solution, with a broadcast algorithm that can generalise to different DHTs and optimisations specific to the use case of search. This includes go-stop signals to halt the search upon successful completion, and the introduction of a TTL to restrict the scope of the search, building on the assumption that most data is going to be replicated at multiple points within the network and hence searching the entire network is redundant. It is, however, still missing any analysis in high churn environments, and provides no solution for efficient response delivery or heterogeneous networks.

The work in this Chapter builds on top of the Efficient Broadcasting algorithm [14] and a combination of the above mentioned Self-Correcting broadcast and Recursive Partitioning Search (RPS) systems. This Chapter fills in the gaps in the query response routing, further optimise the distribution in the context of search, and optimise the load balance of the algorithm for real world networks (i.e. heterogeneous with potentially high levels of churn). In Chapter 5 these optimisations are extended from the search algorithm to the DHTs itself, with a set of optimisations that can be applied regardless of the DHT structure used, which improves both the structure and correctness of the DHT, as well as the ability to efficiently perform blind-search.

4.2 QUERY DISTRIBUTION

Query distribution in my proposed system works by dynamically building a broadcast tree at runtime using the structure of the DHT. The algorithm is broadly based on the Efficient Broadcasting algorithm [14], though since the structure varies depending on the DHT, the algorithm used must also vary depending on the DHT.

In this Section 4.2 existing algorithms are discussed that will be built upon, for efficiently broadcasting a query over different DHT types.

4.2.1 *Ring based DHTs*

Ring-based overlays, such as Chord [59], can generally be covered using the Efficient Broadcasting algorithm [14] by El-ansary et al..

Efficient Broadcasting [14] is an algorithm for broadcasting complex queries, or indeed any data, over DHTs using Chord [59] as an example. To initiate a query, a node n will send the query along with a limit to every node in its finger table. The limit parameter given is the identifier of the next finger in the finger table, and is used to restrict the forwarding space of the receiving node to $(n, \mathtt{limit})$. The last node in the finger table is given a limit of the originating node. When the message is received by a node it forwards the broadcast to any fingers it has within the forwarding space, giving each one a new limit.

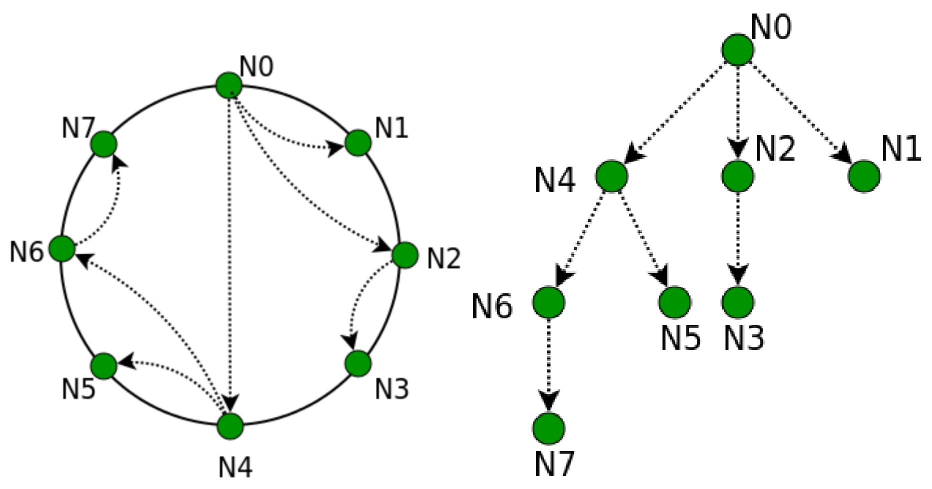


Figure 4.1: Efficient Broadcasting with a ring-based DHT. Left: The broadcast operation from node N0 depicted on a Chord ring. Right: The same operation, depicted as a broadcast tree.

4.2.2 *Prefix routing DHTs*

Castro et al. [9] describe an algorithm for broadcasting over Pastry, and other prefix routing based networks.

While not specifically designed with search in mind, it can be used to broadcast complex queries. A node broadcasts a message by sending the message to all nodes in its routing table; each message is tagged with the routing table row r. When a node receives a message it forwards the message to all nodes in its routing table with rows greater than r. This continues until a node receives a message tagged with r and has no entries in rows greater than r.

4.2.3 *Cartesian space based DHTs*

Multi-dimensional cartesian space based overlays, such as CAN [48], make use of a broadcast algorithm described by Ratnasamy et al. [49] for performing application-level multicast over CAN.

In a CAN with dimension d, each node has at least $2d$ neighbours; one to move forward in dimension d and one to move backwards. To initiate a broadcast the source node forwards the message to all its neighbours. When a node receives a broadcast message from a node with which it neighbours along dimension i, it will forward the message to those neighbours along dimension $1...(i-1)$ and the neighbours in dimension i on its other side. To prevent the message from looping back around a node it does not forward a message along a particular dimension if that message has already traversed at least half-way across the space from the source coordinates along that dimension. It is worth noting that for a perfectly partitioned coordinate space the algorithm ensures each node receives the message exactly once. For an imperfectly partitioned space, a node may receive the same message from multiple neighbours.

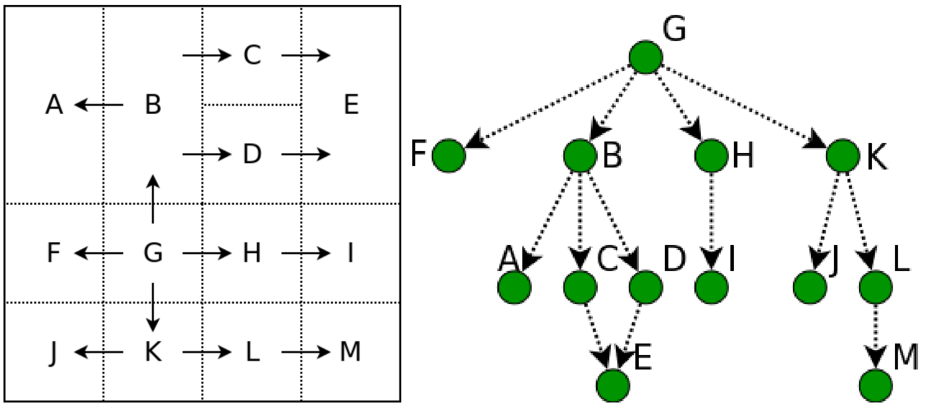


Figure 4.2: Broadcasting in a Cartesian space. Left: The broadcast operation from node G depicted on a CAN. Right: The same operation, depicted as a broadcast tree.

4.2.4 *Hypercube based DHTs*

Hypercube-based overlays, such as HyperCup [54], can be covered using the broadcast algorithm described by Schlosser et al. [54] for broadcast search over HyperCup.

This algorithm has already been covered in Section 3.2.6.

Using the appropriate above mentioned algorithm the proposed blind-search system can be applied to the majority of existing DHTs. This gives implementers the ability to choose the DHT which best suits the needs of their application.

While all of these broadcasting algorithms can be used for search query distribution, they do not consider the case of responses. In particular, this Thesis focuses on the case where many results are found, and hence a naive direct response will overwhelm the originator.

4.3 RESPONSE ROUTING

The most simple, and most naive, approach to response routing is for the search query to contain either the identifier or IP:port of the query originator, and have the locating node either route the response or contact the originator directly. Whether this approach is appropriate depends on how many responses are expected to be received. This depends on the popularity of the search term, and the scope of the search.

The scope of the search can be greatly reduced by the use of a TTL or Go+Stop signals [66], however this might not always be appropriate, in some cases a query might be specifically searching for all occurrences. In these cases a smarter form of response routing is required.

In this section (4.3) two novel approaches for efficient response routing are proposed.

4.3.1 *Routing with Collation*

The first approach for efficient response routing, called Routing with Collation, that I propose, trades search speed for simplicity. In Routing with Collation a search query will include with it the originating nodes identifier, and a unique search identifier. Any nodes with search matches will recursively (independently of the routing algorithm specified by the DHT) route the result to the originator, including both the originators identifier and the search identifier in the response. Each node along the route will delay the forwarding by a set length of time, and collate any responses they receive during this window.

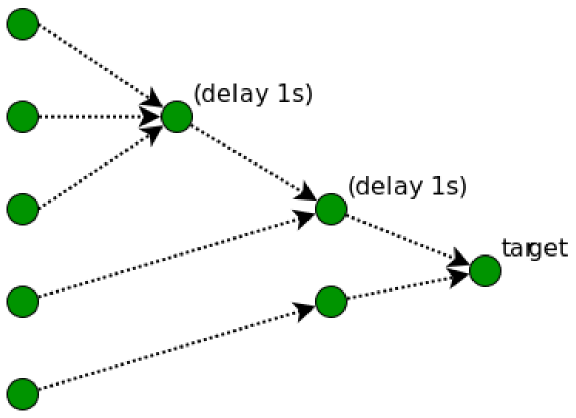


Figure 4.3: Response routing with collation and a delay of 1 second.

This approach makes use of the fact that as a message is routed through a DHT, it will converge as it reaches nearer the destination, allowing nodes around the destination to handle part of the load. As the number of responses increases, the distance from the destination that the messages will start to converge increases, and hence the number of nodes responsible for collating the response increases.

The effectiveness of this approach will vary depending on the characteristics of the underlying DHT. For example in a one-hop network such as DH1T [46] all responses would be routed directly to the originator without passing through any other nodes, and hence have no chance of being collated.

While this approach is simple, it does add a large time penalty, which may or may not be acceptable depending on the use case. Choosing an appropriate time window is also challenging, as nodes near the top of the search broadcast tree will start sending responses much before nodes at the bottom have even received the request.

4.3.2 *Reverse Tree Navigation*

The second approach I propose for efficient response routing is to make use of the same broadcast tree that the request used, and propagate responses back up the tree.

In this approach, again a search query will include a unique search identifier, and will already know its parent in the tree as it is the node from which the message was received. Any nodes with search matches will send the result to their parent, which in turn will forward it to their parent, until it reaches the search origin.

There can be two variations of this approach, a node may either send its response to its parent as soon as it's available, or it may wait for responses from all of its children first.

Sending the response when it becomes available is simple, but will not reduce the amount of traffic reaching the originating node - only guarantee it arrives in a predictable way. However this approach can then take advantage of the technique used by Routing with Collation, with every node waiting and collating responses before forwarding to their parent.

This approach also will add a large time penalty to search responses, though solves the problem of differing underlying DHTs - as long as the broadcast tree has been constructed with a reasonable branching factor (see Section 4.4) there will always be a reasonable number of hops through which the responses can be collated.

4.4 LOAD BALANCE

In terms of load balance, the most important part of the algorithm, which is not covered in any previous work at the time of writing, is the branching factor.

The branching factor is the number of children nodes which a node includes when constructing the broadcast tree, this can range anywhere from 0 to N, depending on the routing table size, the nodes position in the tree, and the structure of the underlying DHT. In networks such as CAN [48], the broadcast mechanism does not lend itself to changing the branching factor; it would fundamentally change the behaviour of the algorithm and prevent all nodes being covered.

However in networks making use of the Efficient Broadcasting [14], the branching factor can easily be varied, with lower branching factors delegating a larger portion of work to other nodes, and larger branching factors resulting in a quicker search.

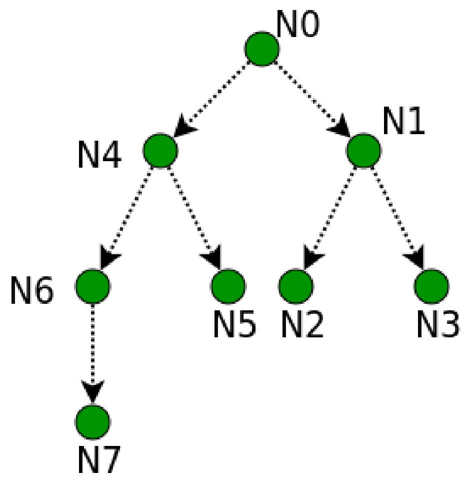


Figure 4.4: Extreme example of branching factor 2, gives a tree depth of 3.

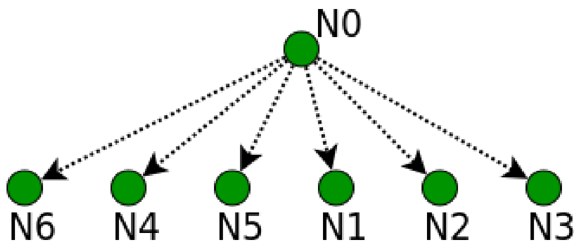


Figure 4.5: Extreme example of branching factor N, gives a tree depth of 1.

The ability for nodes to choose their own branching factor allows for a more much balanced use of resources within a heterogeneous network. Networks with larger routing tables, such as Epichord [40], allow for a much higher branching factor if desired, and are better positioned to take full advantage of this.

4.5 PERFORMING UNDER CHURN

Tree-based structures are inherently poor at handling breakages. Any missing node in the tree results in its children, and their children, and their children, ..., being split from the main tree. In the context of blind-search, this means a single failure can cause a large section of the network to not receive the search query. For example consider the case, shown in Figure 4.6, where a node in the first level of the tree fails, causing half the network to not receive the search query.

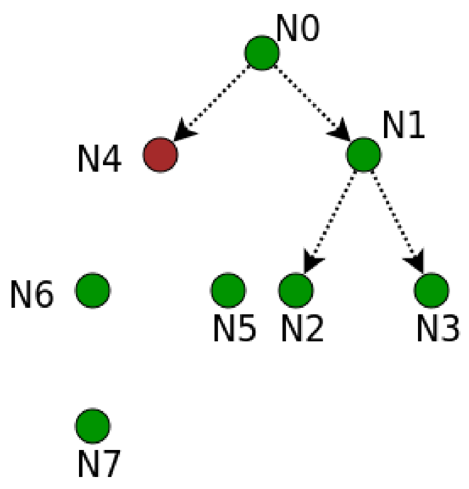


Figure 4.6: A broadcast tree with failure at N4 prevents half the network from receiving the search query.

It would be possible to attempt to detect this situation using a failure detector, such as described by [23], however this adds significant complexity to the algorithm. In Chapter 5 optimisations of the DHT are considered, to reduce both the occurrences and impact of such failures. Experimental results can be found in Chapter 6.

4.6 SUMMARY

In summary, in this Chapter a novel system, "A Blind-Search System", for efficiently performing blind-search over varying types of DHT has been presented. This system uses a combination of different efficient broadcast algorithms, which all build on the idea of using the existing structure in the DHT to generate a broadcast tree on the fly. The specifics for how this is done depends on the type of DHT, but each implementation follows the same theory - that is, partitioning the network based on your routing table and delegating coverage of sections to child nodes. In my system, "A Blind-Search System", each algorithm implements the same interface - allowing the system to select the most appropriate algorithm for a given DHT type.

Looking back at the design criteria for such a system, in a steady state my system guarantees delivery of the search query to every node in the network, with minimal redundant messages, by virtue of the efficient broadcasting algorithms used.

Load balance varies depending on the underlying DHT routing table structure, but the system is capable of varying branching factor during broadcast, allowing nodes to decide for themselves what a suitable load is. By choosing a branching factor of 2 (the minimum possible) the broadcast tree has a depth of $O(\log_2(N))$, and a single node is responsible for at most 2 messages per query.

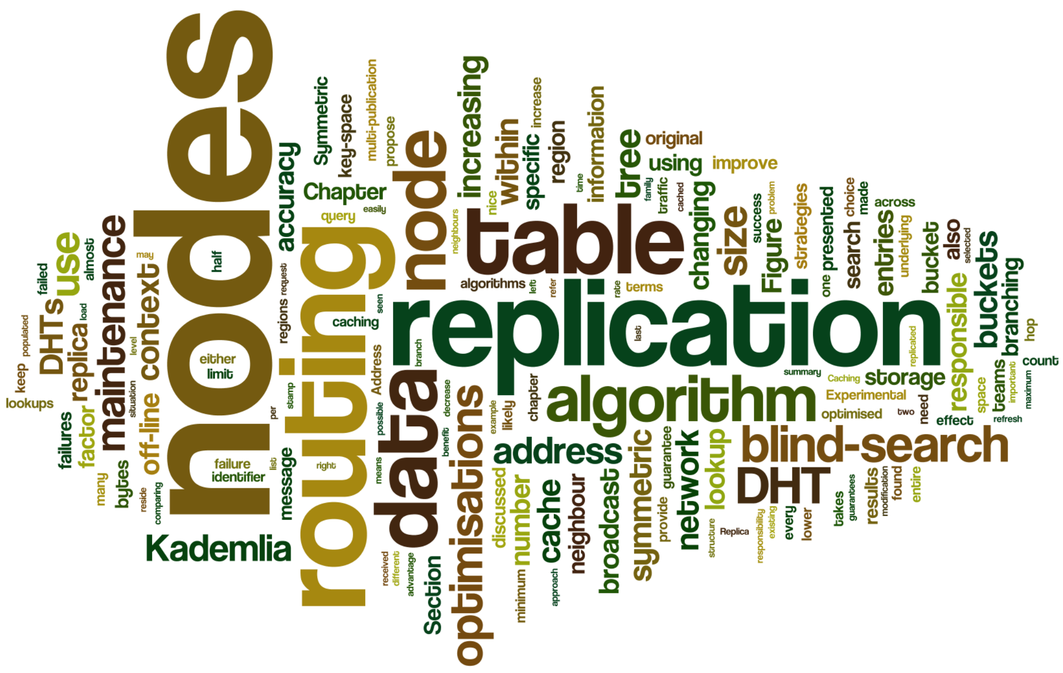
This allows for performance to be optimised in heterogenous networks, where a mobile node may choose a branching factor of 2 though a server may choose not to limit the branching factor.

Efficient response routing is performed using a technique proposed called Reverse Tree Navigation. With Reverse Tree Navigation the response to a search query is always sent to the node from which the query was received. This allows responses to be collated as they traverse back up the broadcast tree, ensuring the load generated from search responses is proportional to the load generated from search requests (in other words, the branching factor chosen by a node is also respected by the nodes sending responses to it).

In all efficient broadcasting based systems the main weak point is susceptibility of the system to churn. In the existing efficient broadcast based systems HyperCup and RPS this is not addressed. In Self-Correcting broadcast the issue of churn is tackled through the use of a complex failure detector which attempts to detect and re-route failed messages. In Chapter 5 I address the issue of churn through optimisations to the underlying DHT and choice of data replication algorithm.

In Chapter 6, experimental results covering the system in this chapter are presented.

5 OPTIMISING DHTS FOR BLIND-SEARCH



In the previous chapter A Blind-Search System was presented, with efficient response routing, query scoping, and load balancing. While this algorithm itself is optimised in the context of search, further optimisation can be performed at a lower level. In this chapter I present optimisations to the underlying DHT, which can be made to improve the performance, both in terms of success rate and query latency, of blind-search.

It is important to note that the optimisations in this chapter are designed to be applicable to any DHT, not just a specific family.

5.1 INCREASING ROUTING TABLE SIZE

In many DHTs a nodes routing table size is limited, often in the order of $\log(N)$. While this allows for a guarantee of the average maximum hop count for a request, it also enforces that guarantee on the average minimum hop count.

In the context of blind-search, a limit to the size of a nodes routing table restricts the maximum branching factor the node can choose. This prevents nodes with high capacity increasing their branching factor, preventing speeding up the search or alleviating load being delegated to other nodes which may not have as much free capacity.

Looking at Kademlia [45] as a concrete example (see Section 2.2.1.2); the most distant bucket in a nodes routing table is accountable for half of the entire address space, as can be seen from Figure 5.1. Although there may not be a node for every possible node identifier, it can be reasonably assumed that nodes will be distributed evenly across the address space due to their use of consistent hashing. Around half of the nodes will therefore be from the opposite side of the address space. This means that Kademlia is artificially limiting the amount of information about the other half of the network. Lookups of these nodes will happen via the k nodes in this bucket, potentially putting an excessive demand on these nodes, and increasing hop count unnecessarily.

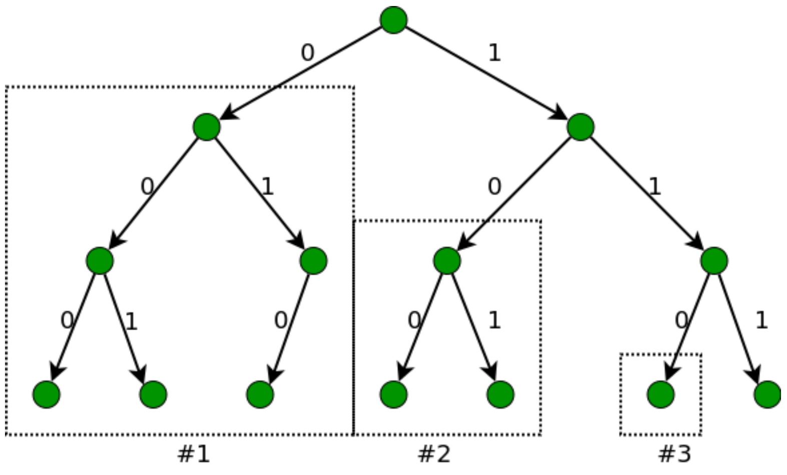


Figure 5.1: A Kademlia routing table with 3 buckets.

There are two ways in which the Kademlia routing table can be modified - changing the number of buckets, or changing the number of nodes that reside within the buckets. By changing the number/layout of buckets the maintenance traffic required is also affected, as it is directly linked to the amount of buckets in use. Instead, in this Chapter I propose changing the number of nodes that reside within the buckets, using a generic technique referred to as Address Caching.

5.1.1 *Address caching*

In various one-hop networks, such as Epichord [40], nodes keep a cache of all nodes they have heard of. Epichord maintains the minimum guarantees that Chord has, but also keeps the cache of other nodes. This optimisation proposes that the limit of k nodes per bucket in Kademlia can be removed, and indeed any other network that imposes similar constraints, by storing information about all nodes learned of in a cache. For Kademlia we refer to this as NKademlia.

To prevent an increase in maintenance traffic, buckets only need to keep up-to-date information for k nodes, where k is the original node limit defined by the DHT. The remaining nodes in the bucket are cached and are available to refill the bucket upon failure of one of the k nodes.

Storing a User Datagram Protocol (UDP) address and time stamp takes around 12 bytes of storage: IP address (4 bytes), port number (2 bytes), time stamp (8 bytes). Even storing a cache of addresses for 1 million nodes takes as little as $12Mb$ of memory. Since local storage is cheap this is not seen to be a problem.

Analysing the routing table as a whole, including the cached addresses, the accuracy is likely to be lower than that of the original DHT without address caching. If it is decided to only use the cache to replace entries leaving in the original routing table, then this is not an issue, however if making use of these extra nodes during lookups, or blind-search, it becomes more important to keep these cache entries maintained.

Experimental results for Address Caching can be found in Chapter 6.

5.2 INCREASING ROUTING TABLE ACCURACY

By using address caching the size of the routing table has been increased, but the accuracy of these extra nodes is lower than that of the primary routing table. This sub-section discusses optimisations aimed at increasing routing table accuracy.

Improvements to the maintenance algorithm can help improve both the routing table size and accuracy. This has benefits for the regular operation of the DHT as well as for blind-search, with reduced occurrences of tree failures, and increased choice in branching factor.

5.2.1 *Downlists*

In many DHTs, nodes are only able to detect offline nodes if they are used when performing a lookup. To improve this situation I propose nodes who discover offline entries while performing a lookup should share this information with appropriate other nodes.

Each node maintains a list of all nodes which it has discovered are offline during its last lookup. At the end of the lookup the entries in this list are sent to all other nodes which were responsible for providing those entries during the lookup. These nodes then also remove the received off-line entries from their own routing table.

A protocol for removing nodes based on data received from other nodes could easily be misused by a malicious node. This problem can be minimised by verifying the offline nodes are actually offline using a ping message.

This approach can be further extended by nodes forwarding information about failed nodes onto the k closest neighbours of the failed node - those nodes are more likely to know of the failed node and hence should be updated.

It should be noted that this modification will not have any effect when using recursive lookups, as the nodes are responsible for forwarding the message themselves, and hence will detect and handle any failures locally.

5.2.2 *Levelling*

I also propose a routing table maintenance modification called Levelling, based on NICE[32]. In Levelling maintenance the routing table is continuously refreshed and checked for connectivity. The refresh task is triggered periodically, and selects a region then sends a ping message to the most stale node. Regions can be selected either sequentially or at random. This continuous refresh guarantees each region has at least one contactable node, and distributes maintenance traffic in a smooth way.

In the original NICE algorithm, which Levelling is based upon, a region is selected with no consideration for the size of the region. In certain DHTs, like Kademlia, many regions are sparsely populated while a small number are highly populated. Uniform maintenance of all regions is not necessarily the most useful. With Levelling, I instead select regions by randomly selecting an identifier anywhere in the key-space, then selecting the region responsible for the chosen identifier. This increases the chances of large regions being chosen, with the chance proportional to the size of the region.

This algorithm can either be applied to the entire DHT routing table, or just the address cache, depending on the underlying DHT in use. Experimental results can be found in Chapter 6.

5.3 OPTIMISING DATA REPLICATION

In the context of blind-search, optimisations to data storage itself provide no real benefit. However, optimisations to data replication can have a huge effect on both speed and success of blind-search.

As discussed in Chapter 2 there are three main families of data replication algorithms: neighbour, path, and multi-publication. Some DHTs prescribe the use of a specific data replication algorithm, and some do not. In almost all cases there is no need to couple a specific data replication algorithm with a DHT, and changing the algorithm in use can provide an easy boost in performance.

5.3.1 *Symmetric Replication*

Symmetric replication [25] is part of the multi-publication family of data replication algorithms, and was described in detail in Section 2.1.2.2.

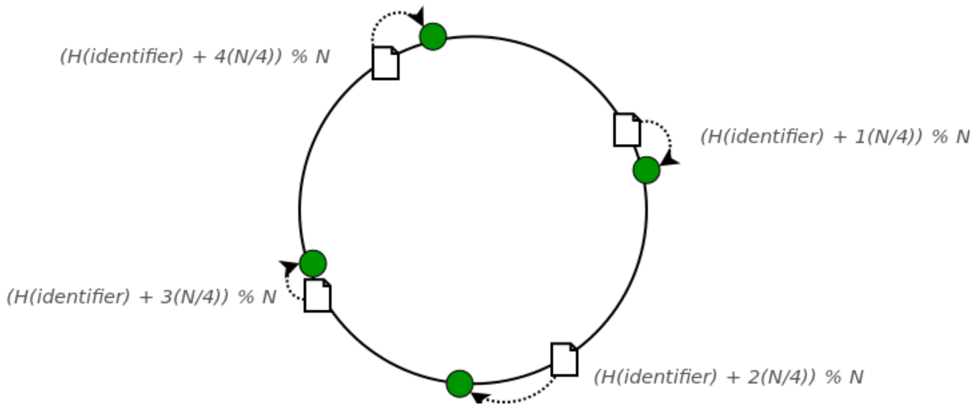

(H(identifier) + 4(N/4)) % N
(H(identifier) + 1(N/4)) % N
(H(identifier) + 3(N/4)) % N
(H(identifier) + 2(N/4)) % N

Figure 5.2: Placement of replica in the symmetric replication strategy.

In the context of blind-search, symmetric replication has the advantage of evenly distributing data across the entire key-space. When a broadcast tree is constructed it is likely that nodes close together in the key-space belong to the same branch in the broadcast tree. This is because tree construction tends to assign sequential ranges of the key-space to specific nodes, as shown in Figure 5.3.

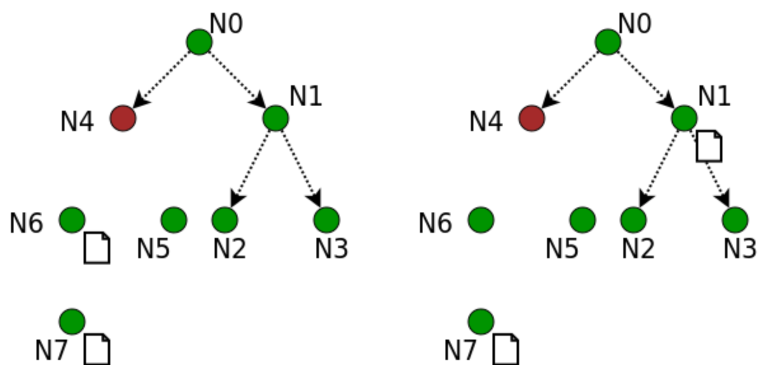


Figure 5.3: Distribution of replicated data within a typical broadcast tree, comparing neighbour (left) and symmetric (right) replication.

In this situation, replicating data among neighbours provides almost no redundancy. By changing to symmetric replication is is guaranteed that the failure of a branch in the broadcast tree can never lose every replica in the network.

Symmetric replication is an existing concept, introduced by Ghodsi, Alima, and Haridi [25]. While it is compared to other replication strategies in terms of lookups, this Chapter presents a novel use as a replacement for other replication strategies in existing DHTs as a means to improve the success rate of blind-search.

Experimental results comparing symmetric replication to other replication strategies, both in terms of regular lookup and blind-search, can be found in Chapter 6.

5.3.2 *Replica Teams*

While symmetric replication has many advantages over other replication strategies, neighbour replication has the advantage of automatic failover - in most DHTs when a node leaves the network the responsibility of its data falls to an adjacent node. With neighbour replication this adjacent node will already have a replica of the data, and can take over responsibility immediately. With any multi-publication replication strategy this is not the case, and the query would need to be serviced by another request to a different publication of the data.

To solve this I propose Replica Teams, which takes an approach in between Neighbour and Symmetric replication. In Replica Teams, teams of replica are split symmetrically across the network, as shown in Figure 5.4. Symmetric replication can be thought of as replica teams with the number of nodes per team set to one.

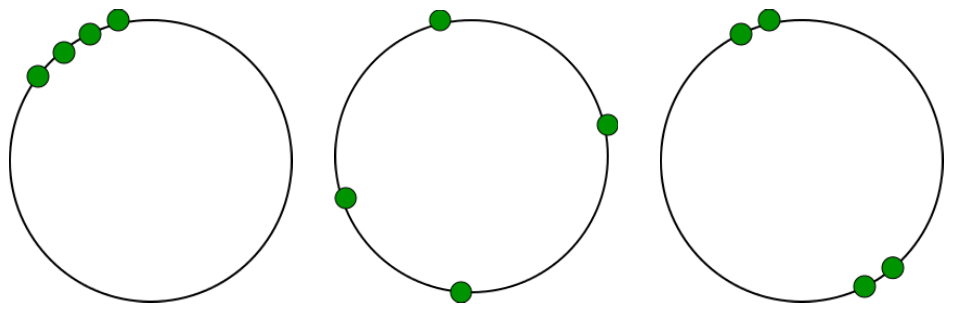


Figure 5.4: Comparing the distribution of replicated data within a network using neighbour replication (left), symmetric replication (center), and replica teams (right).

5.4 SUMMARY

In summary, this chapter has presented a collection of DHT level optimisations, all within the context of blind-search, which have been generalised and can be applied to almost any network, independent of the underlying DHT structure. While each of the optimisations is presented in the context of blind-search, they all also have benefits outside of this context and make sense to be implemented regardless of the use of A Blind-Search System.

An optimisation called Address Caching was presented, in which nodes maintain a cache of other nodes they discover through regular operation. This cache can be used both to replace dead nodes in the primary routing table and hence reduce maintenance costs, as well as to increase the possible branching factor during blind-search by supplementing the primary routing table. As discussed in Chapter 4, the choice of branching factor is important for load balance, and optimising performance in heterogenous networks.

To increase the routing table accuracy and hence reduce the occurrence of failure during broadcast two optimisations, called Downlists and Levelling, were presented. With Downlists, nodes include lists of down addresses in regular communication, allowing faster propagation of failures across the network, and hence faster removal of failed nodes from routing tables.

Levelling introduces a light weight maintenance algorithm aimed primarily at use within the above presented address cache, in which periodically a random key in the network is chosen and the closest node in the address cache is probed to ensure it is still alive. This helps to ensure that the address cache does not become stale.

The most important optimisation presented, in the context of blind-search, is the use of Symmetric Replication instead of Neighbour Replication, for replication of data within the DHT.

Due to how broadcast trees are constructed by the efficient broadcasting algorithms used, nodes close together in the broadcast tree tend to be close together in the DHT. This means, with Neighbour Replication, replica tend to be grouped within the same branch of the broadcast tree, and hence no redundancy against failure of that branch exists. Symmetric Replication on the other hand guarantees an even distribution of replica in the DHT, and hence guarantees that replica will never be grouped within the same branch of the broadcast tree. By making use of Symmetric Replication instead of Neighbour Replication the impact of failure on blind-search can be greatly reduced.

Finally, a modification to Symmetric Replication is presented, which I call Replica Teams. Replica Teams is a middle ground between Neighbour and Symmetric replication, where teams of replica are evenly distributed across the DHT. This is done because in the majority of DHTs when a node fails, the responsibility for its data falls to its neighbour. By creating teams of replica, the neighbouring node will already have the required data and be able to instantly take over responsibility - yet the benefits of Symmetric Replication still exist. Maintenance costs are also reduced, as in the majority of DHTs neighbouring nodes are already monitored closely anyway.

With the mentioned optimisations applied, my system ("A Blind-Search System, presented in Chapter 4) becomes resilient to churn. This is done not only through reducing the occurrences of failure, though also through optimising the placement of replica within the network such that the impact of failure is considerably lower. In Chapter 6 simulation results for the optimisations in this chapter are presented.

6

EXPERIMENTATION

This chapter builds upon the previous three chapters. Chapter 3 introduced the problem of Complex Queries in DHTs. The solution of Blind-Search, along with its known weaknesses, was introduced in Chapter 4. A number of optimisations both to Blind-Search itself (Chapter 4) and the underlying DHT (Chapter 5) was presented to combat these weaknesses. In this chapter experimental evidence motivating the work in this Thesis and backing up previous claims is presented.

6.1 INTRODUCTION

DHTs are an important building block for P2P applications because they offer scalable and efficient routing between nodes within a bounded number of hops. Since DHTs are designed to scale to hundreds of thousands, or even millions, of nodes it is important to have a way to test these algorithms as well as the applications which might make use of them, on a large scale. To solve this problem a number of P2P simulation frameworks have been developed, with the most well known being: OverSim [6], P2PSim [26], Peersim [31], and PlanetSim [21].

For this work, OverSim was chosen as the simulation framework. OverSim benefits from a modern modular architecture (described below), making it easier to work with than most alternatives. This also gives it more power and flexibility than most alternatives. Instead of trying to reinvent the wheel, OverSim sits on top of the well known and proven network simulation framework, OMNeT++, and its INET package. Using this, OverSim can run the same simulation code over a simple approximation of an Internet Protocol (IP) network, a full stack simulation of an IP network, or even over a real IP network between real nodes.

A further advantage of OverSim is its development team and community. OverSim is a mature simulation framework, yet is still under active development and maintenance, unlike many alternatives that have not been updated in years.

6.1.0.1 *OverSim*

OverSim [6] is designed as a modular simulation framework, with many common overlay features implemented as part of a generic base overlay class. OverSim provides message passing using Remote Procedure Calls (RPC), and supports both iterative and recursive routing. Applications within OverSim are split into multiple tiers, allowing an application (such as the DHT TestApp) to sit on-top of another application (such as the DHT layer). These applications are implemented as modules and interface with overlays through the Key-Based Routing (KBR) Application Programming Interface (API) [13], which represents basic capabilities common to all structured overlays. The OverSim architecture is illustrated in Figure 6.1.

The main applications provided by OverSim are the KBR TestApp and the DHT TestApp. The KBR TestApp is used to test the KBR services exposed by each overlay, such as routing look-up requests to specific keys. The DHT TestApp is used to test the DHT application, by performing DHT PUT and GET requests with randomly generated data items.

OverSim provides a number of different network models, for both structured and unstructured overlays. At the time of writing, the current version of OverSim is OverSim-20121206, and contains models for Bamboo [50], Broose [65], Chord [59], GIA [11], Kademlia [45], Koorde [37], NICE [7], NTree [22], Pastry [51], Quon [4], and Vast [10].

At the lower layer OverSim provides multiple underlay models to allow simulation in more detail at an extra cost of performance, or abstraction for better performance but less detail in the underlying network. Using the simple network model data packets are sent directly from one node to another by using a global routing table and a delay proportional to a randomly assigned physical location. The INET underlay model includes simulation models for all network layers, including UDP (or Transmission Control Protocol (TCP)) and IP. The single host underlay allows for simulation of a single node, connected to other OverSim instances over a real network.

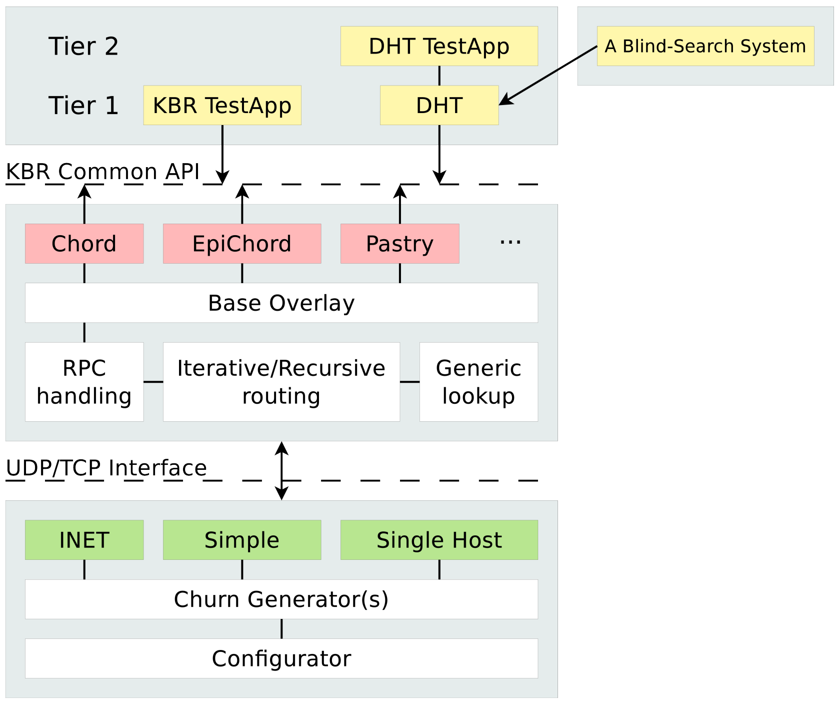


Figure 6.1: Modular architecture of OverSim. Data replication strategies are implemented in Tier 1, and Blind-Search in Tier 2, showing how they are both usable over any routing implementation (such as Chord, EpiChord, Pastry).

6.2 SIMULATION SETUP

Throughout this chapter default OverSim configuration parameters are used, except where specified otherwise. Each experiment is repeated ten times and the results averaged.

The first step in the experimental phase is to validate assumptions around the simulation framework in use. This is split into three main tasks: validating the simulated underlying UDP/IP network, validating the simulated overlaying P2P network, and validating the simulated environment.

6.2.1 *Validating the underlying network*

As discussed above, an OverSim model can make use of multiple underlying network models [6] depending on the accuracy of the simulation that is required, and the computation power that is available.

SIMPLE

> The Simple underlay model is by far the most scalable. This model maintains a global routing table, allowing packets to be sent directly from one node to another. Packets between nodes are delayed by either a constant time, or by placing each node in a two-dimensional Euclidean space and calculating a delay based on the nodes distance from each other. Nodes can also be assigned a bandwidth, access delay, and packet loss ratio. This allows all relevant [6] influences of the underlying network to be simulated with a single event. Due to the simplicity of the Simple model, it allows simulation of a much larger number of nodes.

INET

> The INET model is built on top of the INET package for OMNeT++, which provides simulation of all network layers from the Machine Address Code (MAC) layer. This provides a more accurate simulation than the Simple model, at a much higher cost. The main advantages to the INET model come when working with overlay node placement in the underlying network.

SINGLEHOST

> The SingleHost model allows OverSim to only simulate an individual node, which can be connected to other nodes over a real network. This allows for reuse of existing models when testing on services such as PlanetLab.

The following results in this chapter make use of the Simple underlay network, allowing simulation of much larger scale networks than would otherwise be possible. However, first it is important to confirm that the Simple model does indeed accurately simulate the entire underlying network stack, and that the results taken from an application sitting on top of a Simple model are akin to those taken from an application on top of an INET model.

For these experiments a Chord network was used, with no churn, and network sizes ranging from 50 nodes to 20,000 nodes. OverSims KBRTestApp was used, with a one-way test.

The first set of results compares the lookup latency across different network sizes.

In this experiment, Figure 6.2 shows that in fact the simple network (modelled using the SimpleUnderlayNetwork class) does not provide a perfect model of the INET network (as modelled using the InetUnderlayNetwork class), with lookup latency following a similar shape to that of the InetUnderlayNetwork, but at an elevated level of almost 3 times the expected.

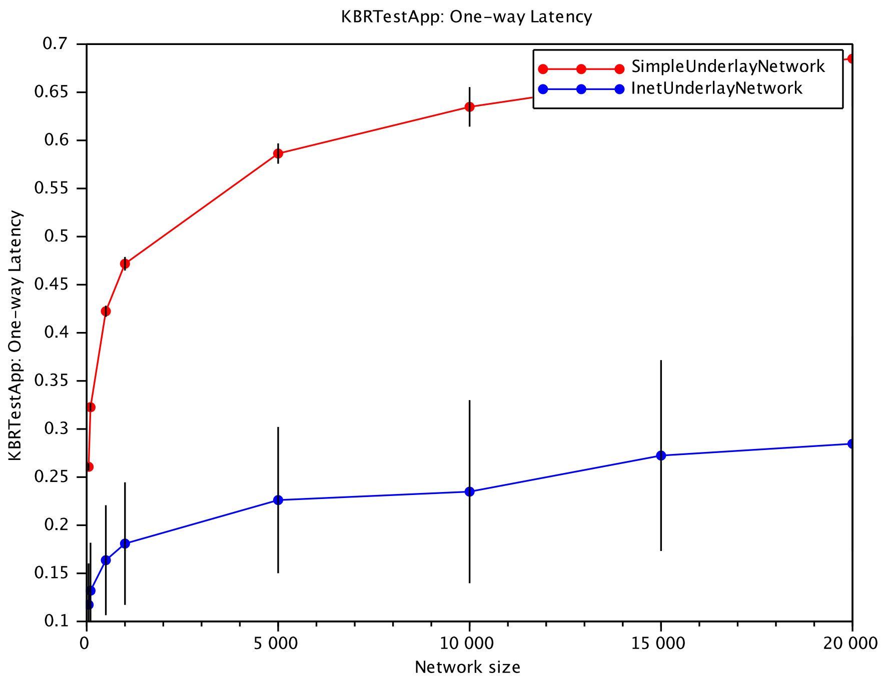


Figure 6.2: Lookup latency of SimpleUnderlayNetwork against InetUnderlayNetwork in OverSim, showing an almost linear discrepancy in results.

However, when looking at metrics purely based on performance of the overlay network there is, as expected, no difference between the SimpleUnderlayNetwork and InetUnderlayNetwork. Figure 6.3 shows the lookup hop count to be exactly the same.

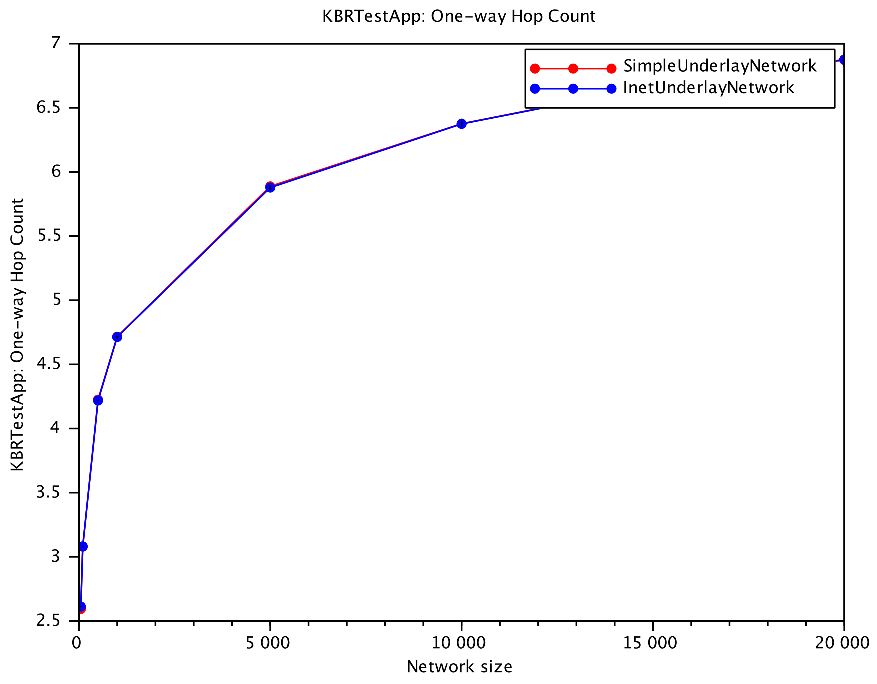


Figure 6.3: Hop count of SimpleUnderlayNetwork against InetUnderlayNetwork in OverSim, showing both SimpleUnderlayNetwork and InetUnderlayNetwork results are identical.

The last set of results, shown in Figure 6.4, in this experiment look at the bandwidth consumption of the overlay, when using SimpleUnderlayNetwork compared to InetUnderlayNetwork. As expected, there is no significant difference to be seen.

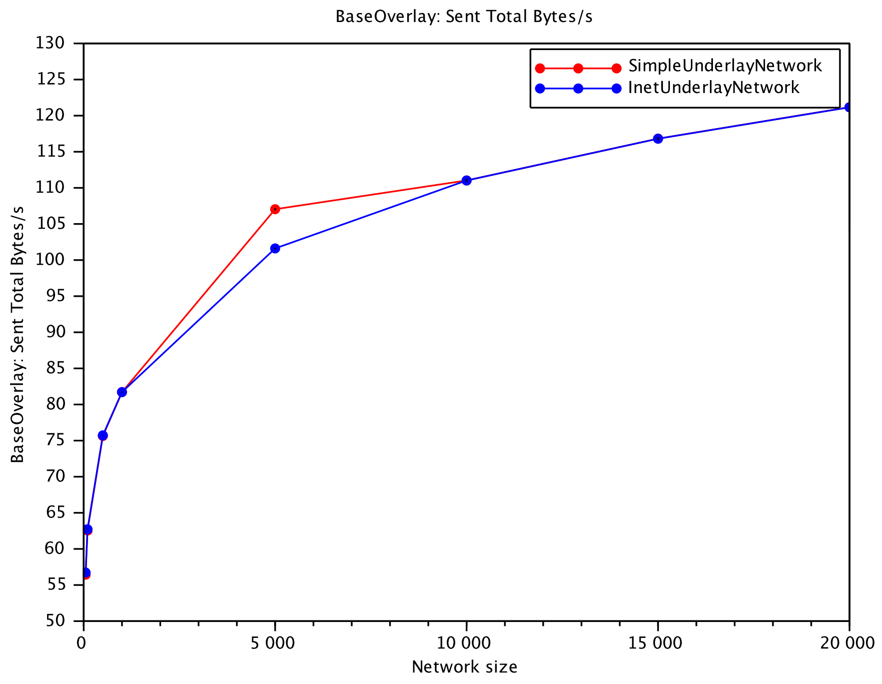


Figure 6.4: Bytes sent from SimpleUnderlayNetwork against InetUnderlayNetwork in OverSim, showing no significant differences.

In conclusion, this experiment shows that the SimpleUnderlayNetwork is indeed not a perfect model of real IP/UDP traffic, and will provide different latency results than the InetUnderlayNetwork. However, it also showed that for purely overlay-based metrics the underlying network model makes no difference. Hence using the more efficient SimpleUnderlayNetwork is acceptable.

6.2.2 *Validating the overlaying network*

As well as validating the underlying network model, the P2P overlay models in use in this Thesis must also be validated. These models include: Chord (Section 2.2.1.1), Kademlia (Section 2.2.1.2), Pastry (Section 2.2.1.3), and Epichord (Section 2.2.2.2). This validation has been published separately in 'An Evaluation of EpiChord in OverSim' [18] and 'An Evaluation of Chord and Pastry Models in OverSim' [19], and is included in Appendix A.

6.2.3 *Validating the environment*

The main environmental factor within the simulated environment is the distribution and rate of churn - that is, how often and in what pattern nodes are created and destroyed in the simulated network. In OverSim, churn can be simulated using five different models or a combination there of.

NOCHURN

The NoChurn model should not be considered as a churn model, though technically it is. Nodes are added to the network until it reaches a defined size, after which it provides a stable environment with no churn. It is the default churn model used by OverSim.

TRACECHURN

The TraceChurn model reads events from an input file and models the network churn accordingly. This allows importing of very specific churn patterns which cannot be generated by an algorithm. Often this is used for either replaying a captured real-world scenario or testing uneven patterns of churn, such as a simulated network split.

RANDOMCHURN

The RandomChurn model is the most basic algorithmic model for simulating churn, whereby nodes are randomly created or destroyed according to a schedule and defined probability value. The nodes destroyed are chosen at random, hence no bias is given to the properties of the chosen node.

LIFETIMECHURN

The LifetimeChurn model calculates the nodes lifetime upon creation using a given probability function. The node then has its death scheduled according to the calculated lifetime. Upon death, a deadtime is calculated, after which a new node will be added back in to the network. Supported probability functions are weibull, pareto shifted, and truncnormal.

PARETOCHURN

The ParetoChurn model is similar in result to the LifetimeChurn model, but approached in a two stage process, as described by Yao et al. [70].

The following results in this section all make use of the LifetimeChurn model using the weibull probability function, as recommended by Steiner et al. [58], Stutzbach and Rejaie [61], and Nurmi et al. [47].

Steiner et al. [58] also present results showing lifetime mean in the real world P2P networks Gnutella, Kad, and BitTorrent. In Gnutella and Kad they show that roughly 90% of nodes have a lifetime that lasts 5 minutes or longer, however less than 50% last over half an hour. BitTorrent fares much worse, with less than 90% nodes having a lifetime lasting for even 1 minute, and only 20% making it to half an hour. For modelling BitTorrent Steiner et al. suggest a weibull distribution with $\gamma = 40$. For Gnutella and Kad no figures are given, so $\gamma = 60$ is taken as a reasonable lower bound for experiments.

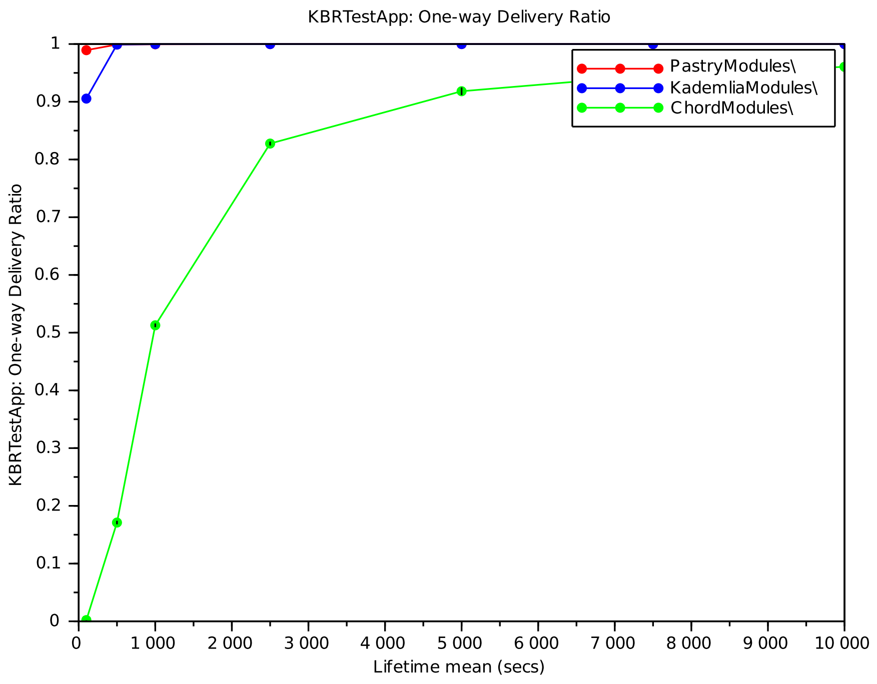


Figure 6.5: Success rate under high churn for Chord, Pastry, and Kademlia.

Figure 6.5 shows the effect of LifetimeChurn using the weibull probability function on varying network sizes. This provides a baseline for the expected effect of churn, as well as confirming that the churn function is behaving as expected. We see that in fact the success rate of Chord under high churn is incredibly poor, dropping to almost total message failure at a churn rate of 2 minutes, where-as both Pastry and Kademlia cope much more reasonably, with a delivery ratio of over 90% even under churn as high as lifetime mean of 2 minutes.

In summary, the simple underlay model in OverSim provides a reasonable estimated simulation of a real UDP/IP network, though results relating to metrics from the UDP/IP itself are not as accurate as the INET underlay model. The overlay models of Chord, Kademlia, Pastry, and Epichord have all been independently validated against the authors original models, and proven to provide accurate metrics. The different churn models in OverSim have been evaluated, and the Lifetime churn model with a Weibull distributed chosen for future use.

6.3 BLIND-SEARCH

First the basic Blind-Search algorithm is implemented without any optimisations, proving the correctness of the algorithm and providing a baseline for measuring optimisations against. The following results are taken using the overlays Chord and Pastry. A search is considered successful when the node containing data is successfully contacted with the search query (routing of responses is not yet considered).

In these experiments the network size is varied to investigate how Blind-Search scales. There is no churn introduced yet.

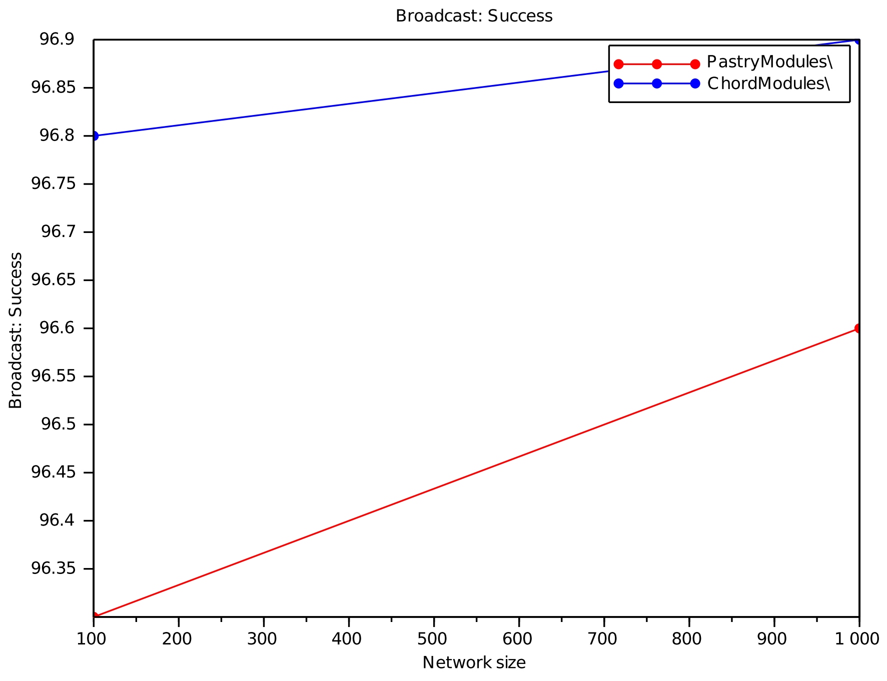


Figure 6.6: Blind-Search success rate for Chord and Pastry without churn, under 100% due to network stabilisation.

Figure 6.6 shows that Blind-Search without churn can obtain a high query success rate. The success rate is between 96% and 97% for both Chord and Pastry. Although there is no churn the network requires a certain length of time to stabilise fully after creation, while maintenance runs and routing tables become fully populated. When altering the transition time (that is, the initial time given for the network to stabilise before collecting results) this success rate raises and lowers, confirming the hypothesis that the lower than 100% success is due to routing tables continuing to stabilise.

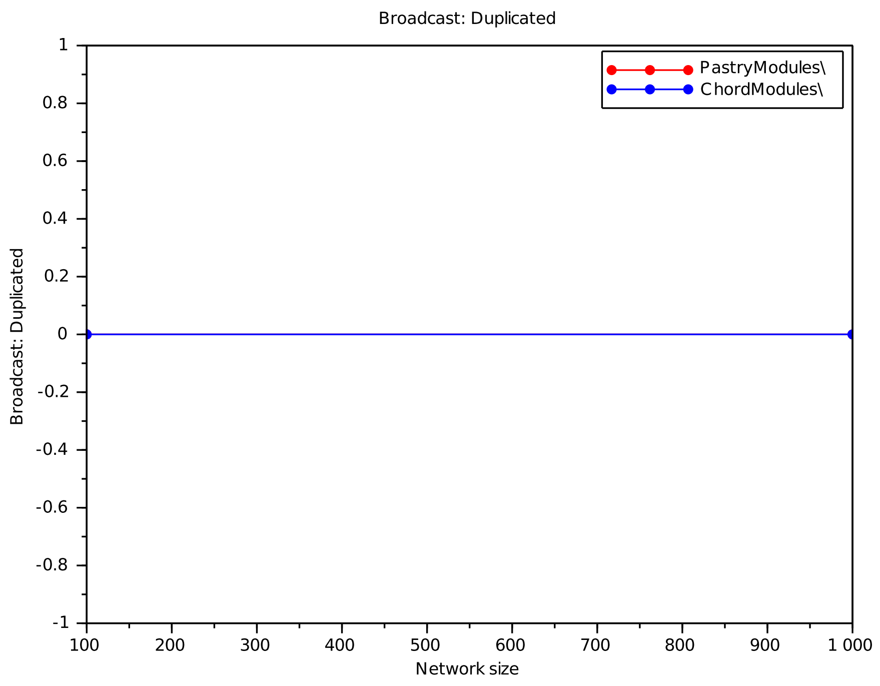


Figure 6.7: Blind-Search message duplication for Chord and Pastry, showing zero message duplication in both cases.

In Figure 6.7 the number of duplicated messages is shown to be zero for all network types and sizes. This, combined with the success rate, shows that the algorithms described are correct.

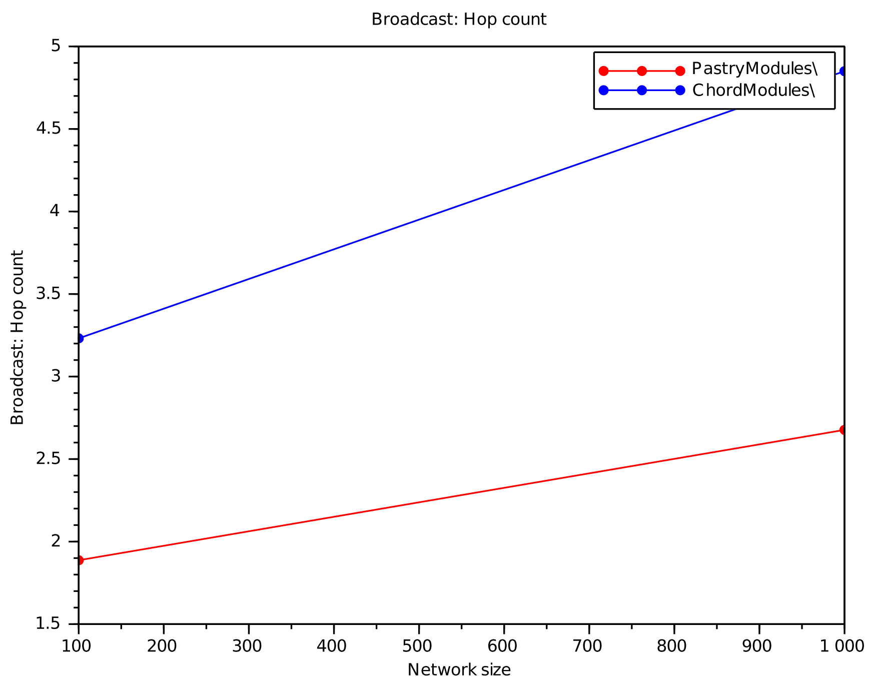


Figure 6.8: Hop count of Blind-Search for Chord and Pastry.

Figure 6.8 shows that the mean hop count for Blind-Search queries grows logarithmically with network size, hence at least in terms of query time, these algorithms do indeed scale.

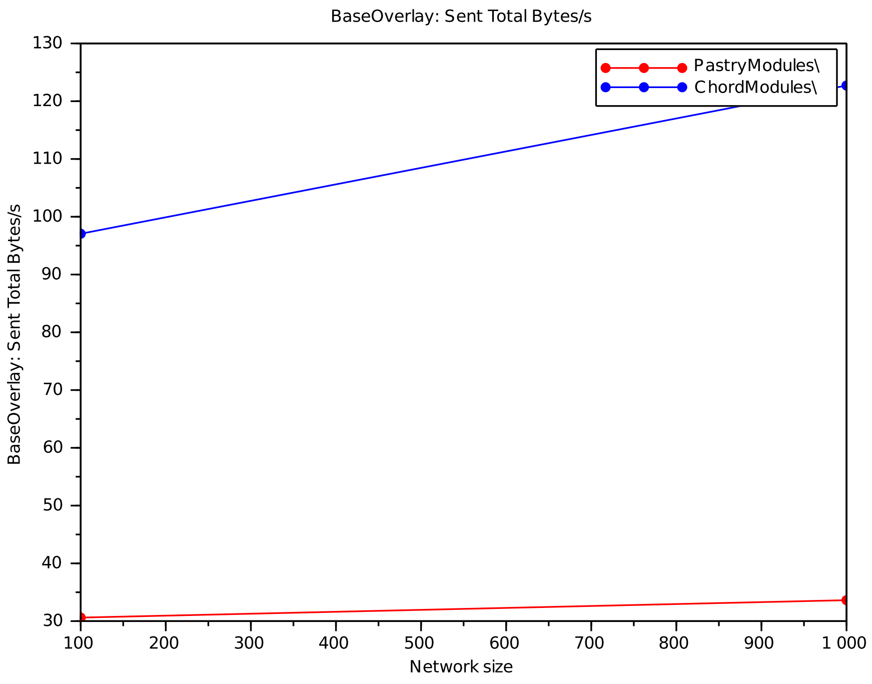


Figure 6.9: Bandwidth usage during Blind-Search for Chord and Pastry.

Comparing the mean total bytes sent per node during Blind-Search, Figure 6.9 shows that the mean bandwidth usage also grows logarithmically with network size.

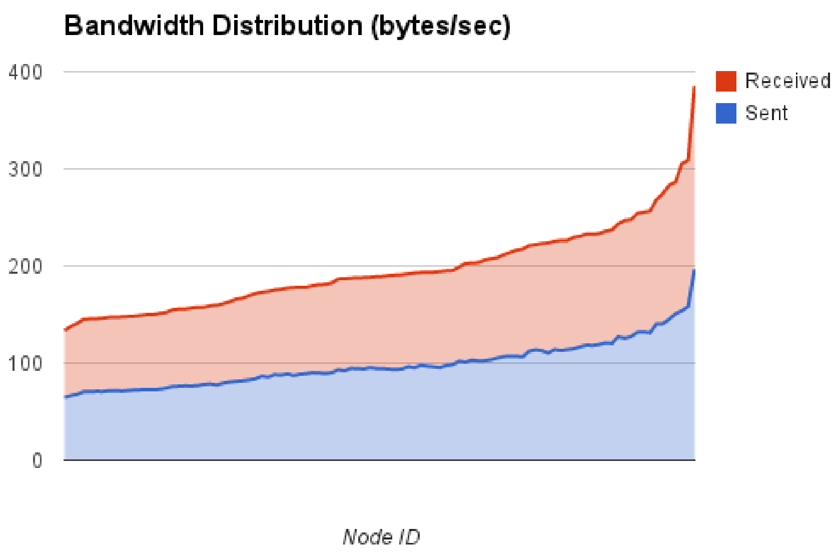


Figure 6.10: Bandwidth distribution during Blind-Search for Chord and Pastry.

Figure 6.10 shows that although the bandwidth usage not high, it is also not balanced across the entire network. Looking at a single run of Chord, while the majority of nodes have a similar bandwidth usage, there are a few outliers responsible for double that of others. These are the nodes near the top of the broadcast tree, with the highest fanout.

6.3.1 *Performing under churn*

The final step for evaluating Blind-Search is to introduce churn.

In Figure 6.11, churn with a lifetime mean ranging from 2 minutes to 2 hours in introduced. It shows that as the churn rate increases, the success rate of Blind-Search dramatically decreases, becoming almost useless when the lifetime mean gets to minutes.

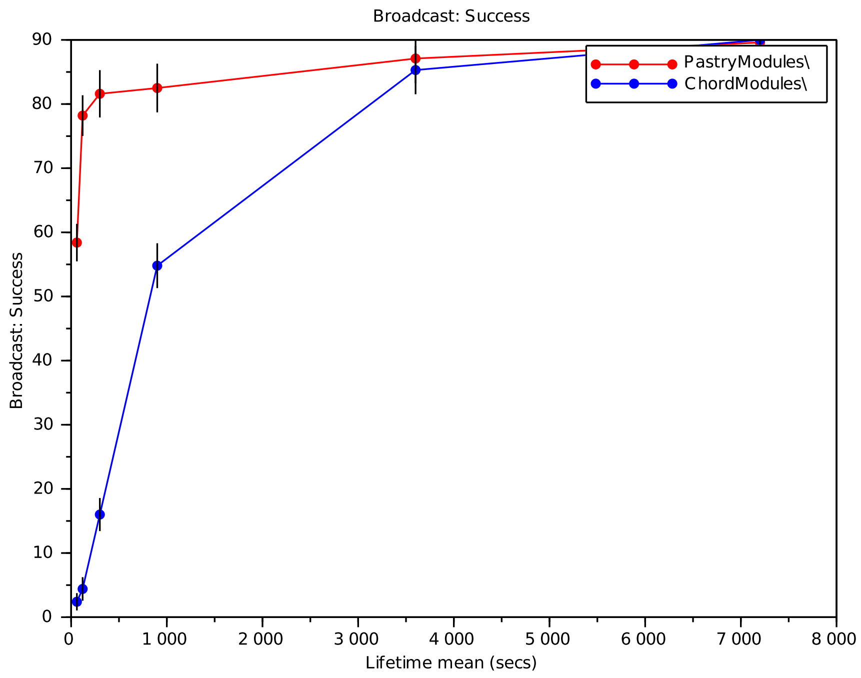


Figure 6.11: Blind-Search success rate under high churn for Chord and Pastry, showing a lifetime mean of less than 1 hour has a huge impact on success rate.

Overall, the results presented show that Blind-Search can successfully be performed over Chord and Pastry networks, despite their fundamentally different design and network topology. Under optimum conditions a success rate of close to 100% can be achieved with no redundant messages, with minimal bandwidth usage and a logarithmic hop count.

We see that by varying the branching factor during Blind-Search, the bandwidth consumption can be shared more fairly across the network at the cost of overall hop count, and hence latency.

Introducing churn, however, highlights the weakness of the efficient broadcasting algorithms, whereby a single node failure can have a dramatic effect on a section of the network. Hence high node failures rates cause incredibly poor Blind-Search success rates. In the following section a solution to this using data replication is presented.

6.4 OPTIMISING DATA REPLICATION

As discussed in Section 5.3, DHTs can be optimised for Blind-Search by altering the data replication strategy in use.

6.4.1 *Symmetric Replication*

The largest gain in query success rate under churn can be obtained by making use of Symmetric Replication instead of the more traditional Neighbour Replication, used by Chord, Pastry, and Kademlia by default.

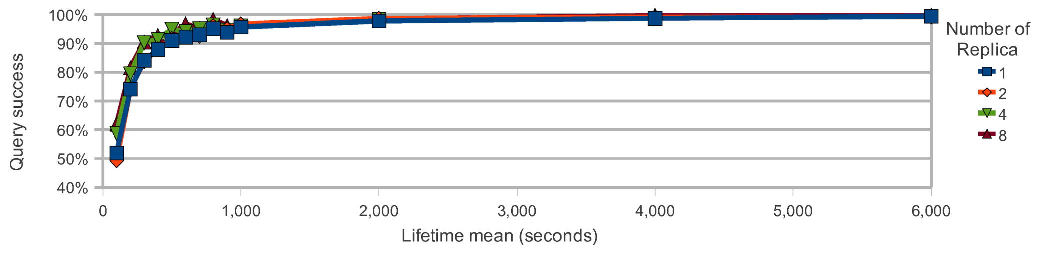


Figure 6.12: Success rate of Blind-Search over Chord with Neighbour Replication, showing no significant difference in success rate with number of replica.

Figure 6.12 shows the success rate of Blind-Search over the Chord DHT, with varying degrees of data replication, using the default Neighbour Replication. This shows that the degree of replication when using Neighbour Replication has no significant effect on the success rate of Blind-Search - making the replication redundant in this use case.

Figure 6.12 also shows that with a churn rate of 2 minutes the success rate drops as low as 50%. Realistically, any churn rate with a lifetime mean under 15 minutes causes the success rate to drop below 90%, and this approach becomes unreliable.

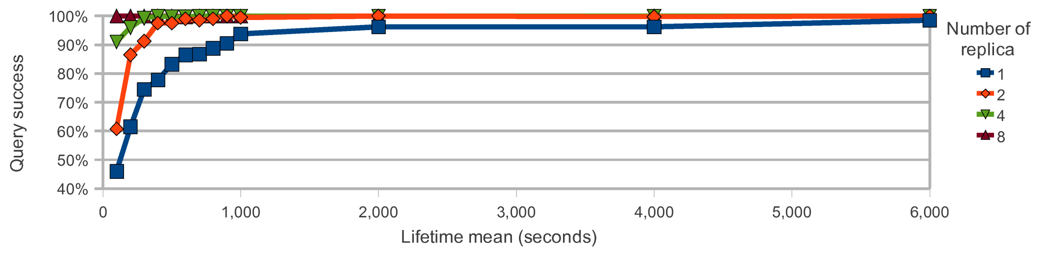


Figure 6.13: Success rate of Blind-Search over Chord with Symmetric Replication, showing 100% success rate being achieved using 8 replica.

When the Neighbour Replication is replaced with Symmetric Replication, Figure 6.13 shows that the degree of replication now has a dramatic effect on the success rate, increasing the success rate of Blind-Search by roughly 30% per replica. Using Symmetric Replication 8 replica able to keep a Blind-Search success rate of 100% even with a lifetime mean of as low as 1 minute (compared to a success rate of 50% with 8 replica using Neighbour Replication).

These results show that the use of Symmetric Replication over Neighbour Replication has huge benefits for Blind-Search, and in fact for Blind-Search to work reliably under churn could be considered a requirement. Because of this it is argued that the data replication strategy used should not be determined by the network type, but considered as an extra layer on top of the DHT which should be chosen to suit the requirements.

6.5 DISTRIBUTED HASH TABLE OPTIMISATIONS

In Chapter 5 optimisations to DHTs which can be made to benefit Blind-Search were discussed. Experimental results for these optimisations are now presented.

6.5.1 *Increasing routing table size*

6.5.1.1 *Address caching*

The below simulations were performed using the proposed NKademlia algorithm, in which an address cache is used allowing growth of the routing table up to a set limit. For example, NKademlia 2048 defines a maximum routing table size of 2048 nodes. NKademlia is described in more detail in Section 5.1.1.

Parameter	Value(s)
key length	512bits
network size	10,000
nodes per bucket (k)	8
bucket refresh	3,600s
max stale count	2
bits to consider (b)	1
lifetime mean	1800s, 3600s, 7200s, 14000s
measurement time	1,200s
lookup rate	10s
routing type	iterative
max routing table size	$64 \ldots 2048$

Table 6.1: NKademlia: Simulation parameters.

In all NKademlia results I compare against both Kademlia and NR128. NR128 [32] is the state of the art optimisation to Kademlia in which the routing table bucket responsible for the most nodes is increased to hold 128 entries, with each subsequent bucket half the previous (until a minimum size of k).

From these simulations we observe a reduced lookup latency compared to regular Kademlia, shown in Figure 6.14, with the latency decreasing as the routing table size increases. With a network size of 10,000 nodes the latency settles with a maximum routing table size of around 1,000 nodes, with further increases in routing table size having little effect.

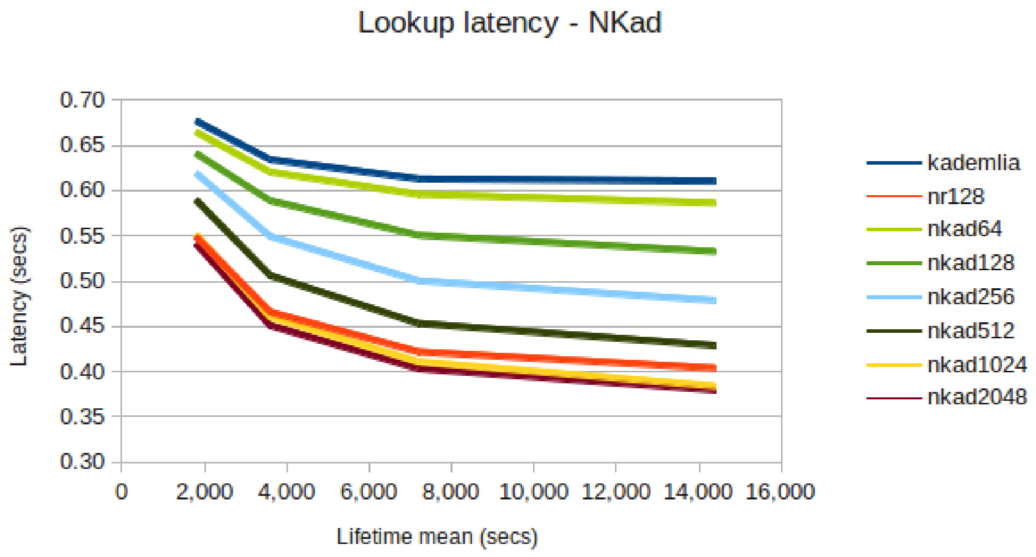


Figure 6.14: Nkademlia: Lookup latency under churn, showing an address cache as small as 64 nodes even has a performance improvement over Kademlia in a 1,000 node network.

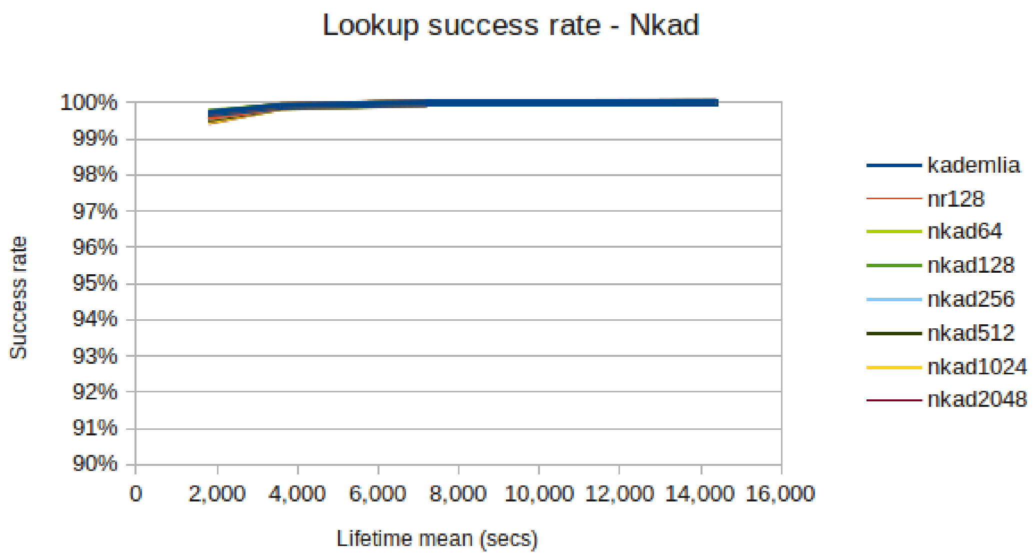


Figure 6.15: Nkademlia: Lookup success rate under churn, showing address caching has no significant effect.

Figure 6.15 shows that the increased routing table has no effect on the success rate of lookups, as is to be expected.

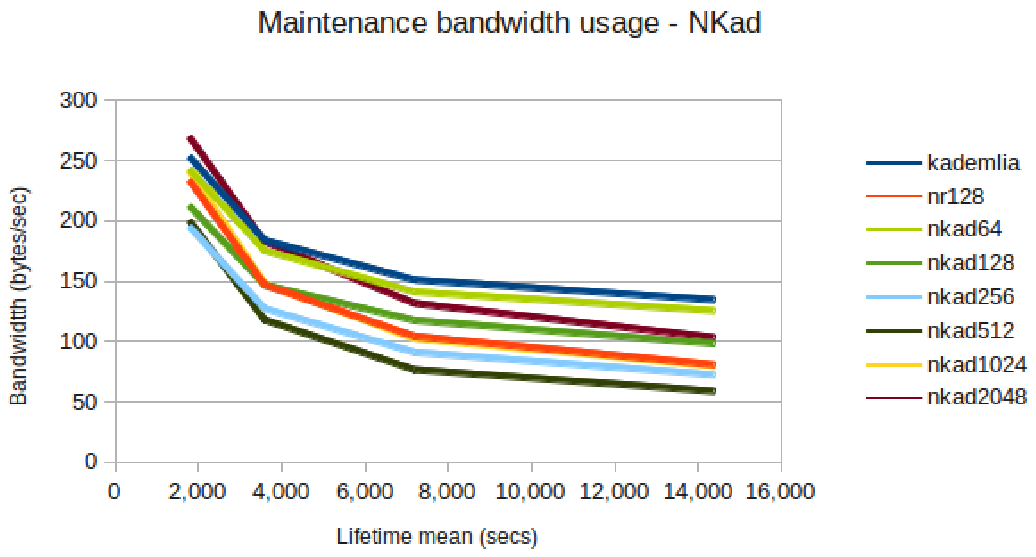


Figure 6.16: Nkademlia: Maintenance bandwidth usage under churn, showing an address cache as small as 64 nodes even provides a reduction in maintenance bandwidth in a 1,000 node network.

Figure 6.16 shows there is reduction in bandwidth when using a higher maximum routing table size. This is because the extra nodes in the routing table do not cost anything as they are purely a cache, and no active maintenance is performed on them. However, when one of the primary nodes in a bucket is ejected, the bucket is automatically refilled without any additional maintenance. In the best case, this node is alive and no maintenance was required. In the worst case this node is dead and another must be chosen, again requiring either no maintenance or a regular lookup if the cache is now empty. Additionally, this lower bandwidth usage can also be attributed to a lower number of messages being sent during a lookup, as the hop count reduces.

Overall, the NKademlia system, when compared to regular Kademlia or NR128, provides lower latency for regular lookup operations, a higher branching factor for Blind-Search, and reduced maintenance costs.

6.5.2 *Increasing routing table accuracy*

Increasing the routing table size shows a significant decrease in lookup latency, at the cost of decreased routing table accuracy, as can be seen in Figure 6.17. For regular lookup operations this does not appear to have much effect on success rate. However, for Blind-Search this reduced accuracy will have a major effect, similar to the effect reduced accuracy due to high churn rates has.

6.5.2.1 *Levelling*

The first approach for increasing routing table accuracy is the use of Levelling maintenance, in which periodically a random key is chosen, and the responsible node is sent a ping message to ensure liveliness. Levelling is described in more detail in Section 5.2.2.

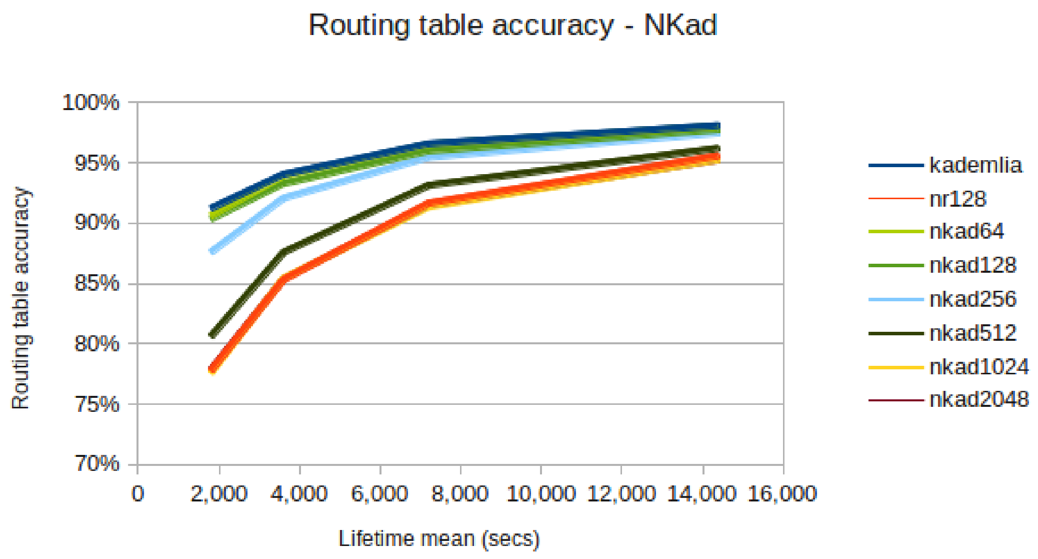


Figure 6.17: Nkademlia: Routing table accuracy under churn. Although previous figures showed NKad results in improved performance, this figure shows that in fact routing table accuracy is lower.

As shown in Figure 6.17 having greater than k nodes in a bucket causes Kademlia maintenance to become ineffective and routing table accuracy to drop. Next, the previous simulations is re-ran over a longer time period, measuring how the routing table accuracy and lookup latency change over time.

Parameter	Value(s)
key length	512bits
network size	10,000
nodes per bucket (k)	8
bucket refresh	3,600s
max stale count	2
bits to consider (b)	1
lifetime mean	7,200s
measurement time	12,000s
lookup rate	10s
routing type	iterative

Table 6.2: Nkademlia: Simulation parameters.

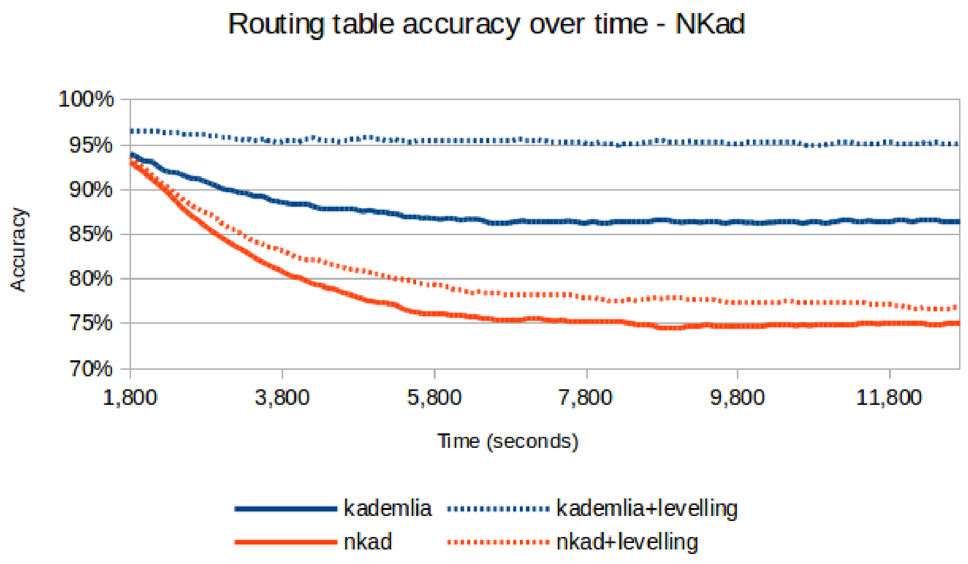


Figure 6.18: Nkademlia: Routing table accuracy over time, with and without the use of Levelling.

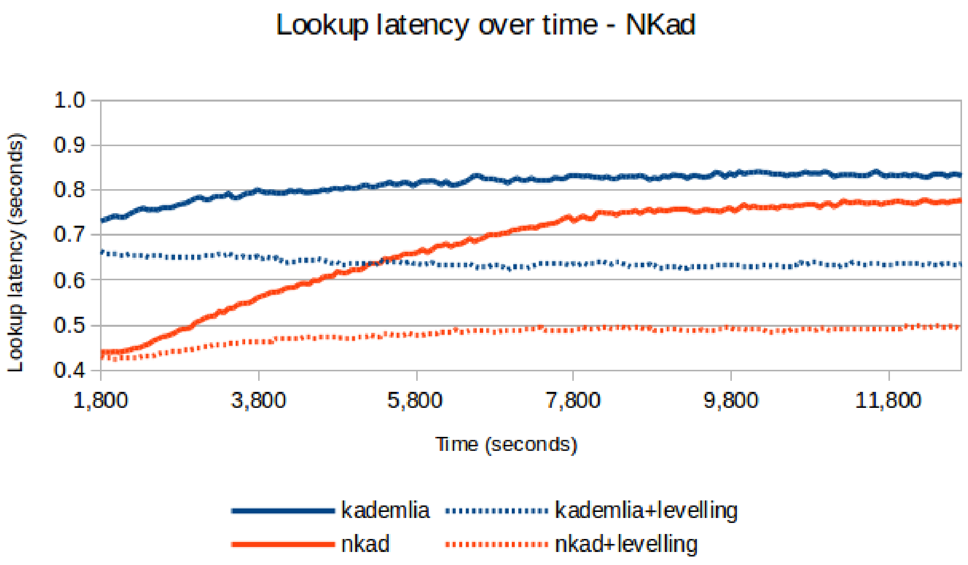


Figure 6.19: Nkademlia: Lookup latency over time, with and without the use of Levelling.

As seen in Figure 6.18 and Figure 6.19 the routing table accuracy of Nkademlia drops to a much lower level than that of regular Kademlia, however this does not seem to negatively impact the lookup latency. It is also shown that the introduction of Levelling maintenance helps keep the routing table accuracy and lookup latency at a steady level over time, compared to the upwards curve for lookup latency when using NKademlia without Levelling maintenance.

By making use of address caching in DHT it is possible to reduce the average hop count for requests, both in terms of regular lookups and Blind-Search. The side effect of this is reduced maintenance costs. By introducing techniques such as Levelling, the accuracy of the routing table over time can be dramatically improved, leading to higher success rates and lower hop counts, again both in regular lookups and Blind-Search.

6.6 SUMMARY

In this chapter I first justified the choice of OverSim as the P2P simulation framework, then provided validation of the techniques and models that are used in the remainder of this chapter. The main advantage of OverSim is its modular design, allowing it to reuse the industry standard network simulator OMNeT++ as a base. This modular design also allows for plugging together of components to build a complete system - for example it is possible to take the widely used Chord, Pastry, and DHT modules, and build just the blind-search module on top. Validation of the used models is provided in independantly published papers, but also included in Appendix A.

Simulation results for my Blind-Search System (see Chapter 4), first without churn, were presented. These showed a query success rate of between 96% and 97%, with no duplicated messages, and logarithmic growth in hop count as network size increases. These were as expected, and showed that without churn the system is a success.

When the system was placed under high churn the results showed the query success rate drop to between 2% and 58%, confirming the hypothesis that any efficient broadcast based system is highly susceptible to churn.

The first, and most significant in the context of blind-search, optimisation for which results are presented is the replacement of Neighbour Replication with Symmetric Replication.

Simulation results of A Blind-Search System over Chord with Neighbour Replication showed a success rate of 50% under high churn, and no significant difference between 1 replica, 2 replica, 4 replica, and 8 replica. Replacing with Symmetric Replication, the success rate ranges from 50% to 100% - with 4 replica providing a success rate of 90%, and 8 replica a success rate of 100%.

These results showed that while indeed A Blind-Search System is highly susceptible to churn, by simply replacing the default Neighbour Replication in many DHTs with a more suited Symmetric Replication the impact of failures can be greatly reduced, to the point where the system can efficiently operate even under high churn.

The next optimisation presented was Address Caching (see Chapter 5), in which nodes maintain a cache of addresses for other nodes they have passively discovered. This cache is then used to refill the routing table, as well as supplement it to allow an increased branching factor during blind-search.

Compared to Kademlia and NR128 [32], Kademlia with Address Caching (referred to as NKademlia) significantly reduced lookup latency by up to 40% over Kademlia and 8% over NR128. Maintenance bandwidth consumption was also significantly reduced when compared to Kademlia, also by up to 40%.

While Address Caching was originally proposed primarily to increase the possible branching factor, in the context of blind-search, these results show that it has far wider applications than just blind-search.

Results for Downlists and Levelling (see Chapter 5) were presented, showing that without either routing table accuracy of both Kademlia and NKademlia decreases over time, and hence the lookup latency increases over time. With the introduction of Downlists and Levelling the routing table accuracy is shown to stay constant, resulting in a constant lookup latency over time.

Overall, through simulation this chapter has shown that this system meets the original design criteria of efficient distribution with minimal duplicate messages. It has shown that churn is a huge problem for any system based on efficient broadcasting, and then shown that the solutions proposed in Chapter 5 do indeed solve this issue for A Blind-Search System.

7 CONCLUSIONS

The goal of this thesis was to design an efficient method for performing complex queries, which will work across a large range of DHT types, in a heterogeneous and high churn environment. Towards this goal, a review of P2P networks and complex queries were first presented in Chapters 2 and 3.

These reviews showed that while there are a large number of search systems built on structured P2P networks, there is only a single family of systems that is able to support all types of complex queries - those built on the principal of efficient broadcasting. Efficient broadcast uses the structure of the network to broadcast a message to all nodes, with minimal duplicates. It also showed that the majority of existing systems were very prescriptive, requiring the use of a specialised DHT structure, which would trade off certain properties such as load balance to achieve their goals. Of the systems which were capable of supporting all types of complex queries, only one was a generic solution capable of working on multiple types of DHT - and this system had no resiliency to churn.

7.1 CONTRIBUTIONS

7.1.1 *A Blind-Search system*

In Chapter 4 my Blind-Search System was presented, bringing together the state of the art in broadcast techniques from different underlying networks and adapting them for use in a search system. This provided a generic system that works across different DHT types, supports all complex query types, and produces minimal redundant messages.

Simulation results, shown in Chapter 6, showed that in a stable environment this system could successfully support all complex query types in an efficient manner. However, without the below contributions, this system was susceptible to overloading the search originator during searches for populate services, and to degraded performance under high levels of churn.

After the below optimisations were applied, the system was capable of performing all types of complex queries with a success rate of 100% even under high churn, and minimal redundant messages.

7.1.2 *Novel query response routing algorithm*

An observation made from the above Blind-Search system, is the ability to flood the origin node when performing a popular search, with as many as $O(N)$ response messages. To solve this problem, a novel query response routing algorithm was presented in Chapter 4, which uses reverse tree navigation and message collation to reduce load, especially in searches with a large number of positive responses.

With this query response routing algorithm, this Thesis has advanced the state of the art in routing of search results within DHTs and allowed the proposed Blind-Search system to operate without the risk of overloading nodes performing searches.

7.1.3 *Advancements in maintenance algorithms*

Optimisations to the routing table in DHTs were evaluated, with multiple generic optimisations presented in Chapter 5 which improve success rate and performance, while reducing overall maintenance traffic.

These optimisations included increasing the routing table size through Address Caching - keeping a cache of all nodes that an individual learns about through passive observation. This address cache can then be used for replacing missing nodes in the primary routing table when they are lost due to maintenance, as well as supplementing the routing table during Blind-Search. Results presented in Chapter 6 showed an up to 40% decrease in both lookup latency and maintenance bandwidth when Address Caching is introduced in a Kademlia network.

A further optimisation increased the routing table accuracy, through the use of Downlists - in regular communication, nodes include an extra field with their messages, including a list of recent nodes they have detected as being offline. Receiving nodes can verify this information with a ping message, then update their routing tables immediately rather than waiting for a failed request of their own.

A final optimisation called Levelling helped to distribute maintenance traffic evenly over the entire network rather than evenly over the set of buckets in Kademlia. This resulted in an improved routing table accuracy, especially in combination with the address cache.

The combination of these optimisations was presented in Chapter 6, showing improvement to the routing table accuracy, and hence success rate and performance of Blind-Search, and actually reduced background maintenance costs.

7.1.4 *Advancements in DHT data replication*

This Thesis has argued for treating DHT data replication separately from the routing algorithm, and in Chapter 6 has shown that by making use of Symmetric Replication instead of Neighbour Replication during blind-search it is possible to increase the success rate from as low as 50% to 100% at no additional cost. This is achieved through the observation that neighbours in the DHT are likely to belong to the same branch of a broadcast tree, and hence in the context of blind-search, Neighbour Replication provides no redundancy.

The state of the art in DHT data replication algorithms was advanced through the introduction of Replica Teams. Replica Teams allow the use of types of replication algorithms other than Neighbour Replication, while keeping the lower maintenance cost of Neighbour Replication. This is achieved by splitting replica in to teams rather than individual nodes and distributing the teams per the replication algorithm.

Bringing together these contributions results in A Blind-Search System, capable of performing all types of complex queries across a large range of different DHT types. This system can perform searches for both rare and common services/documents, with efficient response routing that will not overload the search originator.

By making use of Symmetric Replication using Replica Teams and the DHT maintenance optimisations discussed, the success rate and performance of Blind-Search can be kept at a high level even under extreme churn rates, without additional cost or sacrifice to regular DHT operations.

7.2 LIMITATIONS

7.2.1 *Real world deployment*

The work in this thesis has been backed through extensive simulations, using the well known simulation engine OverSim. The environment is validated in Chapter 6, showing that while simulated latency results are likely to be inaccurate, the actual algorithms are correctly implemented, and hence results around correctness and success of queries can be trusted.

As it is unfeasible to deploy a test network of sufficient scale for testing purposes there has been no testing of the work presented in real world deployments. It may be interesting to evaluate query latency, as well as its effect on success due to message timeouts. However, the effect on this system, if any, is likely to just look like an increase in churn rate.

7.3 FUTURE WORK

In this section potential future research questions posed by this work are discussed.

7.3.1 *Scoping the search*

In search, there are 2 primary desired outcomes: finding a matching result, and finding all matching results.

In this Thesis the focus is on finding all matching results. Once a search is initiated it is not considered complete until it has reached as many nodes within the network as was possible.

A potential future research direction is the investigation of support for searches in which the desired outcome is only to find a single matching result. In this case, the search can be optimised using techniques such as Go+Stop Signals [66] and TTL scoping [44], to terminate as soon after a successful match as possible. In this case data replication plays a role in not just redundancy, but also efficiency and performance. There is a tradeoff to be made with a replication ranging from $O(1)$, providing results in $O(N)$ queries, to a replication factor of $O(N)$, providing results in $O(1)$ queries.

7.3.2 *Branching factor in heterogeneous networks*

In Section 4.4 the load balancing of Blind-Search was discussed, in which the branching factor (the number of children a node decides to send the query to) plays a large role. In some networks, such as Chord, the branching factor is relatively fixed due to the amount of nodes in the routing table, however in others such as EpiChord or D1HT, the branching factor can be as high as $O(N)$.

In heterogeneous networks, some nodes will have an abundance of resources, so forwarding the query to a large number of children is not a problem. On the other hand some nodes, perhaps mobile devices, will have severely limited resources and will struggle to forward the query. By exploiting this observation it may be possible to design a system which dynamically varies the branching factor based on a nodes self-identified resources. This would result in a more manageable load for all participants and increased performance in the presence of high resourced nodes.

7.3.3 *Address cache staleness*

The concept of an address cache (Section 5.1.1) was introduced in this thesis. It allows much cheaper and faster replacement of nodes removed from the primary routing table, resulting in lower maintenance costs and higher success rates. However, over time this address cache is likely to fill with stale nodes, resulting in dead nodes in the primary routing table being replaced by dead nodes from the address cache.

Future work could investigate the use of a cheap maintenance algorithm, perhaps such as Levelling maintenance, to be used on the address cache, ensuring liveliness of its contents.

7.3.4 *Downlists with bloom filters*

Including Downlists (Section 5.2.1), lists of nodes that are known to have gone offline recently, in regular communication has been shown to provide improvement in routing table accuracy, at the cost of increased bandwidth usage during regular communication.

Bloom Filters provide a way of very efficiently encoding a set, such that the receiver can be confident an element is not in the set, with a low rate of false positives. Considering the Downlists algorithm already makes use of a ping operation on each node it considers to be down to confirm received data is valid, these false positives would be acceptable, and bloom filters may provide a dramatic bandwidth reduction in DHTs implementing Downlists.

7.3.5 *Data replication factor*

While this thesis investigated the use of different data replication strategies, and proposed the use of a new concept, Replica Teams, the comparisons of different strategies was always relative. The work in this thesis did not aim to recommend specific values for data replication that would be required to achieve specific Service Level Agreements (SLAs) for success rate or performance.

7.4 SUMMARY

In this chapter the key achievements and research contributions of this Thesis have been highlighted, along with the limitations and open research questions which surfaced.

This Thesis presented a novel, generic, Blind-Search system which can efficiently perform all types of complex queries over multiple DHT types. This is achieved through the use of efficient broadcasting algorithms, and advancements in both DHT maintenance and data replication. Over existing systems, A Blind-Search System provides support for all types of queries rather than just a subset, in a generic system that can be applied over any existing DHT type. Due to the optimisations presented in Chapter 5 this system is the only such system that is also resilient to high levels of churn, and optimised for use in heterogenous networks.

To conclude, the research presented in this Thesis has extended the state of the art in DHT complex queries, blind-search, data replication, and maintenance. It met the requirements and objectives outlined in Chapter 1, and finally offered potential future directions for continuation of work in the area of Blind-Search.

APPENDIX A

- Jamie Furness, Mario Kolberg, and Marwan Fayed. An Evaluation of Chord and Pastry Models in OverSim. In *Modelling Symposium (EMS), 2013 European*, pages 509–513, 2013
- Jamie Furness, Farida Chowdhury, and Mario Kolberg. An Evaluation of EpiChord in OverSim. In *5th International Conference on Networks and Communication*. ACM Press, 2013

An Evaluation of Chord and Pastry Models in OverSim

Jamie Furness, Mario Kolberg, Marwan Fayed
Computing Science and Mathematics
University of Stirling
Stirling, Scotland
{jrf,mko,mmf}@cs.stir.ac.uk

Abstract—**Peer-to-peer (P2P) simulation frameworks are excellent tools for developing and testing P2P algorithms, however there has been very little work done on validation of the models within these frameworks. Validation of these models is an important issue, as without knowing the models are valid we can not necessarily rely on the results generated using such models. In this work we provide an independent evaluation of both the Chord and Pastry Distributed Hash Table (DHT) models within OverSim, and validate the models by comparison against results presented in the original Chord and Pastry papers.**

Keywords—Peer-to-Peer overlay, Distributed Hash Table, OverSim, Chord, Pastry.

I. Introduction

Distributed Hash Tables (DHTs) are an important building block for Peer-to-Peer (P2P) applications because they offer scalable and efficient routing between nodes within a bounded number of hops. Since DHTs are designed to scale to hundreds of thousands, or even millions, of nodes it is important to have a way to test these algorithms as well as the applications which might make use of them, on a large scale.

To solve this problem, a number of P2P simulation frameworks have been developed: OverSim [1], P2PSim [2], Peersim [3], PlanetSim [4], and many others. In this work we look at validating some of the models provided within OverSim.
OverSim is an open-source P2P simulation framework that we are using in our ongoing work, based on the discrete event simulation system OMNeT++. OverSim includes both a simple underlay model for larger scale simulation as well as a more detailed underlay model derived from the INET framework of OMNeT++, which allows for simulation of all network layers down to the MAC layer. Also included with OverSim are models of many different overlay networks: Bamboo [5], Broose [6], Chord [7], GIA [8], Kademlia [9], Koorde [10], NICE [11], NTree [12], Pastry [13], Quon [14], and Vast [15].

A. Contributions

As is the case with any simulation, results that rely on a model can only be considered valid if the model itself is valid. We could find no such independent validation, and present this work to fill that gap. The main contribution of this paper is an independent evaluation of both the Chord and Pastry models included with OverSim. We define scenarios, based on parameters taken from the original papers, which test both the routing algorithm and maintenance algorithm for each model. In the first scenario we measure both the success rate and performance (path length) of look-up queries. In the second scenario we measure the look-up success rate after multiple node failures. We compare the results obtained using the models in OverSim against the results presented in the original Chord and Pastry papers.

The remainder of the paper is organized as follows: Section II acknowledges related work. Section III gives a short overview of the Chord and Pastry algorithms. In Section IV OverSim is described, in particular the models for Chord and Pastry. Section V presents results comparing the OverSim models against results from the original papers. We conclude in Section VI.

II. Related Work

There are a number of different P2P simulation frameworks which provide models of the Chord and/or Pastry algorithms, as listed in Section I. Despite the large number of overlay models available the only validation we are aware of considered Chord, and involved comparing only the average path lengths generated by the OverSim model against those generated by the P2PSim model [1], instead of against the original paper. Our work aims to fill this gap by validating both the Chord and Pastry models against results from their original papers.

III. Background

In this section we present an overview of both the Chord and Pastry algorithms for completeness.

A. Chord

Nodes in a Chord [7] network are ordered numerically in a ring structure modulo 2m, where m is a network parameter. Data items are assigned to the first node which has an identifier that is equal to or follows in the ring.

In a network of size N each node maintains routing state information for $O(\log_2(N))$ other nodes, namely the k nodes succeeding it, known as the successors, and a set of finger nodes. The finger nodes are chosen at logarithmically increasing distance around the ring, the ith entry in the table at node n contains the identity of the first node that succeeds n by at least 2^{i-1} $(i \geq 1)$. This means that nodes have a more complete view of the area nearby, with less links to far away nodes in the network. As a message is forwarded around the network, the closer it becomes to the destination the more likely nodes are to have a link to the destination node. An example Chord network showing the choice of finger table nodes can be seen in Figure 1.
Due to the distribution of nodes within a Chord routing table, it is expected that the average path length increases logarithmically with the size of the network.

In the original Chord paper [7] the protocol is evaluated by simulation. Results evaluated the distribution of keys across all nodes within the network, as well as the average path length and distribution for varying network sizes. To evaluate the stabilization protocol look-up failures were compared against node failures. In Section V-A we compare these original results to results obtained using the Chord model within OverSim.

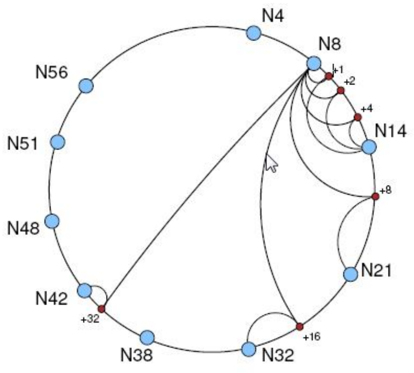


Figure 1: An example Chord network, showing the choice of finger nodes for Node N8.

Pastry [13] also assumes a circular identifier space with each node ordered in a circular name-space modulo 2128. Data items are stored at the node whose identifier is numerically closest to the data items identifier.

Each Pastry node maintains a routing table, a neighbourhood set, and a leaf set (otherwise known as the name-space set). In a network of size N, using identifiers with base 2^b, each nodes routing table is designed with $\log_b(N)$ rows, where each row holds b - 1 entries. All the entries at row r of the routing table each refer to a node whose identifier shares the current nodes identifier in the first r digits, but whose $(r + 1)^{th}$ digit does not match that of the current node. The neighbourhood set contains links to the m closest nodes, and the leaf set contains the k nodes whose identifiers are closest and centred around the local nodes identifier.

The number of nodes traversed while routing a message is expected to increase logarithmically with the size of the network. The maximum path length is expected to be $\log_2{}^b(N)$.

In the original Pastry paper [13] results are presented displaying the average path length and distribution for varying network sizes. Additionally results are presented to evaluate the locality properties of Pastry routes, by comparing the distance a message travels using Pastry with that of a message in a routing scheme that maintains complete routing tables.

IV. OverSim Models

OverSim [1] is designed as a modular simulation framework, with many common overlay features implemented as part of a generic base overlay class. OverSim provides message passing using RPC, and supports both iterative and recursive routing. Applications within OverSim are split into multiple tiers, allowing an application, such as the DHT TestApp, to sit on-top of another application, such as the DHT layer. These applications are implemented as modules and interface with overlays through the Key-Based Routing (KBR) API [16], which represents basic capabilities common to all structured overlays. The OverSim architecture is illustrated in Figure 2.

The main applications provided by OverSim are the KBR TestApp, and the DHT TestApp. The KBR TestApp is used to test the KBR services exposed by each overlay, such as routing look-up requests to specific keys. The DHT TestApp is used to test the DHT application, by performing DHT PUT and GET requests with randomly generated data items.

OverSim provides a number of different network models, for both structured and unstructured overlays. At the time of writing the current version of OverSim is OverSim-20101103, and contains models for Bamboo [5], Broose [6], Chord [7], GIA [8], Kademlia [9], Koorde [10], NICE [11], NTree [12], Pastry [13], Quon [14], and Vast [15].

At the lower layer OverSim provides multiple underlay models to allow simulation in more detail at an extra cost of performance, or abstraction for better performance but less detail in the underlying network. Using the simple model data packets are sent directly from one node to another by using a global routing table. The INET underlay model includes simulation models for all network layers. The single host underlay allows for simulation of a single node, connected to other OverSim instances over a real network.

A. *Chord Overlay Model*

The Chord model provided within OverSim is an implementation of the algorithms described in the original Chord paper [7], as well as various additions, described below.

1) *Aggressive join mode:* When aggressive join mode is enabled a node will update its predecessor pointer as soon as it receives a join request, and respond to the joining node with the address of its old predecessor. A message is then passed to the old predecessor to alert them of their new successor.

2) *Memorize failed successor:* This modification keeps note of when a successor node fails, and uses this to decide if we should accept the node provided in a stabilize response.

3) *Extended finger table:* The extended finger table modifies the finger table to include buckets of nodes at each finger position, instead of a single node at each position.

4) *Proximity routing:* If using the extended finger table modification, proximity routing will sort the contents of each finger bucket based on their observed latency. This prioritises lower latency nodes when there are multiple to choose.

5) *Merge optimizations:* The merge optimizations are a set of different optimizations which can be made. They include informing the original predecessor when we receive a new predecessor, informing a node who incorrectly believes they

are our predecessor of our real predecessor, and pinging fingers in our finger table to clear dead nodes.

Each of the additions described is optional, and can be toggled in the simulation configuration file. An overview of the simulation parameters used can be found in Table I.

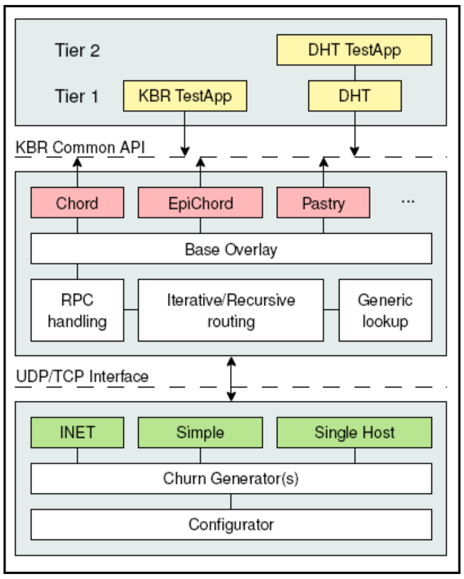


Figure 2: Modular architecture of OverSim.

TABLE 1: CHORD SIMULATION PARAMETERS.

Parameter	Scenario 1	Scenario 2
routing type	iterative	iterative
look-up interval	1m	-
network size	{8, 16 . . . 16,284}	1,000
number data items	-	10,000
replication rate	-	1
transition time	2m	15m
measurement time	2h	-
stabilize delay	20s	20s
fix fingers delay	120s	120s
check predecessor delay	5s	5s
successor list size	8	8
aggressive join	on	on
extended finger table	off	off
proximity routing	off	off
memorize failed successor	off	off
merge optimizations	off	off

B. *Pastry Overlay Model*

The Pastry model within OverSim is an implementation of the algorithms described in the original Pastry paper [13], with various optional components, described below.

1) Optimize look-ups: By default Pastry will try to find the closest destination node from its routing table only. The optimize look-ups modification searches the routing table, neighbourhood set, and leaf set to ensure that the returned node is definitely the closest that we know about.

2) Discovery algorithm: The discovery algorithm attempts to discover a reliable node with low latency to send the initial join request to, instead of using the bootstrap node. It works by recursively querying the closest known node for its routing table until no closer node is found.

3) Proximity neighbour selection: Using proximity neighbour selection, when nodes are merged in to the routing table they will replace existing entries if they have a lower latency than the existing entry.

4) Minimal join state: Instead of transferring the entire routing table, neighbourhood set, and leaf set on join, the neighbourhood set and leaf set are only transferred if required.

As with the Chord model, each of the additions described is optional, and can be toggled in the simulation configuration file. An overview of the simulation parameters used can be found in Table 2.

TABLE 2: PASTRY SIMULATION PARAMETERS.

Parameter	Scenario 1	Scenario 2
routing type	semi-recursive	semi-recursive
look-up interval	1m	-
network size	{1,000 . . . 10,000}	1,000
number data items	-	10,000
replication rate	-	1
transition time	2m	15m
measurement time	2h	-
bits per digit (b)	4	4
number of leaves	16	16
number of neighbours	32	32
optimize look-ups	off	off
discovery algorithm	off	off
proximity neighbours	on	on
partial join path	off	off
minimal join state	off	off

C. *Memory Requirements*

Observed memory requirements for a selection of different OverSim models are shown in Figure 3. As expected the memory usage of multi-hop networks such as Chord, Broose, or Kademlia grew close to linearly, as each node only maintains state for a logarithmic number of other nodes.

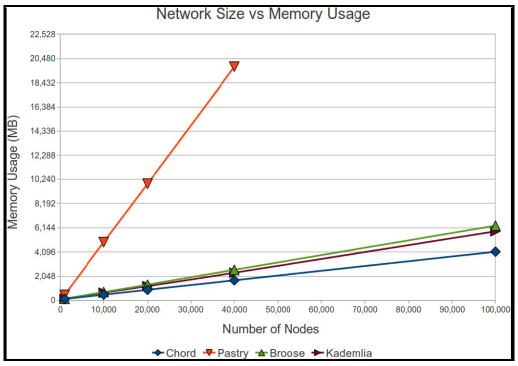


Figure 3: Memory required for simulating varying sized networks using OverSim models with default parameters.

In Figure 3 we observe that the memory usage of a Pastry network using the OverSim model does not seem to match that of other multi-hop networks, possibly implying that the Pastry model might have a memory leak.

V. Experimental Results

To evaluate the models we define 2 scenarios under which both models are tested.

In Scenario 1 the network is first populated with a set number of nodes. Once the network has reached the desired size it is given time to stabilize, and then look-ups are initiated from random nodes for random keys using OverSim's KBRTestApp, at an average rate of 1 look-up per node per minute. During this time both the result (success or fail) and hop count are recorded for each look-up. This scenario allows us to compare both the success rate and look-up performance of each model, validating the models routing algorithm.

In Scenario 2 the network is first populated with a set number of nodes, as in Scenario 1. Once the network has reached the desired size it is given time to stabilize, and then a set number of random data items are distributed within the DHT using OverSim's DHTTestApp and a trace file. We then kill a percentage of nodes within the network, and again give it time to stabilize. Finally we issue look-ups from random nodes for all the data items originally stored within the network, including those belonging to now dead nodes. This scenario allows us to test the models maintenance algorithms when faced with multiple node failures, validating the models maintenance algorithm.

Each scenario was repeated 10 times, with results averaged.

A. Chord Results

In Scenario 1 the Chord network completed 100% of lookups successfully, as to be expected when the network is stable. The average path length for each measured network size was a close match to that reported in the original paper, as shown in Figure 4. The distribution of look-up path lengths in a 4,096 node network (as chosen in the original paper) is shown in Figure 5, again we can see this is a close match.

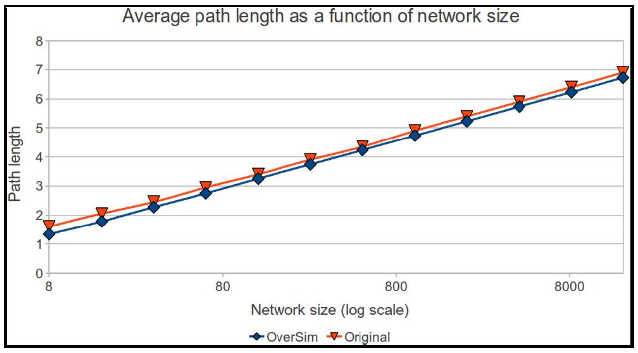


Figure 4: Chord: The average path length as a function of network size.

The results for Chord from Scenario 2, showing look-up failures versus node failures, are shown in Figure 6. These results show that the maintenance algorithm is able to correctly repair routing tables, and the only data lost is that which was stored on the nodes which failed.

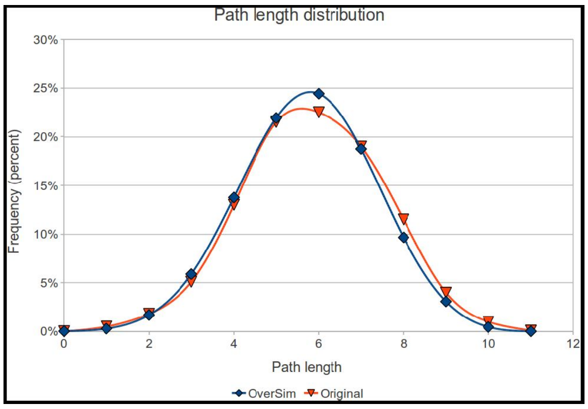


Figure 5: Chord: Distribution of the path length in the case of a 2^{12} node network.

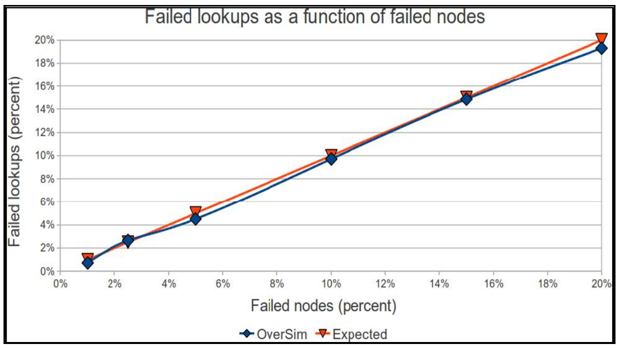


Figure 6: Chord: Look-up failures as a function of node failures.

B. Pastry Results

Due to memory constraints, discussed in Section IV-C, we were unable to simulate a Pastry network larger than 40,000 nodes. In the original paper, results were presented for network sizes ranging from 1,000 to 100,000 nodes. We have limited our comparisons to networks ranging from 1,000 to 10,000 nodes.

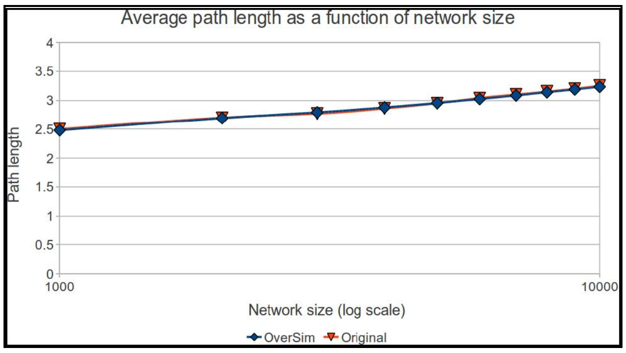


Figure 7: Pastry: The average path length as a function of network size.

In Scenario 1 the look-up success rate for Pastry was also 100%, as expected. The average look-up path length is shown in Figure 7. Again they are a close match to the results in the original paper, and in fact keep in line with the theoretical maximum path length of log2b (N). Due to memory constraints we could not obtain path length distribution for a 100,000 node network to compare with the original paper, however the path length distribution for a 10,000 node network using the OverSim model, and for a 100,000 node network from the original paper can be seen in Figure 8. In both cases the shape of the distribution is a close match, however offset due to the differing network size.

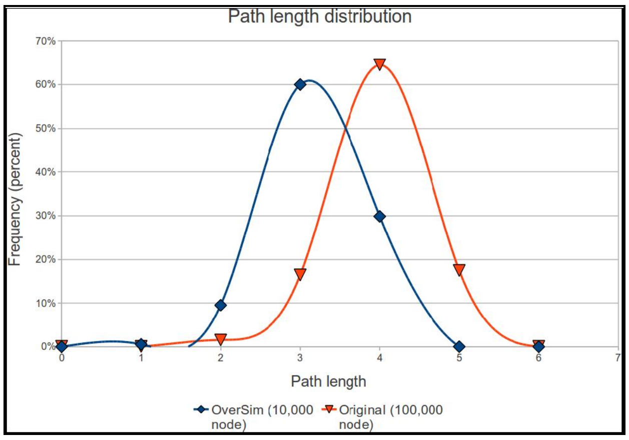


Figure 8: Pastry: Distribution of the path length in the case of a 10,000 node OverSim network and 100,000 node network from original paper.

In Scenario 2 the maintenance algorithm is able to, again, correctly repair routing tables, and the only data lost is that which was stored on the nodes which failed - shown in Figure 9.

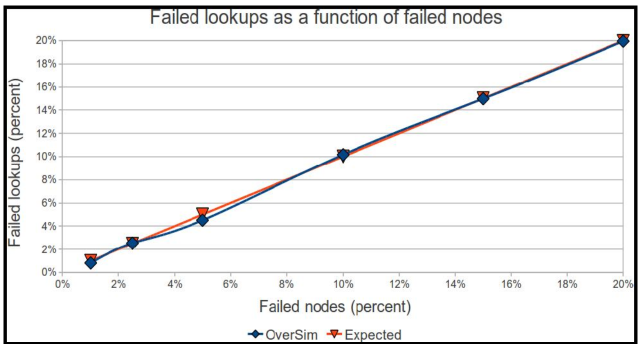


Figure 9: Pastry: Look-up failures as a function of node failures.

VI. Conclusions

In this paper we have given an overview of both the Chord and Pastry DHT algorithms, as well as the OverSim P2P simulation framework. We have described both the Chord and Pastry models provided with OverSim, and compared results obtained using these models against those provided in the original papers as a form of validation. Our results have shown than both the Chord and Pastry models behave as expected, and produce results which closely match the original papers. This allows us to conclude that both models are valid representations of their respective algorithms, and that any data obtained from work building on-top of these models is realistic.

References

[1] I. Baumgart, B. Heep, and S. Krause, “OverSim: A Flexible Overlay Network Simulation Framework,” in Proceedings of 10th IEEE Global Internet Symposium (GI ’07) in conjunction with IEEE INFOCOM 2007. Anchorage, AK, USA: IEEE, May 2007, pp. 79–84.

[2] T. M. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling, “p2psim: a simulator for peer-to-peer (p2p) protocols.” [Online]. Available: http://pdos.csail.mit.edu/p2psim/

[3] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, “The Peersim Simulator.” [Online]. Available: http://peersim.sf.net

[4] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo, “PlanetSim: A New Overlay Network Simulation Framework,” Software Engineering and Middleware, pp. 123–136, 2005.

[5] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, “Handling Churn in a DHT,” in Proceedings of the annual conference on USENIX Annual Technical Conference, 2004, p. 10.

[6] L. Viennot, “Broose: a practical distributed hashtable based on the de-Bruijn topology,” in Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings. Ieee, 2004, pp. 167–174.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” in SIGCOMM ’01. ACM Press, 2001, pp. 149–160.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker,“Making gnutella-like P2P systems scalable,” in SIGCOMM ’03. USA: ACM Press, 2003, pp. 407–418.

[9] P. Maymounkov and D. Mazières, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric,” Peer-to-Peer Systems, vol. 2429, pp. 53–65, 2002.

[10] M. F. Kaashoek and D. Karger, “Koorde: A Simple Degree-Optimal Distributed Hash Table,” Peer-to-Peer Systems II, vol. 2735, pp. 98–107, 2003.

[11] B. Bhattacharjee, S. Lee, R. Morselli, R. Sherwood, D. Levin, and Suman Banerjee, “NICE.” [Online]. Available: http://www.cs.umd.edu/projects/nice/

[12] C. GauthierDickey, V. Lo, and D. Zappala, “Using n-trees for scalable event ordering in peer-to-peer games,” in NOSSDAV ’05. ACM Press, 2005, pp. 87–92.

[13] A. Rowstron and P. Druschel, “Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems,” in IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001, pp. 329–350.

[14] H. Backhaus and S. Krause, “QuON - a Quad-Tree Based Overlay Protocol for Distributed Virtual Worlds,” International Journal of Advanced Media and Communication, vol. 4, no. 2, pp. 126–139, 2010.

[15] S.-C. Chang, T.-H. Chen, J.-S. Chiou, Y.-L. Huang, S.-Y. Hu, and G.-M. Liao, “VAST.” [Online]. Available: http://vast.sourceforge.net/

[16] F. Dabek, B. Zhao, P. Druschel, and J. Kubiatowicz, “Towards a Common API for Structured Peer-to-Peer Overlays,” Peer-to-Peer Systems II, vol. 2735, pp. 33–44, 2003.

An Evaluation of EpiChord in OverSim

Jamie Furness, Farida Chowdhury, Mario Kolberg

Computing Science and Mathematics,
Universty of Stirling, Stirling, Scotland
{jrf,fch,mko}@cs.stir.ac.uk

Abstract. EpiChord is a Distributed Hash Table (DHT) algorithm which supports data storage/retrieval in large scale distributed systems. It removes the typical *O(logn)*-state-per-node restriction imposed by the majority of other DHT topologies by employing a reactive routing state maintenance strategy that amortizes network maintenance costs into lookup queries. Under ideal condition, EpiChord's lookup performance can approach O(1) hops - with maintenance costs comparable to traditional multi-hop DHTs. This paper presents an implementation of EpiChord in OverSim, and validates the performance of our model against the performance reported in the original EpiChord paper. We also present some adjustments to the algorithm to remove a discrepancy and then compare our modified results with the original ones. Finally, we present additional results showing the EpiChord algorithm is stable over time and performs well for larger networks.

1 Introduction

Distributed Hash Tables (DHTs) [2] supported Peer-to-Peer (P2P) applications are an ideal substrate for building large scale distributed systems because they are self-organizing, adaptable and scalable and offer efficient routing between nodes within a bounded number of hops. EpiChord [1] is a DHT lookup algorithm which demonstrates that node state restrictions can be relaxed which were imposed by the majority of other DHT algorithms by using a reactive routing state maintenance strategy. Nodes piggyback additional network information on lookup queries to keep their routing state up-to-date. This makes EpiChord ideally suited to large scale environments. This paper discusses an implementation [24] of EpiChord within the OverSim Simulator [3]. The model is validated against the original EpiChord paper. Specifically, the contributions of the paper are as follows:

- An independent evaluation of EpiChord, by comparing results from our simulation model to the results presented in the original EpiChord paper.
- Performance evaluation in multiple scenarios, defined in the original paper, which test both the routing and maintenance algorithms of the model.
- Amendments to the original model together with a comparison of the results obtained from our model against the corrected results from the original model.
- Performance evaluation of EpiChord in larger networks and for longer simulations.
- A freely available EpiChord model in OverSim.
- A review of available simulators.

The original implementation of EpiChord was a model for the SSFNet simulation framework [4] which is not publicly available. The authors are not aware of other EpiChord models which are publicly available. In other work [5] we have validated the models for both Chord and Pastry in OverSim.

The remainder of the paper is structured as follows: Section 2 discusses related work, Section 3 provides an overview of the EpiChord DHT algorithm, Section 4 compares network simulators, Section 5 discusses implementation details of the EpiChord model in OverSim, Section 6 presents an evaluation of results after changes to the original EpiChord model. Section 7 presents validating results from our EpiChord model as well as results demonstrating EpiChord's scalability. Section 8 concludes this paper.

2 Related Work

A large number of multi-hop structured Peer-to-Peer (P2P) algorithms have been proposed [2]. These algorithms are characterized by O(log N) hop count. Because each overlay hop translates to potentially many hops in the underlying network, multi-hop overlays have a relatively poor latency characteristic for connecting large numbers of peers. Consequently, systems have been developed to trade-off latency for larger routing tables. However these designs lead to increased network traffic for managing the larger routing tables. Thus efficient overlay maintenance in O(1)-hop (one-hop) overlays is an important research question. Two techniques have emerged [2] for maintaining routing tables in overlays: active stabilization where peers have fixed communication to maintain a target routing table accuracy, and opportunistic updating where routing table maintenance depends on lookup load and available bandwidth.

An example active stabilization algorithm is EDRA (Event Detection and Reporting Algorithm) used in the D1HT one-hop overlay [8]. EDRA has been proposed to give reasonable message rate for high levels of routing table accuracy. For example, D1HT has up to an order magnitude lower maintenance bandwidth usage compared to the OneHop [10], another active stabilization one-hop overlay. EDRA* [11] offers some improvements over EDRA. Examples of opportunistic overlay maintenance include EpiChord [1] (used in this paper) and Accordion [9].

Kelips [7] is a O(1)-hop overlay which uses an epidemic multicast protocol for exchanging overlay membership and other soft state between nodes. Such a protocol consists of two sub-protocols: a multicast data dissemination protocol and a gossip protocol to exchange message history for reliability purposes.

Accordion [9] is a variable hop overlay, in which a peer limits its routing table update message level based on its available bandwidth. During periods of low bandwidth, routing table accuracy can approach that of multi-hop overlays while for higher bandwidth, routing table accuracy reaches one-hop. Accordion uses recursive parallel lookups so as to maintain fresh routing table entries in its neighborhood of the overlay and reduce the probability of timeout. Note that recursive parallel lookups create more load on the target peer compared to iterative parallel lookups.

3 EpiChord Background

EpiChord [1] is a DHT algorithm which can achieve one-hop lookup performance under lookup intensive workloads, and at worst case $O(log_2(N))$ hop, as offered in many multi-hop networks. As the name suggests, EpiChord is based on the Chord DHT [6]. Like Chord, EpiChord is organized in a one-dimensional circular address space where each node is assigned a unique node identifier. The node responsible for a key is the node whose identifier most closely follows the key. In addition to maintaining a list of *k* succeeding nodes, EpiChord also maintains a list of the *k* preceding nodes. Instead of maintaining a finger table, as in Chord, EpiChord maintains a cache of nodes. Nodes update their cache by observing lookup traffic, and add an entry anytime they learn of a node not already in the cache. Nodes in the cache each have a timeout, resulting in stale nodes being removed.

In general terms EpiChord can be thought of as Chord with a cache of extra node addresses. As such the routing algorithm is similar to that in Chord. With a well populated cache this results in lookup performance of one hop. Under high churn the performance drops to that of Chord, $O(log_2(N))$ hops in the worst case.

3.1 Lookup Algorithm

EpiChord uses an iterative lookup algorithm, as it avoids sending redundant queries when using parallel requests. It also allows the querying node to receive all information related to the query path, and hence updates its cache with new entries. To lookup a data item with the key *id*, a node will initiate p queries in parallel - to the node immediate succeeding *id* and to the p-1 nodes preceding *id*. When queried, a node will respond as follows (l and p are both system parameters):

- If it owns *id*, it will return the value associated with *id*, and information on its predecessor and successor.
- If it is a predecessor of *id* relative to the querying node, it will provide information about its successor and the l best next hops towards the destination.
- If it is a successor of *id* relative to the querying node, it will provide information on its predecessor and the l best next hops towards the destination.

When a reply is received, further queries are dispatched in parallel if the querying node learns about any node closer to the target *id* than the best successor and predecessor nodes that have already responded.

3.2 Cache Invariant

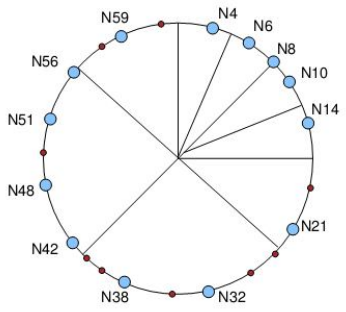


Fig. 1. Example of slicing of address space for N8.

To guarantee worst case lookup performance of $O(log_2(N))$ each node divides the address space into two sets of exponentially smaller slices, as seen in Fig. 1. Each node maintains their cache such that every slice contains at least $\frac{j}{1-\gamma}$ cache entries at all times, where j is a network parameter and γ is a local estimate of the probability that a cache entry is out-of-date. Nodes periodically check their cache slices to ensure that there are sufficient unexpired cache entries. To calculate γ, each node keeps track of n_p, the number of messages sent, and n_t, the number of messages which timed out. γ is calculated using n_t / n_p. In addition, n_p and n_t are periodically (when the cache is flushed) multiplied by a network parameter δ to obtain exponentially weighted moving averages.

3.3 Routing Table Updates

Each node periodically probes their immediate neighbours to ensure that they are still alive. The delay between these stabilization attempts is calculated based on the observed lifetime of nodes in the finger cache. For this reason the finger cache also contains a map of dead nodes, and the observed lifetime is calculated by taking the time between first learning of the node (s_{start}) until learning of its death (s_{end}). The observed lifetime for each dead node is averaged, and the obtained estimate is then multiplied by the lifetime estimate multiplier, ω, to calculate when the next stabilization attempt should be scheduled.

$$s = \frac{\sum s_{end} - s_{start}}{n} \cdot \omega \qquad \textbf{(1)}$$

In case where the sample size, n, is less than 5, the stabilization interval is simply set to the network parameter s.

With active propagation, nodes will inform their neighbours of any detected changes in the successor or predecessor lists as soon as they happen, rather than waiting for the next stabilization attempt. This increases the maintenance bandwidth when under high churn, however also results in more accurate successor and predecessor lists, and hence fewer false-negatives.

If a node has an outdated view of the local key space that they are responsible for, they may fail to respond correctly to all queries. By including their believed predecessor and successor in the query response, the querying node can either make a step towards the destination or, if the believed predecessor does not respond, determine that the responsible node is dead. This false-negative detection allows the querying node to resolve the lookup correctly. If a false-negative is detected, the querying node will immediately inform the new responsible node that their predecessor has failed and now they should be responsible for the requested key.

4 Review of Simulators

Before deciding on OverSim, a detailed review of other available and active P2P network simulators was carried out. A summary of these tools is provided in Table 1.

PeerSim [6] is written in Java. Its main focus is to provide high scalability and can handle a network of up to 10^6 nodes. However, this scalability comes at the cost of not including a model of the behavior of the underlying communication network, e.g. TCP/IP stack and latencies. P2PSim [13] is a discrete event simulator for P2P overlays written in C++. It supports Chord, Accordion, Koorde, Kelips, Tapestry, and Kademlia. However, these implementations are specific to P2PSim and do not model all features of the protocols. P2PSim has been simulated with up to 3,000 nodes using the Chord implementation. This simulator is largely undocumented and therefore hard to extend.

Overlay Weaver [14] is a toolkit for P2P Overlays written in Java. It has been tested with tens of thousands of nodes (their website quotes 300,000). Chord, Kademlia, Pastry, Tapestry and Koorde are available. The simulations have to be run in real-time environments and there is no statistical output which makes its use very limited. PlanetSim [14] is a discrete event simulation framework for both structured and unstructured overlays, written in Java. It has a modular, well-structured architecture and services can be re-used for other overlays. Chord and Symphony models exist and can consist of up to 100,000 nodes. However, it provides rather limited support to collect statistics. It also has a very simplified underlying network layer without any consideration of bandwidth and latency costs.

Table 1. A comparison of available active P2P simulators.

Simulator	P2P Protocols	Network size	Language
PeerSim	Collection of internally developed P2P models	$>10^6$	Java
P2PSim	Chord, Accordian, Koorde, Kelips, Tapestry, Kademlia	3000	C++
Overlay Weaver	Chord, Kademlia, Koorde, Pastry, Tapestry and FRT-Chord	Tens of thousand	Java
PlanetSim	Chord, Symphony	100,000	Java
NS2	Gnutella	N/A	C++/OTcl
SSFNet	Chord, EpiChord	33,000	Java/C++/DML
OverSim	Chord, Kademlia, Pastry, Bamboo, Broose, Gia	100,000	C++
PeerfactSim.Kom	CAN, Chord, Kademlia, Gia, C-DHT, Gnutella 0.4/0.6, Pastry	50,000	Java
D-P2P-Sim+	Chord	400,000	Java

NS2 [16] is a discrete-event simulator that provides substantial support for simulation of lower layer protocols. Only one P2P protocol, Gnutella, is available in NS2. Simulations in NS2 are constructed using C++ and OTcl. It is mostly used for small networks and is generally unsuitable for large scale P2P overlay networks.

SSFNet [4] is a discrete-event simulation framework written in Java and C++. This framework is built on the Scalable Simulation Framework (SSF) and uses the Domain Modelling Language (DML) to configure networks. Chord and EpiChord have been implemented in SSFNet. There is a claim that SSFNet manages to run models with 33,000 nodes, however, the authors of the original EpiChord paper [1] and ourselves could not simulate networks with more than 10k nodes.

OverSim [1] is an open-source P2P simulation framework for the OMNeT++ simulation environment. It provides a generic lookup mechanism and an RPC interface to facilitate additional protocol implementations. It allows large-scale simulations of simplified networks as well as complex heterogeneous underlay networks. Several P2P algorithms such as Chord, Kademlia, Bamboo, Broose, Koorde, NICE, NTree, Pastry, and GIA have been implemented in OverSim. Models can scale to over 100,000 nodes. More comprehensive surveys of P2P network simulators can be found in [19,20].

PeerfactSim.Kom [17] is a discrete event based P2P simulator environment. Its focus is on being extendable and on large scale network models. This simulator offers the potential to model different types of peer-to-peer systems including distributed CDNs, streaming applications and overlay systems. It comes with a built-in churn generator. The simulator includes models of lower layers but does not yet include TCP.

D-P2P-Sim+[18] is a distributed simulation environment which employs multi-threading, asynchronous message passing and distributed environment with graphical user interface. There is little information on this simulator besides a short paper and poster. These report simulated network sizes of up to 400,000 nodes. It seems the only implemented overlay algorithm is Chord. However, the system is extendible and other algorithms could be implemented. Multiple computers running the simulator may be interconnected to achieve larger simulated network sizes.

Based on this study OverSim was selected for our experimentation due to its flexibility with respect to underlay characteristics and possible high scalability.

5 Oversim Implementation

OverSim [3] is designed as a modular simulation framework, with many common overlay features implemented as part of a generic base overlay class. OverSim provides message passing using Remote Procedure Calls (RPC), and supports both iterative and recursive routing. Applications within OverSim are split into multiple tiers, allowing an application to sit on-top of another application. These applications are implemented as modules and interface with overlays through the Key-Based Routing (KBR) API [21], which represents basic capabilities common to all structured overlays. As mentioned above, OverSim provides a number of different network models, for both structured and unstructured overlays. The OverSim architecture is illustrated in Fig. 2.

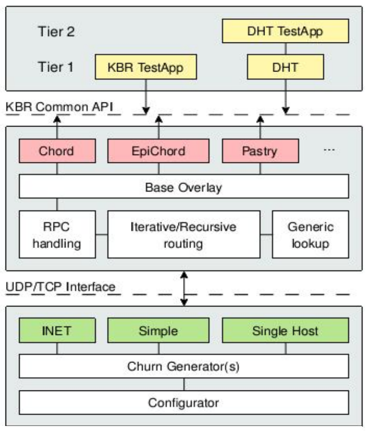


Fig. 2. Oversim architecture

At the lower layer OverSim provides multiple underlay models to allow for inclusion of specific underlay characteristics in the simulation (at a cost of scalability), or underlay abstraction for increased scalability. Using the simple model, data packets are sent directly from one node to another by using a global routing table. The INET underlay model includes simulation models for all network layers. The single host underlay allows for simulation of a single node, connected to other OverSim instances over a real network.

Below we discuss some alterations which we made to the original EpiChord protocol when implementing it as an OverSim module.

5.1 Node Join Protocol

In the original EpiChord algorithm, upon receipt of a join request a node will instantly update their predecessor list and finger cache to include the joining node. In our implementation we found this was occasionally causing messages to be routed to nodes who are still in the process of joining, and not yet ready to correctly handle requests. To solve this issue we implemented a three-way handshake. In our implementation the joining node will send a final acknowledgment when they are ready to handle requests, indicating they can now be safely added as a predecessor.

5.2 Lookup Algorithm

The OverSim framework provides modules for iterative and recursive routing, as can be seen in Fig. 2, with support for parallelism. While this makes implementation of many overlays easier and reduces duplicated code, only certain parts of the module can be easily overridden. This was a problem for EpiChord, primarily due to the non-linear order in which nodes are to be queried, and EpiChord's ability to check for false negative responses. To implement these features we had to make changes to the iterative routing module, allowing us to override additional parts of the module with code specific to EpiChord.

6 Results - Changes to the Original Model

6.1 Application layer Lookups

In the original EpiChord model all lookup types (JOIN, MAINTENANCE, and APPLICATION) are included when calculating results. The KBRTestApp in OverSim only includes lookups it has initiated (APPLICATION) in the results. We feel this is actually a more useful metric for anyone wishing to build on-top of EpiChord, so we instead recalculated the results from the original model using only APPLICATION lookups. A comparison of the average path lengths can be seen in Fig. 3; the other metrics remained unchanged.

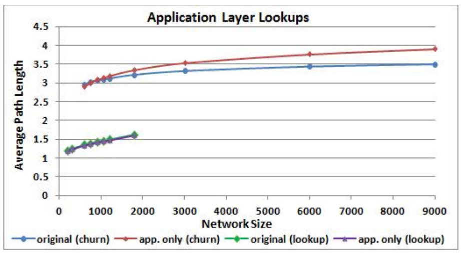


Fig. 3. Comparison of average path length with APPLICATION lookups only vs. all lookup.

In [22], authors proposed two generic classes of workloads: *lookup intensive* and *churn intensive*. These metrics were adopted by the EpiChord authors for experimentation. For the purposes of validating our model, we also adopt these two metrics. In the *lookup intensive* workload, node lifetimes are exponentially distributed with a mean of 10 minutes, and each node performs lookups on average every 0.5 seconds. In this scenario the background maintenance traffic is negligible compared to the active lookup rate. In the *churn intensive* workload, node lifetimes are again exponentially distributed with a mean of 10 minutes, however this time each node only performs lookups on average every 100 seconds. In this scenario the lookup rate is so low, most of the lookups captured are lookups arising from node joins and cache maintenance.

Fig 3 shows the average path length remains unchanged for the *lookup intensive* workload. This is to be expected, as the lookup intensive workload is dominated by APPLICATION lookups. In the churn intensive workload we see a rise in average hop count as the network size increases; this is because the result was originally dominated by JOIN and MAINTENANCE lookups, which tend to be for closer keys.

6.2 Fixing p

In the source of the original model we encountered a minor mistake[1], which, in many cases, resulted in *p+1* parallel requests being generated - rather than the supposed maximum of *p*. Results comparing the average path lengths and success rates when *p=1* can be seen in Fig. 4.

From these results we observe a rise in average path length, and a small decline in lookup success rate, for both workloads. We also observe a drop in the size of nodes cache tables, which increases with the network size. This is to be expected, as fewer queries are dispatched and hence fewer new nodes are discovered.

[1]When receiving a timeout or negative response, further queries are dispatched while *pending* <= p_{max}, resulting in $p_{max}+1$ pending queries.

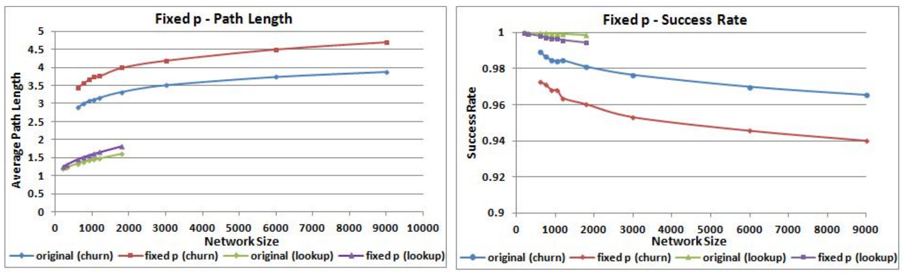


Fig. 4. Average path length and success rate with fixed p.

7 OverSim Results

To match the original scenarios, lookups were performed throughout the entire simulation, with measurements taken from the very beginning. OverSim, by default, only starts performing lookups and recording measurements once the network has reached the desired size, however this is configurable in the settings.

An overview of the simulation parameters can be found in Table 2. When we refer to results from the original model, we refer to the results generated after taking the changes in Section 6 into account. All results are averages of 5 simulation runs.

Table 2. OverSim simulation parameters.

Description	Lookup Intensive	Churn Intensive
Lookup Interval	0.5s	100s
Network Size	{600,…..,2000}	{600,…..,2000}
Lifetime Mean	600s	600s
Stabilize Delay	60s	60s
Cache TTL	120s	120s
Cache Flush Delay	20s	20s
Cache Check Multiplier	3	3
Measurement Time	3000s	3000s
Neighbour list size	4	4
Redundant nodes, *l*	3	3
Parallelism, *p*	1,3,5	1,3,5
Required nodes/slice, *j*	2	2
Lifetime multiplier, ω	0.5	0.5
Slice multiplier, δ	0.5	0.5

7.1 Finger Cache State

During simulation we measure the average finger cache size for each node, as well as the average accuracy of each node's finger cache. The accuracy is a measure of how many nodes in the finger cache are actually still active within the network.

We observe an average finger cache accuracy of 87% across all network sizes and both scenarios - almost identical to that of 87.5% reported in the original paper.

As expected the finger cache size observed in the *lookup intensive* workload is much larger than that in the *churn intensive* workload, due to the extra node information received within lookup messages. The observed finger cache size for varying network sizes under a lookup intensive workload and churn intensive workload can be seen in Fig. 5.

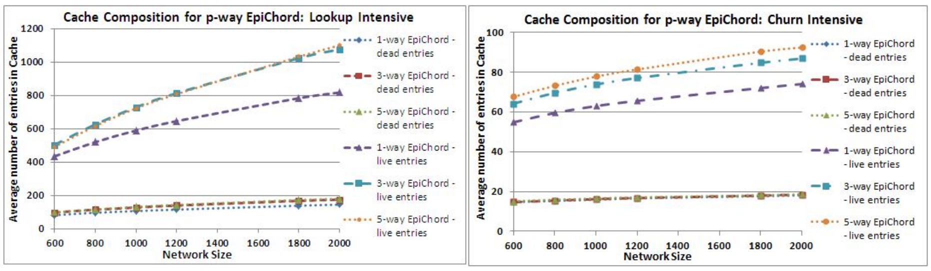


Fig. 5. Cache composition for p-way EpiChord under lookup intensive and churn intensive workload.

7.2 Lookup Success Rate

Every lookup performed can be classified into one of four categories:

- Success: The node responsible for the requested key responds positively.
- Failure: No positive response received and no more viable candidates, or reached the maximum hop/time limit
- False-positive: A node has responded positively but is not responsible for the requested key.
- False-negative: A node did not respond positively but should be responsible for the requested key.

By using false-negative detection, described in Section 3.3, nodes can detect and handle false-negatives; ultimately they are treated as successful lookups.

The observed success rate for both lookup intensive and churn intensive workloads is shown in Fig. 6. Here we use column diagram to show the success rate for *p-way* parallel queries (*p=1,3,5*) for different network sizes up to 2000 nodes.

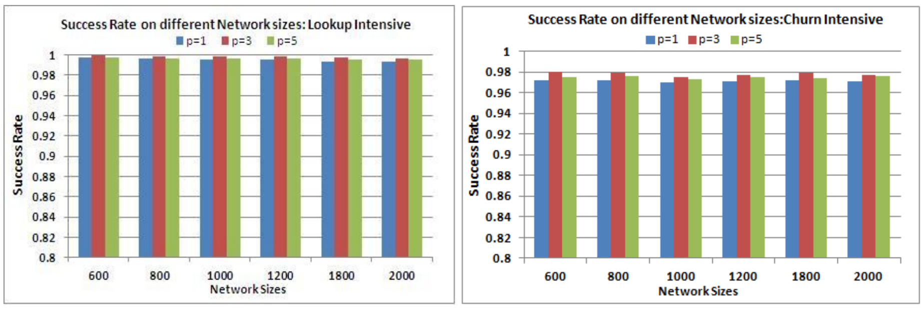


Fig. 6. Comparison of Success Rate for p-way EpiChord for varying network sizes under lookup intensive and churn intensive workload.

As shown in Fig. 6, the lookup success rate is marginally higher for lookup intensive workload than for the churn intensive workload. This is expected as under the lookup intensive workload, the larger number of lookups helps to keep the routing state up-to-date whereas for the churn intensive workload, the information propagation rate is lower. Increasing parallelism has only a very slight effect on the success rate. It appears that the lookup improvement is not worth the extra cost of the parallel lookups. The success rates for *p=5* is marginally lower than for *p=3*. This rather counter intuitive behavior has also been observed in the original paper and is due to the *5-way* network generating fewer cache-refreshing lookups than a *3-way* EpiChord network.

7.3 Lookup Path Length

For each successful lookup performed we also measure the path length - the number of hops taken to find the final destination. Fig. 7 shows the observed path length for both lookup intensive and churn intensive workloads. We observe that in the lookup intensive workload, the hop count varies from 1.1 to 1.4 in both *3-way* and *5-way* EpiChord networks, which signifies that each node has almost complete routing table information and thus allows passing messages nearly in one hop. On the other hand, the hop count varies from 2.8 to 3 under churn intensive workloads with fewer lookups which also satisfies the *O(log n)*-hop lookup performance as depicted in the original paper. Again, the results suggest that an increased level of parallelism in the lookups only marginally improves the hop count, whereas the increased number of lookups issued in the lookup intensive workload has a much more pronounced positive effect.

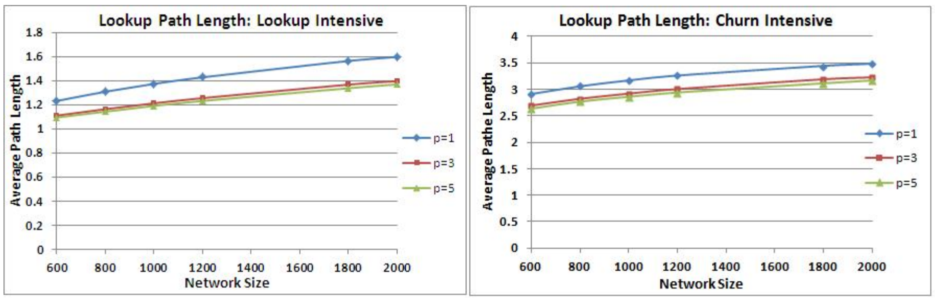


Fig. 7. Comparison of Lookup Path Length for p-way EpiChord for varying network sizes under Churn Intensive and Lookup Intensive workload.

7.4 Stability and Scalability

We measured the stability of the EpiChord model in OverSim. Fig. 8(a) shows the measurement phase vs. success ratio graph for *p=1, 3, 5.* In OverSim, during the measurement phase, the statistics are collected. Our model has been tested up to 100,000s and demonstrates that the model is stable after an initial period of between 10000s (for p=1) and 20000 (for p=3,5). EpiChord also has been tested for scalability in terms of network size for scenarios with 5,000 - 20,000 nodes. Fig. 8(b) shows the results for lookup success ratio for different network sizes. This set of results means that the network size does not affect the success rate of EpiChord. As before, p only improves the performance in a rather minor way.

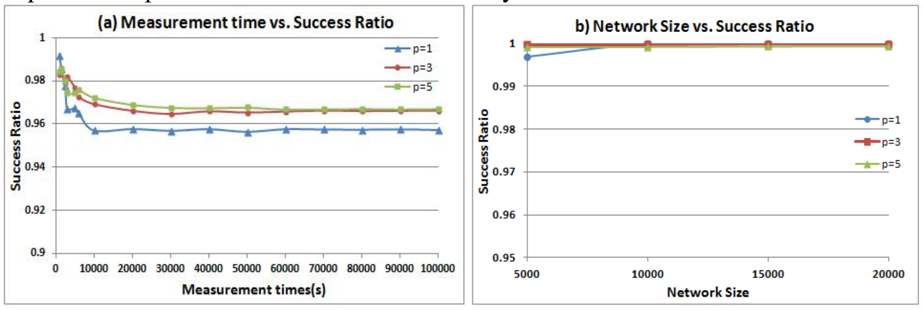


Fig. 8. a) Success Ratio of EpiChord for varying measurement times for *p=1,3,5* demonstrating the stability of the model; b) Average Success Ratio of EpiChord for networks with 5,000 to 20,000 nodes.

8 Conclusion

This paper presented our OverSim EpiChord model, and validated it by comparing our results against the performance of the original EpiChord model. The results for our model closely match those from the original model, supporting the claim that our model is a valid implementation of the EpiChord algorithm. We have then presented amendments to the model and investigated the effects on the performance of the model. Furthermore we have shown that EpiChord and our model in OverSim is stable over an extended period of time. We have also demonstrated that EpiChord achieves excellent results for larger networks. EpiChord's performance is strongly influenced by the number of lookups issued by the nodes as routing table information is attached to lookup return messages. Thus an increased number of lookup message improve the performance of the network, whereas an increased level of parallelism only marginally improves performance. Due to its excellent lookup performance for large scale networks, EpiChord appears well suited to support large distributed environments.

Separately, we have used this model to simulate the effect of different lookup traffic setups, and high node churn to investigate EpiChord's suitability for use in mobile networks [23].

References

1. B. Leong, B. Liskov, and E. D. Demaine. EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management. Computer Communications, Elsevier Science, Vol. 29, pp. 1243-1259.
2. K. Dhara, Y. Guo, M. Kolberg, X. Wu, Overview of Structured Peer-to-Peer Overlay Algorithms, Handbook of Peer-to-peer Networking, Springer, 2009.
3. I. Baumgart, B. Heep, S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. 10th IEEE Global Internet Symposium (GI '07), May 2007.
4. The SSFNet project. Accessed 01-August-2012. [Online]. Available:http://www.ssfnet.org/
5. J. Furness, M. Kolberg, Considering complex search techniques in DHTs under churn, in: 2011 IEEE Consumer Communications and Networking Conference (CCNC), IEEE, 2011.
6. I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. Conf on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01), 2001. ACM.
7. I. Gupta, K. Birman, P. Linga, A. Demers, R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS '03), 2003.
8. L. Monnerat, C. Amorim. D1HT: A Distributed One Hop Hash Table. 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS), April 2006.
9. J. Li, J. Stribling, R. Morris, M. F. Kaashoek. Bandwidth-efficient management of DHT routing tables. Symposium on Networked System Design and Implementation (NSDI) 2005.
10. A. Gupta, B. Liskov, R. Rodrigues. Efficient routing for peer-to-peer overlays. 1st Symposium on Networked Systems Design and Implementation (NSDI), 2004.
11. J. Buford, A. Brown, M. Kolberg. Analysis of an Active Maintenance Algorithm for an O(1)-Hop Overlay. IEEE Globecom 2007.
12. PeerSim P2P Simulator. Accessed 05-Jan-2013. http://peersim.sourceforge.net.
13. P2Psim: A Simulator for Peer-to-Peer (P2P) Protocols. http://pdos.csail.mit.edu/p2psim/
14. K. Shudo, Y. Tanaka, S. Sekiguchi. Overlay Weaver: An Overlay Construction Toolkit, Computer Communications,Vol.31, Issue2, pp. 402-412 (2007).
15. PlanetSim: An Overlay Network Simulation Framework. http://planet.urv.es/planetsim
16. The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/

17. D. Stingl, C. Groß, J. Rückert, L. Nobach, S. Kovacevic, R. Steinmetz. PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems, Intl. Conf. on High Performance Computing & Simulation (HPCS), 2011.
18. S. Sioutas, K. Tsichlas, G. Papaloukopoulos, Y. Manolopoulos, E. Sakkopoulos. A novel Distributed P2P Simulator Architecture: D-P2P-Sim. ACM Intl. Conf. on Information and Knowledge Management (CIKM), Hong Kong, 2009
19. A. Brown and M. Kolberg. Tools for peer-to-peer network simulation. Internet-Draft Version 00, IETF, January 2006.
20. S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A Survey of Peer-to-Peer Network Simulators. In The Seventh Annual Postgraduate Symposium, Liverpool, UK, 2006.
21. F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz. Towards a Common API for Structured Peer-to-Peer Overlays. Peer-to-Peer Systems II, 2735:33–44, 2003.
22. J. Li, J. Stribling, F. Kaashoek, R. Morris, and T. Gil. A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs under Churn. In INFOCOM, 2005.
23. F.Chowdhury, M.Kolberg. An Investigation of EpiChord with high Node Churn. Submitted.
24. J. Furness, F. Chowdhury, M. Kolberg. EpiChord model for OverSim. http://www.cs.stir.ac.uk/~fch/EpiChord_Model/

BIBLIOGRAPHY

[1] Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi. DKS(N, k, f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 344–350. IEEE, 2003. ISBN 0-7695-1919-9. doi: 10.1109/CCGRID.2003.1199386. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1199386.

[2] Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. *Proceedings. Second International Conference on Peer-to-Peer Computing,*, pages 33–40, 2002. doi: 10.1109/PTP.2002.1046310. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1046310.

[3] James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4):37–es, 2007. ISSN 15496325. doi: 10.1145/1290672.1290674. URL http://portal.acm.org/citation.cfm?doid=1290672.1290674.

[4] Helge Backhaus and Stephan Krause. QuON - a Quad-Tree Based Overlay Protocol for Distributed Virtual Worlds. *International Journal of Advanced Media and Communication*, 4(2): 126–139, 2010.

[5] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. *Proceedings of the First International Conference on Pervasive Computing*, pages 195 – 210, 2002.

[6] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007*, pages 79–84, Anchorage, AK, USA, May 2007. IEEE. ISBN 978-1-4244-1697-4. doi: 10.1109/GI.2007.4301435. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4301435.

[7] Bobby Bhattacharjee, Seungjoon Lee, Ruggero Morselli, Rob Sherwood, Dave Levin, and Suman Banerjee. NICE. URL http://www.cs.umd.edu/projects/nice/.

[8] Alan Brown, Mario Kolberg, and John Buford. Chameleon: An Adaptable 2-Tier Variable Hop Overlay. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–6, Las Vegas, NV, USA, 2009. IEEE.

[9] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1510–1520. IEEE, 2003. ISBN 0-7803-7752-4. doi: 10.1109/INFCOM.2003.1208986. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1208986`.

[10] Shao-Chen Chang, Tsu-Han Chen, Jiun-Shiang Chiou, Yu-Li Huang, Shun-Yun Hu, and Guan-Ming Liao. VAST. URL `http://vast.sourceforge.net/`.

[11] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like P2P systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '03*, pages 407–418, New York, New York, USA, 2003. ACM Press. ISBN 1581137354. doi: 10.1145/863997.864000. URL `http://portal.acm.org/citation.cfm?doid=863955.864000`.

[12] Adina Crainiceanu, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke, and Jayavel Shanmugasundaram. P-Ring : An Index Structure for Peer-to-Peer Systems. Technical report, In Cornell Technical Report, 2004.

[13] Frank Dabek, Ben Zhao, Peter Druschel, and John Kubiatowicz. Towards a Common API for Structured Peer-to-Peer Overlays. *Peer-to-Peer Systems II*, 2735:33–44, 2003. doi: 10.1007/b11823. URL `http://www.springerlink.com/content/r9p1gmyek16kfklq/`.

[14] Sameh El-ansary, Luc Onana Alima, Per Brand, and Seif Haridi. Efficient Broadcast in Structured P2P Networks. In *2nd International Workshop On Peer-To-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, 2003.

[15] Jamie Furness and Mario Kolberg. A Survey of Blind Search Techniques in Structured P2P networks. In *Proceedings of The 11th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, Liverpool, UK, 2010.

[16] Jamie Furness and Mario Kolberg. Considering complex search techniques in DHTs under churn. *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 559–564, January 2011.

[17] Jamie Furness and Mario Kolberg. Improving Wide Area P2P Service Discovery Mechanisms using Complex Queries. In Anand R. Prasad, John F. Buford, and Vijay K. Gurbani, editors, *Future Internet Services and Service Architectures*, chapter 9, pages 183–203. River Publishers, 2011.

[18] Jamie Furness, Farida Chowdhury, and Mario Kolberg. An Evaluation of EpiChord in OverSim. In *5th International Conference on Networks and Communication*. ACM Press, 2013.

[19] Jamie Furness, Mario Kolberg, and Marwan Fayed. An Evaluation of Chord and Pastry Models in OverSim. In *Modelling Symposium (EMS), 2013 European*, pages 509–513, 2013.

[20] Jun Gao and Peter Steenkiste. Efficient Support for Similarity Searches in DHT-Based Peer-to-Peer Systems. *2007 IEEE International Conference on Communications*, 5(1):1867–1874, 2007. ISSN 10842756. doi: 10.1053/siny.1999.0115. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4288982.

[21] Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor, and Robert Rallo. PlanetSim: A New Overlay Network Simulation Framework. *Software Engineering and Middleware*, pages 123–136, 2005. doi: 10.1007/11407386_10.

[22] Chris GauthierDickey, Virginia Lo, and Daniel Zappala. Using n-trees for scalable event ordering in peer-to-peer games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video - NOSSDAV '05*, pages 87–92, New York, New York, USA, 2005. ACM Press. ISBN 158113987X. doi: 10.1145/1065983.1066005. URL http://portal.acm.org/citation.cfm?doid=1065983.1066005.

[23] Ali Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD thesis, The Royal Institute of Technology (KTH), 2006.

[24] Ali Ghodsi, Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi. Self-Correcting Broadcast in Distributed Hash Tables. In *15th International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, Marina del Rey, CA, USA, 2003.

[25] Ali Ghodsi, LO Alima, and Seif Haridi. Symmetric replication for structured peer-to-peer systems. In *Proceeding DBISP2P'05/06 Proceedings of the 2005/2006 international conference on Databases, information systems, and peer-to-peer computing*, pages 74–85, Berlin, 2007. URL http://link.springer.com/chapter/10.1007/978-3-540-71661-7_7.

[26] Thomer M. Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. p2psim: a simulator for peer-to-peer (p2p) protocols. URL http://pdos.csail.mit.edu/p2psim/.

[27] Omprakash D Gnawali. A keyword-set search system for peer-to-peer networks. Technical report, Massachusetts Institute of Technology, 2002.

[28] A González-Beltrán, P Milligan, and P Sage. Range queries over skip tree graphs. *Computer Communications*, 31(2):358–374, 2008. ISSN 01403664. doi: 10.1016/j.comcom.2007.08.003. URL http://linkinghub.elsevier.com/retrieve/pii/S0140366407002940.

[29] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. Approximate Range Selection Queries in Peer-to-Peer Systems. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, 2003.

[30] H V Jagadish, Beng Chin Ooi, and Quang Hieu Vu. BATON : A Balanced Tree Structure for Peer-to-Peer Networks. In *VLDB '05 Proceedings of the 31st international conference on Very large data bases*, pages 661–672, 2005. doi: 10.1.1.118.5966.

[31] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim Simulator. URL http://peersim.sf.net.

[32] Raul Jimenez, Flutra Osmani, and Bjorn Knutsson. Sub-Second Lookups on a Large-Scale Kademlia-Based Overlay. In *11th IEEE International Conference on Peer-to-Peer Computing 2011*, number June, 2011.

[33] S. Lennart Johnsson and Ching-Tien Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, 38(9):1249 – 1268, 1989.

[34] Yuh-Jzer Joung and Li-Wei Yang. KISS: A Simple Prefix Search Scheme in P2P Networks. In *Proc. Ninth Int'l Workshop Web and Databases (WebDB '06)*, pages 56–61, 2006. doi: 10.1.1.84.3592.

[35] Yuh-Jzer Joung and Li-Wei Yang. Wildcard Search in Structured Peer-to-Peer Networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1524–1540, November 2007. ISSN

1041-4347. doi: 10.1109/TKDE.2007.190641. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4339217.

[36] Yuh-Jzer Joung, Li-Wei Yang, and Chien-Tse Fang. Keyword search in DHT-based peer-to-peer networks. *IEEE Journal on Selected Areas in Communications*, 25(1):46–61, January 2007. ISSN 0733-8716. doi: 10.1109/JSAC.2007.070106. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4062563.

[37] M. Frans Kaashoek and David Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. *Peer-to-Peer Systems II*, 2735:98–107, 2003. doi: 10.1007/b11823. URL http://www.springerlink.com/content/unmqcqy0yxpu32xp/.

[38] David R Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures SPAA 04*, 3279(i):36, 2004. ISSN 03029743. doi: 10.1145/1007912.1007919. URL http://portal.acm.org/citation.cfm?doid=1007912.1007919.

[39] Salma Ktari, Mathieu Zoubert, Artur Hecker, and Houda Labiod. Performance Evaluation of Replication Strategies in DHTs under Churn. In *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia (MUM '07)*, pages 90–97, Oulu, Finland, 2007. ACM. ISBN 9781595939166. doi: 10.1145/1329469.1329481. URL http://portal.acm.org/citation.cfm?doid=1329469.1329481.

[40] B Leong, B Liskov, and E Demaine. EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management. Technical Report 9, May 2006. URL http://linkinghub.elsevier.com/retrieve/pii/S0140366405003750.

[41] Ben Leong, Barbara Liskov, and Eric D. Demaine. Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management. In *Proceedings. 2004 12th IEEE International Conference on Networks (ICON 2004)*, volume 2004, pages 270–276, Singapore, 2004. IEEE. ISBN 0-7803-8783-X. doi: 10.1109/ICON.2004.1409145. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1409145.

[42] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth-efficient Management of DHT Routing Tables. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 99–114. USENIX Association, 2005.

[43] Nathan Linial and Ori Sasson. Non-Expansive Hashing. In *Proceeding STOC '96 Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, volume 18, pages 509–518, January 1996. doi: 10.1145/237814.237999.

[44] Min-Yen Lue, Chung-Ta King, and Ho Fang. Scoped Broadcast in Structured P2P Networks. In *Proceedings of the 1st international conference on Scalable information systems - InfoScale '06*, Hong Kong, 2006. ACM. ISBN 1595934286. doi: 10.1145/1146847.1146899. URL http://portal.acm.org/citation.cfm?doid=1146847.1146899.

[45] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *Peer-to-Peer Systems*, 2429:53–65, 2002. doi: 10.1007/3-540-45748-8_5. URL http://www.springerlink.com/content/2ekx2a76ptwd24qt.

[46] Luiz R Monnerat and Claudio L Amorim. D1HT: A Distributed One Hop Hash Table. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 1–10, Rhodes Island, Greece, 2006. IEEE. ISBN 1-4244-0054-6. doi: 10.1109/IPDPS.2006.1639278. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1639278.

[47] Daniel Nurmi, John Brevik, and Rich Wolski. *Modeling machine availability in enterprise and wide-area distributed computing environments*. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28700-1. doi: 10.1007/11549468_50.

[48] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 161–172, San Diego, California, United States, October 2001. ACM. doi: 10.1145/964723.383072. URL http://portal.acm.org/citation.cfm?doid=964723.383072.

[49] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-Level Multicast using Content-Addressable Networks. *Networked Group Communication*, 2233: 14–29, 2001. doi: 10.1007/3-540-45546-9_2. URL http://www.springerlink.com/content/ahdgfj8yj9exqe03/.

[50] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, page 10, 2004. URL http://portal.acm.org/citation.cfm?id=1247425.

[51] Antony Rowstron and Peter Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, 2001.

[52] O. D. Sahin, S. Antony, D. Agrawal, and A. El. Abbadi. PRoBe: Multi-dimensional Range Queries in P2P Networks. *Web Information Systems Engineering – WISE 2005*, pages 332 – 346, 2005. doi: 10.1007/11581062_25.

[53] Dom Schlienger, David Irvine, James Irvine, Mario Kolberg, Jamie Furness, Swee Keow Goo, Jorge Eliécer Gómez Gómez, and Velssy Hernandez Riaño. Works in Progress-A Survey, Cloud File Sharing, and Object Augmentation. *IEEE Pervasive Computing*, 11(2):96, 2012.

[54] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. In *Second International Conference on Peer-to-Peer Computing*, page 104, Washington, DC, USA, 2002. IEEE Computer Society. URL http://portal.acm.org/citation.cfm?id=824472.825529.

[55] Cristina Schmidt and Manish Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, Seattle, Washington, 2003.

[56] Cristina Schmidt and Manish Parashar. Enabling flexible queries with guarantees in p2p systems. *IEEE Internet Computing*, 8(3):19–26, May 2004. ISSN 1089-7801. doi: 10.1109/MIC.2004.1297269. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1297269.

[57] Cristina Schmidt and Manish Parashar. A Peer-to-Peer Approach to Web Service Discovery. *World Wide Web*, 7(2):211–229, June 2004. ISSN 1386-145X. doi: 10.1023/B:WWWJ.0000017210.55153.3d. URL http://www.springerlink.com/openurl.asp?id=doi:10.1023/B:WWWJ.0000017210.55153.3d.

[58] Moritz Steiner, Taoufik En-Najjary, and Ernst W Biersack. A global view of kad. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*, page 117, New York, New York, USA, 2007. ACM Press. ISBN 9781595939081. doi: 10.1145/1298306.1298323. URL http://portal.acm.org/citation.cfm?doid=1298306.1298323.

[59] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Conference on Applications,*

technologies, architectures, and protocols for computer communications (SIGCOMM '01), pages 149–160, San Diego, California, United States, 2001. ACM Press. ISBN 1581134118. doi: 10.1145/383059.383071. URL http://portal.acm.org/citation.cfm?doid=383059.383071.

[60] Daniel Stutzbach and Reza Rejaie. Characterizing Churn in Peer-to-Peer Networks. Technical report, University of Oregon CIS-TR-2005-03, 2005.

[61] Daniel Stutzbach and Reza Rejaie. Understanding Churn in Peer-to-Peer Networks. In *6th ACM SIGCOMM on Internet measurement (IMC '06)*, pages 189–202, Rio de Janeriro, Brazil, 2006. ACM. ISBN 1595935614. doi: 10.1145/1177080.1177105. URL http://portal.acm.org/citation.cfm?doid=1177080.1177105.

[62] Torsten Suel, Chandan Mathur, Jo-Wen Wu, Jiangong Zhang, Alex Delis, Mehdi Kharrazi, Xiaohui Long, and Kulesh Shanmugasundaram. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. In *In WebDB*, 2003. doi: 10.1.1.12.6196. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.6196.

[63] Chunqiang Tang and Sandhya Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. *System*, 1(ii):16, 2004. URL http://portal.acm.org/citation.cfm?id=1251175.1251191.

[64] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. pSearch: Information Retrieval in Structured Overlays. *ACM SIGCOMM Computer Communication Review*, 33(1):89–94, 2003. ISSN 01464833. doi: 10.1145/774763.774777. URL http://portal.acm.org/citation.cfm?doid=774763.774777.

[65] L. Viennot. Broose: a practical distributed hashtable based on the de-Bruijn topology. In *Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings.*, pages 167–174. Ieee, 2004. ISBN 0-7695-2156-8. doi: 10.1109/PTP.2004.1334944. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1334944.

[66] Vladimir Vishnevsky, Alexander Safonov, Mikhail Yakimov, Eunsoo Shim, and Alexander D. Gelman. Scalable Blind Search and Broadcasting in Peer-to-Peer Networks. In *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, pages 259–266, Cambridge, UK, 2006. IEEE. ISBN 0-7695-2679-9. doi: 10.1109/P2P.2006.34. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1698621.

[67] Vladimir Vishnevsky, Alexander Safonov, Mikhail Yakimov, Eunsoo Shim, and Alexander D. Gelman. Tag Routing for Efficient Blind Search in Peer-to-Peer Networks. In *11th IEEE Symposium on Computers and Communications (ISCC'06)*, pages 409–416. IEEE, 2006. ISBN 0-7695-2588-1. doi: 10.1109/ISCC.2006.157. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1691062.

[68] Sai Wu, Hong Gao, and Bei Yu. Supporting High Dimensional Range Queries in Peer-to-Peer Systems. In *Euro-Par 2006, Parallel Processing, 12th International Euro-Par Conference*, Dresden, Germany, 2006. doi: 10.1007/11823285_106.

[69] Zhichen Xu and Zheng Zhang. Building Low-maintenance Expressways for P2P Systems. Technical report, HP Laboritories, Palo Alto, 2002.

[70] Zhongmei Yao, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks. *Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 32–41, November 2006. doi: 10.1109/ICNP.2006.320196. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4110276.

[71] Changxi Zheng, Guobin Shen, Shipeng Li, and Scott Shenker. Distributed Segment Tree : Support of Range Query and Cover Query over DHT. In *IPTPS'06: Electronic publications of the 5th International Workshop on Peer-to-Peer Systems*, 2006.