Monash University

WebNS: Model for a Peer-to-peer Name Service

Decentralised, Human-readable, Secure, Scalable and Available.

This thesis is presented in partial fulfilment of the requirements for the degree of Bachelor of Information and Computer Sciences (Honours) at Monash University

By:

Craig Webster

Supervisor:

Jacques Steyn

Year:

2011

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the work of others has been acknowledged.

Signed

Name

Date

Acknowledgements

I thank my supervisor, Dr. Jacques Steyn, for his constructive feedback, helpful advice and supervision throughout this research. I also thank Sheelagh Walton, Neil Manson and Dr. Conrad Mueller for their recommendations. I thank my fellow honours students and friends, special mentions go to Barry Walls and Paula Murray, for adding some humour and levity to the year. I thank, and apologise to, the friends I did not see much of this year. Finally, I give thanks to my parents, for their unconditional love and support throughout my time at university. This was as much a test for them as it was for me.

Abstract

The incumbent Domain Name System in use on the Internet is a hierarchical system with multiple points of authority used to map human-readable names onto machine-readable addresses. This centralisation of power is juxtaposed to the distributed nature of the Internet and has three main problems: it is vulnerable to orchestrated attacks, the administrative costs of its maintenance are increasing and there are several recorded cases of the centralised power being abused for political reasons. Peer-to-peer networks are resilient to these three problems and may be used to construct an alternative decentralised name service. By carefully constructing a structured overlay network, ensuring efficient replication of name entries and making use of Byzantine agreement based algorithms to ensure consistency and security, a model for a decentralised, secure, scalable and available name service is created.

Keywords: decentralised name service, peer-to-peer name service, Byzantine agreement, Chord, distributed hash table, Byzantine fault tolerance, decentralised public key infrastructure, peer-to-peer, name service, Domain Name System, public key infrastructure

Table of Contents

List of Figures

List of Tables

1. Introduction

A name service is a system that binds computer-readable addresses to human-readable names. Name services remove the need for humans to remember complex and unnatural addresses when directing messages to computers over a network. The name service used on the Internet is the hierarchical Domain Name System. However, this hierarchical system is vulnerable to orchestrated attacks (Ramasubramanian & Sirer, 2004a; Ariyapperuma & Mitchell, 2007), involves ever increasing maintenance costs (Internet Corperation for Assigned Names and Numbers, 2010), and allows for political pressures to adversely affect the system (Holder, 2010; Intellectual Property Act of 2011, 2011).

The growing administrative costs of running the Domain Name System threaten the organisations that are responsible for its operation. The ability for external entities to affect the system, either through technological attacks or through political pressure, results in unreliable names that may be altered or removed. The Domain Name System forms a critical part of the underlying Internet infrastructure (Mockapetris & Dunlap, 1988), and its potential unreliability affects all applications and users dependent on name lookups. Considering the nature of network addresses, the uninability to look up a domain name in a naming service often means that the named resource is not locatable for the average Internet user.

An alternative to heirarchical systems is the peer-to-peer approach. Well constructed peer-to-peer networks have been proven to be incredibly fault tolerant (Ripeanu, Iamnitchi, & Foster, 2002; Al-Kassimi, 2005). They make use of the resources available on each peer to distribute the costs of running the system across every user (Milojicic, *et al.*, 2003; Coulouris, Dollimore, & Kindberg, 2005). Additionally, as peer-to-peer networks contain no central authority, they are immune to the abuse authority and political pressure.

The purpose of this thesis is thus the creation of a model for a name service operating on top of a well-constructed peer-to-peer network, addressing the shortcomings of the incumbent Domain Name System. More formally, this thesis aims to construct a name service that:

- Contains only decentralised components.
- Provides human-readable names, mapped to machine readable addresses.
- Maintains the security of the name-address mappings despite the presence of arbitrarily faulty parties or orchestrated attacks.
- Is scalable to a global number of users.
- Maintains the availability of the name-address mappings, even in the face of arbitrarily faulty parties or orchestrated attacks.

These five design goals are central to this thesis, and provide guidance and rigour to the review and evaluation of previous work, the technologies supporting the construction of the proposed model, and the proposed model itself.

The ultimate deliverable of this thesis is a model for a decentralised name service for use on the Internet, proven to conform to the five defined design goals. In addition to a theoretical discussion of the proposed model, a pseudocode example implementation is provided in **Appendix A**. This is intended to clarify the concepts in this thesis, and to guide the future implementation of systems based on the proposed model.

1.1. Methodology

This thesis follows a Design Science approach. Design science is an attempt to utilise existing theory to create artifacts that serve human purposes, as well as contribute back to the base of knowledge (Hevner, March, Park & Ram, 2004). It is thus driven both by the practical environment, and by the existing knowledge base. It focuses on the merger between theoretical truth and utility. Effective research designs solutions to identified, meaningful problems. Completed research simultaneously expands the body of knowledge through the publication of solutions or results. Scientific rigour is provided through the appropriate use of existing theory to inform design decisions. Design science differentiates itself from routine design by solving problems previously unsolved in a specific domain (Hevner, *et al.*, 2004).

This thesis aims to construct an IT artifact, informed by existing theoretical work. The artifact intends to solve a defined problem, meeting a set of known requirements. Moreover, the problem addressed in this thesis has not been solved previously; it is not a routine design problem. As such, a Design Science methodology is adopted to provide scientific rigour and to guide the design process in finding a solution of an existing and relevant problem.

Hevner, *et al.* (2004), in their seminal paper on Design Science in Information Systems Research, provide seven guidelines for the application of the Design Science methodology, which are briefly discussed below.

1.1.1. Guideline 1: Design as an Artifact

Information Technology deals with the artificial, rather than the natural world. IT research involves both the creation and the study of artificial constructs or artifacts (March & Smith, 1995). Artifacts are defined as "constructs, models, methods and implementations" and thus encompass both theoretical models, as well as their real world instantiations (March & Smith, 1995).

Design Research, by definition, aims to create a purposeful artifact to solve a previously unsolved problem. Because of the novelty of the solutions identified, artifacts are typically innovations that require some refinement or instantiation before being put into practice (Hevner, *et al.*, 2004).

Following this, the primary deliverable of this thesis is a model for a decentralised name service, providing human-readable names in a scalable and secure manner, while remaining available and reliable, even in the face of orchestrated attacks. Based on a literature review conducted in **Section 2. Background**, this is a thus far unsolved problem in Information Technology.

This thesis aims to be comprehensive of the problem domain through analytical evaluation of the proposed model. Instantiation of the model, followed by empirical testing, remains the subject of further research. Comprehensive testing of the model requires the creation of a peer-to-peer network consisting of a global number of nodes, and the capability to control all of those nodes' behaviours.

1.1.2. Guideline 2: Problem Relevance

Design Science represents the merger of theory and practice. A practical problem initiates theoretical research to find a solution (Hevner, *et al.*, 2004). These solutions are found through the construction of artifacts designed to change the world from its current state, to a defined "goal state".

A background discussion of the currently used Domain Name System, and a description of its shortcomings, is presented in **Section 2.1. Name Services**. These shortcomings are not simply theoretical; they potentially affect both the administrators of the Domain Name System, and every person and application dependent

on this service. The Domain Name System forms a critical part of the underlying Internet infrastructure (Mockapetris & Dunlap, 1988). Any failures it experiences create the possibility that named objects can no longer be located. As such, the removal of these shortcomings has practical significance to name service administrators, name owners and end users.

In order to remove these shortcomings, a goal state is proposed in the form of five design goals presented in **Section 2.3. Goals of Decentralised Name Services**. This thesis constructs a model for a decentralised name service within the bounds of these five goals.

To facilitate this model, the problem of decentralised public key infrastructure is addressed; a problem relevant not only to name services, but also so many other decentralised systems.

The effects of the proposed model, including the differences from the current system and the affects the on the various stakeholders of the current system, is presented in **Section 5. Comparison of Approaches**.

1.1.3. Guideline 3: Design Evaluation

Artifacts created through Design Science research must be evaluated to ensure their quality and prove that the defined problem has been solved successfully (Hevner, *et al.*, 2004). Hevner, *et al.* (2004) proposes five classes of evaluation methods: observational, analytical, experimental, testing and descriptive.

Following the introduction of the five design goals, evaluation metrics are provided in **Section 2.3.1. Design Evaluation Metrics**. The proposed model is evaluated in **Section 4. Evaluation and Attainment of Goals** based on these evaluation metrics. As this thesis details the creation of a model rather than an instatiation, the use of observational, experimental and testing based evaluation is not possible. Thus evaluation is done analytically and descriptively. The use of logical proofs based on previously defined design goals is supported by the Design Science Methodology defined by (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007). Due to the diversity of the defined design goals and technologies used, a more flexible logical reasoning approach is used, rather than formal logic, as suggested by Hevner, *et al.* (2004).

In addition to the evaluation of the proposed model itself, evaluation is performed during the literature review, and during the search for appropriate supporting technologies. In the literature review in **Section 2. Background**, existing works in the domain of decentralised name services are evaluated for their conformance to the five design goals. During the exploration of technologies in **Section 3. Supporting Technologies**, each alternative technology is evaluated in order to make informed decisions. While alternative technologies are also evaluated in terms of the five design goals, domain specific goals are also considered, taking into account the diversity of these technologies.

1.1.4. Guideline 4: Research Contributions

Research should be geared towards providing tangible contributions to theory, practice, or in the case of Design Science, both. Design Science research contributions take the form of artifacts, foundations or methodologies (Hevner, *et al.*, 2004). For a contribution to be meaningful, the problem solved should be novel, not simply a routine design problem.

This thesis contributes an artifact in the form of a solution to problems identified in the current Domain Name System. It provides foundations in the form of a model for a decentralised name service, improving on previous attempts which failed to achieve all five defined design goals. The design goals themselves may be used as a methodology for the evaluation of future decentralised name service attempts. The novelty of

the contributed model is proven in the literature review in **Section 2. Background**, where it is determined that no previous attempt at a decentralised name service has met all five proposed design goals.

1.1.5. Guideline 5: Research Rigour

Application of scientific rigour lends credibility to the results obtained. Design Science research closely follows the scientific approach (Baskerville, Pries-Heje, & Venable, 2009), involving the definition of a testable hypothesis, then the evaluation of that hypothesis.

The alternate hypothesis of this thesis is implied in **Section 2.2. Peer-to-Peer Networks**, where a possible solution to the problems plaguing the existing Domain Name System is proposed. Stated explicitly, the hypotheses of this thesis are:

H_A: A decentralised name service can be created, using a peer-to-peer network, offering human-readable names in a secure, scalable and available manner.

H_0: Decentralisation, human-readable names, security, scalability and availability are mutually exclusive design goals, and a name service obtaining all three is not possible using a peer-to-peer network.

The design process itself serves to test these hypotheses. Using the five design goals, every design decision is informed and can be evaluated. The evaluation makes use of the work of previous authors where possible, and analytical observation, descriptions and scenario building where previous work is not found. The diverse nature of the design goals, and the technolgoies supporting the proposed model, defy the use of mathimatical formal models. Rather, by making use of appropriate prior work and logical reasoning, research rigour is obtained (Hevner, *et al.*, 2004).

1.1.6. Guideline 6: Design as a Search Process

The proposed model consists of a unique combination of several existing and tested technologies and theories. As such, the design of this model involves the search for appropriate technologies supporting the creation of a decentralised name service. Design may take the form of an iterative cycle of generation of alternatives, and testing those alternatives against some criteria in order to make and appropriate and informed selection (Simon, 1996, pp. 128 – 129; Hevner, *et al.*, 2004).

The five goals of the proposed model serve as a simple starting point for the search for and evaluation of the various technologies and theories in the model. During this search and evaluation process, additional domain specific goals are highlighted. For example, when evaluating Distributed Hash Tables in **Section 3.2.3. Distributed Hash Table Selection Criteria**, meeting the high level goal of security involves meeting three lower level security goals. In this manner, the general design goals are made more concrete to each evaluated technology, and more rigorous comparisons can be made (Hevner, *et al.*, 2004).

A review of the literature and existing theory behind each evaluated supporting technology is used to generate alternatives. Using the domain specific goals, each alternative is evaluated for its suitability to be included in the proposed model. Each evaluation serves to highlight additional goals that need to be met, and thus the search process iterates. The search process ends when a technology is found that meets the defined goals, or when a reasonable compromise can be made.

1.1.7. Guideline 7: Communication of Research

It is suggested by Hevner, *et al.* (2004) that Design Science research must provide sufficient detail to allow technologically-oriented readers to implement the contributions of the work. The authors also emphasise that contributions and evaluations must be accessable to non-technical audiences.

Following this advice, the construction of the model and its evaluation is presented in a descriptive but detailed manner. The formal definitions of the model's operation, intended to be used for clarification of concepts or as a guide to implementation by technical readers, is included in **Appendix A**.

1.2. Thesis Organisation

This thesis roughly follows the Design Science methodology proposed by Peffers, *et al.* (2007). The methodology was created from a synthesis of seven pieces of work deemed influential in design science. All seven works have strong similarities in most aspects. Thus six activities were defined:

1. Identify problem and motivate
2. Define objectives of a solution
3. Design and development
4. Demonstration
5. Evaluation
6. Communication

Based on this, the remainder of this thesis is structured as follows:

The background of the problem area is introduced, discussing name services, their history and the shortcomings of the current implementations. Peer-to-peer networks are introduced with a discussion of their attributes that may be used to address the flaws identified in currently implemented name services.

Based on this discussion, five design goals are highlighted, allowing a concrete definition of the scope and success criteria of the proposed model. These five goals serve as the objectives of the designed solution.

To inform the remainder of this thesis and to determine the novelty of the solution, a literature review of existing name systems and prior attempts at decentralised name services is conducted. Each attempt is evaluated against the five defined goals, with strengths and weaknesses highlighted.

Informed by previous attempts, a framework for an idealised decentralised name service is defined. Following this framework, various technologies supporting the creation of a decentralised name service are discussed. Following Design Science guidelines of designing as a search process, each technology's alternatives are investigated, and evaluated by analytical or descriptive reasoning, informed by the five defined design goals. Based on these evaluations, the framework is populated through the selection of each piece of technology.

Using the five design goals of decentralised name services as success criteria, the proposed model is demonstrated and evaluated to ensure each design goal is adequately met. Each subsection addresses a single goal and provides a brief summary and evaluation of the model as a whole, with respects to the goal being discussed.

Finally, a discussion on the expected effects of the proposed model is presented. Ideas for the implementation of the proposed model are briefly discussed. Finally, the work in this thesis is summarised into a set of conclusions, a list of contributions, and directions for further research.

2. Background

2.1. Name Services

In order to locate objects and resources in any network, peer-to-peer networks included, the address where an object is stored is required. Addresses are typically machine readable bit strings that allow a network device to deterministically locate a destination.

Unfortunately however, these addresses are not human-readable, nor easily memorable. Thus, addresses are typically replaced with more human-meaningful names or Universal Resource Locators (URLs). In order to bind the human-readable name of an object to its machine routable address, a name service is used. Name services map human-readable resource identifiers to their addresses on a network. This removes the need for users to memorise complicated strings of numbers on order to navigate the network.

The ability to map human-meaningful names onto machine-readable addresses can be considered the primary design goal of name services.

2.1.1. History

The original name service for use on the Internet was based on the HOSTS.TXT file stored on all Internet-enabled hosts, used to map human-readable names to machine-readable addresses (Stewart, 2000). The "authoritative" version of the file was maintained by the Stanford Research Institute (SRI) Network Information Center and was distributed to all of the relatively few host machines in the early days of the Internet; however every site was free to maintain and distribute the file as desired. The growth of the number of machines using the Internet and requiring names prompted the creation of RFC 606 (Deutch, 1973), a proposal for storing host name-address pairings online, removing the need for each host to maintain an ever expanding list of bindings. The HOSTS.TXT file still remains on Internet enabled machines, used for configuration purposes; however the responsibility for maintaining the "authoritative" version of this file was removed from the SRI in September 1991 (Williamson & Nobile, 1991).

This completely centralised system sufficed for ten years between 1973 and 1983 before the administrative costs of maintaining the HOSTS.TXT file were recognised (Mockapetris & Dunlap, 1988; Stewart, 2000). Thus the specification cycle for the current Domain Name System began.

2.1.2. Current System

The naming system in use on the public Internet is the Domain Name System (DNS; Mockapetris, 1983a; 1983b); it maps human-readable names to machine-readable network addresses. Despite the Internet's decentralized nature and distribution of authority (Gillett & Kapor, 1997), the DNS is not decentralised. The DNS root zone is controlled by ICANN (http://www.icann.org/). Each zone under the root is similarly centrally controlled by a single organisation.

The DNS is a global system employing a hierarchy of servers to cope with the exponential expansion of the Internet; each server fulfils requests for its zone (Mockapetris, 1983a; 1983b; Mockapetris & Dunlap, 1988). At the top most level are the root servers, 13 clusters of servers serving requests for server addresses in the appropriate top level domains (Root Server Technical Operations Assn, 2011). TLDs sit directly under the root zone in the hierarchy, authoritatively responsible for all domain names registered under their domain (Postel, 1994). Each lower level domain is registered and thus controlled by an organisation; these may be further sub-divided if desired. Finally non-authoritative caching servers may be implemented to deal with higher loads; however these servers only receive names from higher level servers and do not assist with reliability, they cannot serve as backups of the authoritative servers.

This hierarchy allows the DNS to scale globally by delegating authority to local servers. Authority however, is the primary weakness of the DNS; despite the delegated nature of the system, single nodes still maintain authority. Ramasubramanian and Sirer (2004a) state that 79% of domain names are redundant to only two DNS servers. This makes these names vulnerable to orchestrated attacks; as few as two servers can be overwhelmed to remove one or more of these domains from the Internet. Combined with on-going concerns over the DNS's known vulnerability to denial of service attacks makes for an unstable infrastructure (Ariyapperuma & Mitchell, 2007). While redundancy may have improved since, the centralisation of control of domains still creates both a physical and administrative bottleneck.

Concerns over the scalability of the root zone were investigated by the ICANN in 2009 (Internet Corperation for Assigned Names and Numbers, 2010). The concern was in relation to the requirement that DNS accommodate IPv6, as IPv4 recently exhausted its address space. The problem was averted by limiting the delegation rates of the top level domains and continuing upgrades to the infrastructure involved. While the problem was indeed solved, it does bring attention to the massive administrative costs involved with the running of the DNS service.

Finally, in late 2010, the US Government removed 82 sites from the DNS under claims of copyright infringement (Holder, 2010). Additionally, early 2011 saw the latest in a line of legislative bills aiming to allow US government control of the DNS for copyright management. The bill would force DNS zone operators to prevent name resolution of offending websites (Intellectual Property Act of 2011, 2011). These actions have raised the fears of free Internet advocates such as the Electronic Frontier Foundation (https://www.eff.org/) and the We Rebuild group (http://werebuild.eu). They state that the central control of key Internet infrastructure may be abused.

In summary, three major concerns with the existing DNS are identified (Milojicic, *et al.*, 2003):

- Single points of failure and lack of redundancy for all names.
- Ever increasing administrative costs have raised concerns over the scalability of the DNS.
- Increasing pressure for governmental control.

These concerns are not simply theoretical; they have the potential to affect every person and application that relies on the DNS. The DNS forms a critical part of the underlying Internet infrastructure (Mockapetris & Dunlap, 1988). Removal of a domain name from the DNS, due to any of the above problems, forces the users of the previously-named resource to supply the network address of the resource to access it. The complexity of machine-readable addresses makes it unlikely that all users remember the addresses of all resources they use. This results in a lack of service for the user and a lack of visitors for the resource owner.

Peer-to-peer networks have proven themselves ideal in alleviating the problems of single points of failure by distributing responsibility across the network. They also cope with increasing administrative costs by organically growing to meet demand. Finally peer-to-peer networks, by not relying on any central authority, are immune the failures or pressures of organisations. However, to adequately supplement or replace the existing DNS service, a peer-to-peer implementation would need to be globally scalable. Thus in addition to the first design goal for the provision of human-readable names, the second design goal of global-use name services is the support of an arbitrarily large number of peers with no unreasonable degradation of service.

2.2. Peer-to-Peer Networks

2.2.1. What is "Peer-to-Peer"?

In practice, "peer-to-peer" is a well-used and generally understood term. Several definitions for peer-to-peer networks are found in literature. Shirky (2000) defines peer-to-peer networks as a class of applications taking advantage of resources on machines previously defined as clients, and autonomy free from centralised servers. Singh M. (2001) defines peer-to-peer architecture as simply "not the client-server model"; specifically mentioning decentralisation of control to limit single points of failure. Milojicic, *et al.* (2003) refers to peer-to-peer as "a class of systems and applications that employ distributed resources to perform a function in a decentralized manner." These definitions provide a broad classification of the peer-to-peer computing paradigm.

Schollmeier (2001) provides a more concrete definition, and the one informing this thesis. Peer-to-peer networks contain no node designated as authoritative and all nodes in the network share power and responsibility evenly. Milojicic, *et al.* (2003) classifies this as a "pure" peer-to-peer architecture as opposed to a "hybrid" architecture in which some centralised services or "super-nodes" co-exist on the otherwise decentralised network.

This thesis adopts the definition that peer-to-peer is a network architecture where rights and responsibilities are shared equally and no node holds authority over any network-wide service as in hierarchical network models. Resources are acquired from all peers, processing is done in a decentralised manner and costs and benefits of the functions are shared equally among involved peers. No single or group of peers can cause any major disruption to the operations of the network through their actions or disconnection. This thesis deals only with the concept of "pure" peer-to-peer networks, free from both centralised and autocratic authority, as well as from any "super-peers" of above average authority.

2.2.2. Benefits of peer-to-peer networks

Carefully connected peer-to-peer networks have been proven incredibly fault tolerant (Ripeanu, *et al.*, 2002; Al-Kassimi, 2005). Single node failures have no effect on the operation of the network, and even mass random node failures do not disrupt the functionality of the system. The decentralised and replicated nature of peer-to-peer networks can be utilised to create a more robust name service framework where no single point of failure exists.

Peer-to-peer systems have emerged to meet the need for growth in the Internet; they are not bounded by administrative costs. Rather, their resources grow in proportion to the number of users they support (Coulouris, *et al.*, 2005, pp. 397-398). Peer-to-peer networks take advantage of the resources of the nodes that utilise them (Milojicic, *et al.*, 2003). In this way, as the demands placed on the network grow, so too do the resources available to fulfil these demands.

These attributes may be used to remove the burden of administration from individual organisations in maintaining the naming architecture for the Internet. This has the additional benefit of making the architecture more resistant to failures of servers and the organisations that maintain them. As peer-to-peer networks have no controlling node and no authority, they are resilient to any abuse of power.

In order to gain these advantages however, a third design goal is highlighted. Deploying the system in a peer-to-peer architecture to obtain fault tolerance and scalability requires a pure peer-to-peer network. Any authoritative peers introduce points of failure, and points of authority that may be attacked or abused.

In summary, a peer-to-peer network architecture may be used to overcome the three identified problems with the Domain Name System, as illustrated by Table 1 below.

Identified Domain Name System problems	Peer-to-peer solutions
Centralised architecture creates single points of failure, any authoritative server being attacked may disrupt the entire system.	Distributed and replicated design ensures resilience to even massive node failures.
Centralisation of control creates ever growing administrative burdens.	Resources grow with the number of clients, scaling automatically to greater demand.
Central authority may be abused; organisational failures may disrupt the system.	No central authority exists; the system is immune to organisational politics.

Table 1: Summary of the problems facing traditional DNS in the first column, with identified solutions offered by peer-to-peer networks in the second column.

2.2.3. Limitations of peer-to-peer networks

Peer-to-peer networks' requirement that resources and responsibilities be shared with unknown third parties introduces significant security and performance issues (Schoder & Fischbach, 2003). Determining trust is the most pressing problem facing peer-to-peer networks.

Peer-to-peer networks are likely to contain faulty nodes that do not successfully execute protocol. Faulty nodes may be actively trying to subvert the protocol, or may simply be suffering a benign failure (Pathak & Iftode, 2006). Nodes do not always fail and stop, but may continue to act in an arbitrary, unpredictable and possibly malicious manner. These problems are defined as “Byzantine Faults” in the literature (Lamport, Shostak, & Pease, 1982). Due to the arbitrary nature of these faults, it is a challenge to determine which nodes are faulty, and an even greater challenge to determine which nodes are provably malicious.

As servers cannot distinguish between honest, faulty and malicious nodes, it is unreasonable to assume all nodes in a network are fault free. Services are thus required to operate in the presence of dishonest nodes. Possible solutions include placing trust in a few neighbouring nodes, or introducing a democratic rule, whereby a majority of nodes (where majority is defined by some number) in a group must agree before a message is trusted (Wallach, 2003).

In addition to the above limitations, peer-to-peer networks may suffer from “fairness” issues. A fair node provides and consumes resources in a balanced manner (Androutsellis-Theotokis & Spinellis, 2004). Some nodes faithfully execute protocols and allow use of their resources, while other nodes make use of these shared resources but do not share their own. These issues may not always be malicious however; nodes responsible for small or unpopular services may naturally be sharing fewer resources than other nodes. Fairness is considerably more difficult to protect against.

Given these limitations, in order to design a more robust model, two further design goals are introduced:

- Irrespective of trust boundaries, the system must be able to deliver unique and objectively correct name-address mappings, unmodified by any unauthorised party. This security must be achieved, even in the presence of malicious peers.
- The model must be reliable. Queries and commands must be executed successfully, irrespective of the state of any individual peer. Name entries stored on the network must persist beyond the connection and disconnection of nodes. This requires some form of replication to ensure entries are not lost at node disconnection; the performance loss of redundant storage is nullified by the security and reliability advantages gained.

2.3. Goals of Decentralised Name Services

The preceding discussion highlights 5 primary design goals this thesis aims to address in the modelling of a peer-to-peer name service. To summarise:

- The name system is decentralised, relying on no central authority. This is a strict requirement for the running of the system in a pure peer-to-peer network environment, gaining the benefits of peer-to-peer networks discussed above.
- The name system allows for the use of human-memorable names as mappings to their addresses. This is similar to the function of the existing DNS.
- Names are secure, as defined by the following requirements:
    - Names are unique, each name maps to exactly one address with no duplicate names. The same name does not exist more than once in the network. An adversary cannot cause two addresses to be mapped to the same name, possibly redirecting users away from the original address.
    - Names are modifiable only by the original publisher of the name entry.
    - Names are correct, despite the presence of faulty nodes acting in an arbitrary or malicious manner, possibly in collusion with other malicious nodes in order to undermine security. Names are correct if they are returned when requested, modified only by the original publisher.
- The system is scalable, able to handle large numbers of nodes without significant degradation of service. No significant performance degradation should be seen as the size and scale of the system increases. This is a requirement of any global system, and is required to rival the existing DNS implementation.
- The system has high availability; disconnections of nodes should not cause names or resources to become unreachable. This is the current weakness of the DNS; an orchestrated attack removing only a few nodes can cause lookups to fail. A peer-to-peer network is employed to solve this.

Secondary goals such as performance and load balancing are considered. However these are of lesser importance, and will be discussed as a subset of the primary design goals. For example performance and load balancing will be mentioned in relation to scalability concerns.

2.3.1. Design Evaluation Metrics

Design evaluation involves the definition of metrics and the evaluation of the design artifact (Hevner, *et al.*, 2004). Five design goals have been defined above. In order to facilitate the evaluation of the design artifact, as well as prior work on decentralised name services and alternative supporting technologies, the following evaluation metrics are defined:

- **Decentralisation** will be deemed to be attained if no node or participating entity holds any authority greater than any other node in the network.
- **Human-readability** will be measured based on the extent to which names stored in the system are natural for humans to read and remember. Specifically, names must be selectable by publishing entities. Knowledge of computer-generated names should not be required.
- **Security** will be evaluated based on resilience to Byzantine faults. Byzantine faults refer to any arbitrary behaviour not specified by protocol. These faults are defined in **Section 3.1.2. System Model**. If all security goals are met, despite these faults, the model is deemed secure.

- **Scalability** shall be measured based on the growth in routing time or number of messages sent as the number of nodes in the network increases. Specifically, any growth rate exponential to the total number of users will be deemed unacceptable. Additionally any unpredictable routing time or message number will be deemed unacceptable, as these are impossible to verify objectively.

- **Availability** will be measured as the system's resilience to node failures. The system should cope with any single node failure at any point in time with no adverse effects. Additionally the system should follow the above scalability guidelines when recovering from such failures, to allow mass node failure to occur with no adverse effects.

Descriptive and analytical methods will be employed while evaluating the above metrics. Arguments informed by existing research will be used to determine the suitability of each examined system or technology. Where appropriate research does not exist, or where a simple answer exists, a brief scenario will be constructed to prove or disprove the suitability of a system or technology.

2.4. The Challenge of Decentralised Name Services

Wilcox-O'Hearn (2003) states that the three design goals of decentralisation, human-meaningful names, and secure mapping of names to addresses are not simultaneously achievable. This is known informally as “Zooko's Triangle”.

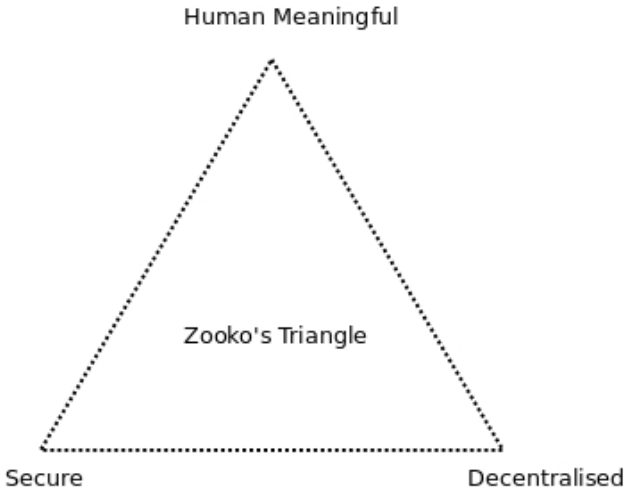

Figure 1: Zooko's Triangle, each side of the triangle represents a design tradeoff for decentralised name services.

Wilcox-O'Hearn’s argument (2003) stems from the fact that names are secure if a name-address mapping is verifiable and unquestionably tied to its value. This is achievable in two ways:

- Names are self-authenticating.
    - If names are cryptographic hashes of the objects they identify or addresses they map to, it can be verified that the name belongs to the object or address by simply calculating the hash.
- Names are authenticated and signed by a trusted third party.
    - If name-address mappings are certified and signed using a digital certificate by a mutually trusted third party, the integrity and authenticity of the binding can be verified by checking the digital signature using the trusted third party's publicly known key.

Both of the above schemes provide integrity for name entries. Both can also provide authenticity of the publisher of the name if the publisher's details are included in the name entry. However the first solution is not human-meaningful, as hashes as names are not easily memorable. The second solution relies on a central authority, violating the decentralisation design goal.

It is thus during the implementation of security that either decentralisation or human-meaningful names are sacrificed. The third option is to forgo security, instead relying on trust between users of the system or relying on the user to make an informed decision on the trustworthiness of a particular mapping.

2.5. Prior Implementations Attaining Two Goals

The previous section highlights three simple implementations for name services, each implementation obtaining two of the three goals of decentralisation, human-readable names and security. Each of the following sections corresponds to one of the edges of Zooko's Triangle in Figure 1.

2.5.1. Human-readable and Secure

The global Domain Name System (DNS; RFC 882 & RFC 883; Mockapetris, 1983; 1983) is the naming service in use on the Internet to translate human-readable names into machine routable network addresses. The DNS is secure and highly scalable due to its hierarchical structure and caching (Mockapetris & Dunlap, 1988). However the DNS is centrally controlled and regulated as a result. The root zone is controlled by ICANN (http://www.icann.org/), and each zone in the hierarchy is responsible for the signing of the entries in their zone. This central control ensures the security of name entries as changes are regulated by single, hopefully trusted, entities. But of course this centralisation prevents the DNS's operation on a pure peer-to-peer network, forfeiting the scalability and fault tolerance benefits they provide.

Several alternative DNS roots are in operation as alternatives to the primarily American controlled DNS. The OpenNIC project (OpenNIC, 2011) aims to keep the DNS free for all users; the Unified Root (Unifiedroot, 2011) allows non-Latin domain names. The Unified Root project may be rendered obsolete by ICANN’s announcement of internationalised domain names (Internet Corporation for Assigned Names and Numbers, 2009). These alternative roots have similar pitfalls as the traditional DNS; they all rely on a central authority to ensure the security of name bindings. Worse, without the funding, but with the condemnation, of the large Internet management organisations (Internet Architecture Board, 2000), these roots are even more vulnerable to increasing administrative costs and denial of service attacks.

2.5.2. Decentralised and Secure

Decentralised and secure names are typically created to be self-authenticating. That is, the name and the object named are verifiable using no external mechanisms. This is the scheme normally used in distributed hash tables such as Chord (Stoica, Morris, Karger, Kaashoek, & Balakrishnan, 2001), Pastry (Rowstron & Druschel, 2001), Kademlia (Maymounkov & Mazières, 2002) and Tapestry (Zhao, Huang, Stribling, Rhea, Joseph, & Kubiatowicz, 2004). Names of objects are typically cryptographic hashes of these objects, or hashes of a subset of the objects’ identifying information. In this way, name and object are inextricably linked, ensuring security of the mapping even as it is distributed across the network. Objects are then locatable using a key based routing scheme (Dabek, Zhao, Druschel, Kubiatowicz, & Stoica, 2003). Distributed Hash Tables will be discussed in further detail later in this thesis. While the use of cryptographic hashes ensures the security of the systems, hashes are inflexible and not human memorable.

A real-world implementation of a global peer-to-peer system utilising the concept of hashes for names is the Freenet project (Clarke, Sandberg, Toseland, & Verendel, 2010). The Freenet project is a peer-to-peer reimplementation of the Internet, allowing for content distribution resistant to censorship. Freenet’s

distributed content naming solution is the Content Hash Key (CHK), created by taking the secure hash of the object they index (Clarke, *et al.*, 2010). They are immune to forgery and the name and value are inextricably linked by the hash function.

While these distributions are decentralised and secure, they neglect the fundamental purpose of a naming system, to allow human-meaningful names to be used to address objects on a network.

2.5.3. Decentralised and Human-readable

Zahn and Schiller (2005) introduced MAPNaS, a lightweight and distributed name service for use on ad-hoc mobile networks. Their implementation uses a distributed hash table to store name-address mappings on individual nodes in the mobile network. By taking the secure hash function of the resource's name or identifier, these bindings can be located using a key-based routing scheme, similar to the distributed hash tables discussed in the previous section. This implementation provides for availability through the use of replication and clustering. However due to the constraints of mobile systems, names are not securely tied to their addresses. A node may freely edit the name entries they become responsible for as no signing of bindings is done to ensure integrity or authenticity. The entire scheme relies on the good behaviour of participating nodes, an assumption that unfortunately cannot be made for global networks.

Search engines such as Google, Microsoft's Bing or Yahoo! may be thought of as name services. A user supplies a name or query, and objects are returned based on their query. There is no central control over the results that are returned, they are determined using complex algorithms, and a user may simply use a different engine if one does not give them the desired results. Thus a search engine maps human-meaningful names onto objects on a network in a decentralised fashion. However these systems are inherently insecure. There is no one-to-one or unique mapping of a name to an object or address, results may change as the algorithms evolve or new sites and services emerge.

These services fulfil the need for human-readable name mappings in a decentralised manner. However the lack of security and continuously changing search algorithms means that two identical queries executed at different places or times may return different results. Even worse, the lack of security in implementations such as MAPNaS (Zahn & Schiller, 2005) allows malicious parties to corrupt name-address bindings; redirecting users of the system to incorrect and possibly dangerous addresses.

2.6. Prior Implementations Attempting Three Goals

Several attempts have been made to achieve a decentralised, human-meaningful and secure name service for the use in peer-to-peer and other decentralised networks. However each of these implementations relies on a different set of assumptions and most do not achieve the desired design goals completely. The following section discusses the common approach seen multiple times throughout the literature reviewed. This is followed by a review of other, more novel approaches. Finally, the Namecoin project is discussed, which warrants its own section as it is the most recently attempted implementation of a decentralised, secure name service and is the leading competitor in this domain.

2.6.1. The Common Approach to Decentralised Name Services

A common approach to the creation of a decentralised name service is the decentralisation of the name database through the use of a distributed hash table and the provision of security through the addition of DNS security extensions (DNSSEC; Eastlake, 1999) and a public key infrastructure. This infrastructure allows for the digital signing of name entries, providing integrity and authenticity. However these implementations also mandate the use of a centralised certification authority. This thesis aims to remove this dependence on a central certification authority and implement the same security in a decentralised manner.

Cox, Muthitacharoen, and Morris (2002) provide a framework for the implementation of name services over peer-to-peer networks, called the Distributed Domain Name Service (DDNS). Their implementation makes use of a Chord structured distributed hash table for efficient storage of name entries on the network (Stoica, *et al.*, 2001). Entries are made secure through the use DNSSEC extensions to authenticate modification of name entries and to ensure the integrity of name entries upon retrieval. DDNS makes use of chained key certificates, each key signed by another key, up the hierarchy until a well-known root key is encountered. In this way keys can be verified through the use of a centrally trusted key.

The DDNS model is expanded in the creation of the Co-operative Domain Name Service (CoDoNS) (Ramasubramanian & Sirer, 2004a). CoDoNS was created as a backup system for the existing DNS with the design goals of extreme scalability and security to overcome the current weaknesses of DNS. CoDoNS relies on a distributed hash table known as Beehive (Ramasubramanian & Sirer, 2004b) to serve as the distributed database and provide replication services. Similar to the DDNS, CoDoNS makes use of DNSSEC extensions.

Similar to the CoDoNS model, even suggesting the use of the Beehive distributed hash table, is the Object Information Distribution Architecture (OIDA) proposed by Fabian (2009). OIDA is a name service designed for use in a pervasive “Internet of Things”, designed for use in short range radio frequency networks. This implementation provides additional security to provide confidentiality of queries and responses. However once again it relies on DNSSEC to provide keys for security functions. OIDA’s offerings of confidentiality are attractive and not in opposition with the stated design goals, however have been deemed out of scope for this thesis. Reasonably, the distributed security mechanisms proposed in this thesis can be extended to provide confidentiality; this remains the subject of further work.

Awerbuch and Scheideler (2004) provide a complex example of a distributed name service. Their model focuses primarily on ensuring that at no point in time can an adversarial entity become the majority in any isolated group of nodes. They state that allowing nodes to remain stationary in the network for long periods of time allows adversaries to slowly infiltrate critical points in the network. They introduce the idea of **churning nodes**, removing and moving nodes constantly, and requiring the republishing of names at regular intervals to ensure malicious nodes cannot “surround” honest nodes. Each honest peer, referred to as an “entity” in this thesis and representing a person or organisation, maintains a small set of nodes under its control. As one node is removed, another can be used; all nodes of a peer can be identified by the peer’s name. This model has similarities with the model proposed in this thesis, primarily the replication of name entries and the requirement that a majority of replica nodes return agreeing responses. However their model requires the use of a certification authority for the secure assignment of names to peers to secure against masquerading as other peers. It is the removal of this requirement that differentiates this work from theirs. The requirement for an authority to provide names for peers operating in a name service seems counterproductive. Their goals for the prevention of Sybil attacks (Douceur, 2002) or Eclipse attacks (Singh, Ngan, Druschel, & Wallach, 2006) inform the research conducted in the securing of distributed routing in overlay networks used in this thesis.

The latest implementation of this model is the DotP2P Project (2010); an open source and community driven project aiming to replace the existing Domain Name System. The system stores name mappings on a peer-to-peer network, however implements a central authority for the issuing of certificates. Thus the system centralises its Top Level Domains (TLDs), citing Zooko's Triangle (Wilcox-O'Hearn, 2003) to justify this design decision. The system does not allow for the revocation of certificates. Thus malicious nodes cannot have their certificates revoked and compromised certificates may be used to impersonate legitimate users. The centralisation of the TLDs and the reliance on a trusted third party for maintenance of certificates undermines the decentralisation goals of the project.

While each of the above implementations provides for a secure name service, DNSSEC uses a centralised certification authority certificates to ensure the validity of public-private keys. This reliance on a central certification authority for the creation of a public key infrastructure undermines the goals of decentralisation. This thesis aims to replace the central certification authority with a decentralised public key infrastructure. This infrastructure both supports, and is supported by the decentralised name database. This infrastructure is discussed in **Section 3.7. Key Management**.

2.6.2. Other Implementations

2.6.2.1. Multiple Roots

Based on the fact that many alternative DNS roots exist, Qiang, Zheng, and Shu (2006) propose a system allowing for the use of multiple roots simultaneously. Their proposal and this thesis share the goal of rectifying the centralised nature of DNS to improve reliability and obtain the “Internet spirit” of openness and shared control.

However due to the fact that multiple roots are used, the security goal of uniqueness of name entries is not guaranteed. While they suggest a method of popularity to define the authoritative name, this proves unreliable in the face of faults, violating their goal of increased reliability. Should the primary DNS fail, their system retrieves the name from another DNS root, however this name-address binding may be different than the popular one in the primary DNS. Thus names may be resolved to different addresses, depending on the state of the system. This proves unpredictable and possibly insecure, preventing it from becoming a piece of critical Internet infrastructure.

2.6.2.2. Unstructured Overlays

Microsoft (2006) provided an implementation of their Peer Name Resolution Protocol (PNRP) in Windows XP, later upgraded in Windows Vista. This implementation relies on the random links between peers in order to achieve routing between them; an unstructured distributed hash table is formed. Upon lookup, a peer calculates the ID of the resource is wishes to locate and contacts the peer indexing the ID closest to the desired ID. The contacted peer then either returns the requested resource, an address of a peer with an ID closer to the desired ID, or an error if the contacted peer does not know any peers with a closer ID. If an error or “dead-end” is encountered, the requesting node backtracks, selecting a new node address forwarding requests along a new path. Requests are continually sent from the requesting node, until the resource is located. Name entries are published to the publisher’s neighbouring nodes as well as random nodes in the network attempting to create a random “small-world network” (Watts & Strogatz, 1998). The need for backtracking provides no predictability in routing, and the less-than-optimal paths taken severely limit the scalability of this solution.

SocialDNS (St. Juste, Wolinsky, Lee, Boykin, & Figueiredo, 2010) is a project aiming to augment the existing DNS system. The project details a name service running on top of a number of peer-to-peer virtual private network links. Nodes each maintain a small database of name entries. Similar to Microsoft's PNRP, the network does not rely on any formal structuring. It uses a two hop broadcast technique to query other nodes' databases. Queries are restricted to two hops to limit the flooding of messages. Thus, the system does not promise global uniqueness. Names may be registered multiple times and a ranking system is used to present results to users; or the system requires user intervention when operating in a “trusted” manner. Users manually add names to their databases or approve names imported from the network. While attempts have been made to make this process user-friendly, it is still not ideal for the lay-user. Lack of scalability, unique names and transparency to the user limit this implementation.

The unstructured nature of these systems means no probabilistically bounded estimates to routing time can be made, nor can it be guaranteed that any particular name entry is reachable at all (Chen, Ramaswamy, & Meka, 2007). This unreachability in unstructured overlays prevents the guarantee of unique name mappings, as seen in the SocialDNS case (St. Juste, *et al.*, 2010). In a secure system, names should only be registered once and any attempts to publish a mapping with the same name should fail. A reliable mechanism for the publishing, updating and locating of names is thus required. This requirement, paired with the design goal of availability, necessitates the use of a technology that can reliably publish, update and query multiple replicas of a mapping simultaneously.

These requirements highlight the need for a structured overlay network to ensure scalability and availability of the system. They also highlight the undesirable nature of broadcasts and backtracking, each request should generate a reasonable amount of traffic and return either a name entry, or a definitive failure message indicating the name entry does not exist.

2.6.2.3. Web-of-Trust Providing Authentication

As illustrated in the common approach to the distribution of name entries, decentralisation is often lost when implementing security through the use of a centralised certification authority for the signing of name entries and the management of public-keys. Zimmerman, (2001) introduced the web-of-trust model to create a decentralised alternative for public key infrastructure.

The web-of-trust model has been attempted in the implementation of the Telecomix DNS (We Re-Build, 2010). *We Re-build* (http://werebuild.eu) is an anonymous activist group collaborating to oppose issues that threaten access to an Internet free of intrusive surveillance and censorship. They introduced the Telecomix DNS in response to the US government's assumption of authority of the DNS root zone. Telecomix DNS creates a set of nodes to handle the DNS root zone. These nodes are connected via a web-of-trust providing for the security of name entries. Entries are signed by each root node's public-key. Based on the trust shared between the roots, the entry becomes indexed on each root.

The drawbacks of web-of-trust models are discussed in detail when examining the alternative means of implementing a public key infrastructure for the secure use of public-keys in **Section 3.7.3. Web of Trust**. Summarising the work in that section, web-of-trust models require either physical authentication of public-keys, or trust to be shared through hopefully trusted endorsements. As the physical authentication of every key is not feasible, and long chains of endorsement introduce risks, web-of-trust models cannot scale to global networks (Wilson, 1998). This fact is recognised by the designers of the Telecomix DNS (We Re-Build, 2010). The system creates an architecture where the root zone is shared only among a relatively small number of nodes. This designation of “root” nodes defeats the design goal of decentralisation.

Other than the web-of-trust, Wallach (2003) mentions briefly that one can trust in the majority of peers on the network in order to reach consensus. This majority trust can be seen in Cachin and Samar’s (2004) work on distributed name services.

2.6.2.4. Threshold Cryptography

Cachin and Samar (2004) fulfilled the need for reliable interactions with a distributed database through the implementation of an atomic broadcast protocol (Cristian, Aghili, Strong, & Dolev, 1995) and threshold cryptography (Desmedt & Frankel, 1990). Their solution removes the central authority found in DNS by distributing the responsibility for the signing of name entries to multiple nodes. A single private key is created and manually distributed to the group of responsible nodes. These “super-nodes” then work together to sign name entries. Their solution is Byzantine fault tolerant for some percentage of malicious nodes, depending on the threshold cryptography scheme employed.

This solution partially alleviates the problem of centralised authority; however it still relies on distinguished server nodes to jointly authenticate name entries, creating a point of authority that can be attacked, although to a lesser extent than traditional DNS. The manual key distribution limits scalability in a peer-to-peer environment, and no mention is made for the distribution of public-keys to authenticate the signed entries. Finally, as keys are created and shared once, the system does not grow in proportion to its user base, sacrificing a major benefit of peer-to-peer networks and failing to address DNS scalability concerns.

2.6.2.5. Handshaking Key Exchange and Byzantine Agreements

Despite the weaknesses of Cachin and Samar's (2004) model, it highlights a family of protocols that may be used to ensure reliable and secure querying and modification of distributed databases. These algorithms, including atomic broadcast and threshold cryptography, provide secure operations, even in the presence of a minority of malicious or faulty nodes. These algorithms are referred to as "Byzantine fault tolerant", due to their similarity or reliance on the Byzantine Generals Problem (Lamport, *et al.*, 1982).

Utilising the majority vote Byzantine agreement algorithms is P2PNS. P2PNS was proposed by Baumgart (2008) in his creation of a peer-to-peer session initiation protocol for mobile ad-hoc networks. His model provides for the decentralised storage of name-address mappings in a distributed hash table. Replacing the centralised public key infrastructure, security is provided by each host attaching their public-key to each message for authentication. This is referred to as "handshaking", or proving one's identity with the first message sent. In Baumgart's implementation (2008), the binding between a public-key and its owner is somewhat secured by the node identifier being a hash of the public-key. This makes it impossible to forge a fraudulent certificate with a desired node ID, as cryptographic hash functions are not reversible.

Node IDs are used in P2PNS to determine which nodes are responsible for the storage of a name entry. P2PNS assumes that a majority of nodes responsible for a name entry are honest, and thus the correctness of name entries is secured through majority vote algorithms. However, as certificates are supplied at handshaking and may be freely edited by their owners, there is no method to revoke a malicious node's certificate and remove that node from the network, even if malicious activity is detected.

While P2PNS attempts to secure node ID assignment through the use of crypto-puzzles, IDs remain valid indefinitely. Allowing nodes to keep IDs indefinitely allows an adversary to select IDs over time (Awerbuch & Scheideler, 2004). An adversary may thus infiltrate the groups of nodes responsible for an entry's storage (Singh, *et al.*, 2006). This can be prevented by revoking IDs faster than they can be selected, randomising group membership (Awerbuch & Scheideler, 2004). Without certificate revocation, this is impossible.

As such, P2PNS allows known malicious nodes to remain in the network, and cannot secure node ID assignment, allowing an adversary to slowly infiltrate groups. The assumption that a majority of nodes are honest may thus be jeopardised. As this assumption is used to ensure the correctness of name entries in P2PNS, its violation means that P2PNS cannot be considered secure. This thesis uses a similar approach of majority agreement on entries, however stores certificates online to allow revocation.

2.6.3. Dot-Bit project and NameCoin

Dot-Bit (Dot-BIT Project, 2011) is a project with goals similar to this thesis, particularly the removal of centralised control over the naming infrastructure. The project provides for unquestionable ownership of name entries, and entries that cannot be altered by any party other than the owner. The project is founded around the NameCoin software (available from https://github.com/vinced/namecoin), a decentralised and secure naming infrastructure. The project is a derivative of the BitCoin project, which aimed to create a secure and decentralised digital currency. The projects thus share large portions of their codebase.

The NameCoin system works as follows. Individual machines “mine” namecoins by repeatedly performing a cryptographic hash function on selected strings, seeking an output with a fixed number of leading zeroes. All machines collaborate to publish a “block” once every ten minutes. Each block is a function of the previous block and all recent transactions. The entire chain of blocks is currently maintained by all nodes in the network, steadily increasing in size as each new block is published every ten minutes. Names are registered by issuing a transaction, attaching a name-address pair to an owned namecoin. This transaction is stored in the blockchain, signed by the owner of the namecoin used to publish the name. The blockchain contains a record of every transaction performed, and every name-address pair.

As the blockchain continues to grow, each transaction becomes steadily more secure as each new block is published, effectively signing the previous block. As long as a majority of the contributing CPU power is owned by honest parties, a dishonest party cannot take control of the blockchain and control transactions.

Names are then retrieved by sequentially scanning the entire blockchain, retrieving the name-address pairs attached to every namecoin. Currently, names can be retrieved by maintaining the entire blockchain on every machine desiring access to the system. Alternatively a client can connect to a third party server that maintains the blockchain, removing the burden of maintaining the blockchain on each machine.

These two retrieval methods highlight the weakness in the NameCoin system. The blockchain, in the system’s current state, grows infinitely. Each block is a product of all preceding blocks, thus the end of the chain cannot simply be removed. Having each and every node maintain the entire blockchain as it steadily increases in size places an unnecessary burden on clients, severely limiting scalability. Additionally, as name-address mappings are placed into the blockchain in the order they are published, they are not sorted by name. This precludes the use of efficient search algorithms that may locate name entries in better than linear time, and forces the use of a sequential scan of the entire blockchain.

The alternative solution of connecting to a server removes the decentralisation of the system, relying on designated DNS resolvers. It additionally removes the requirement that a node contribute to the system in order to use it. This may limit the scalability or security of the system; as only a small proportion of users actually contribute, an attacker requires far less resources to become the majority in the system.

3. Supporting Technologies

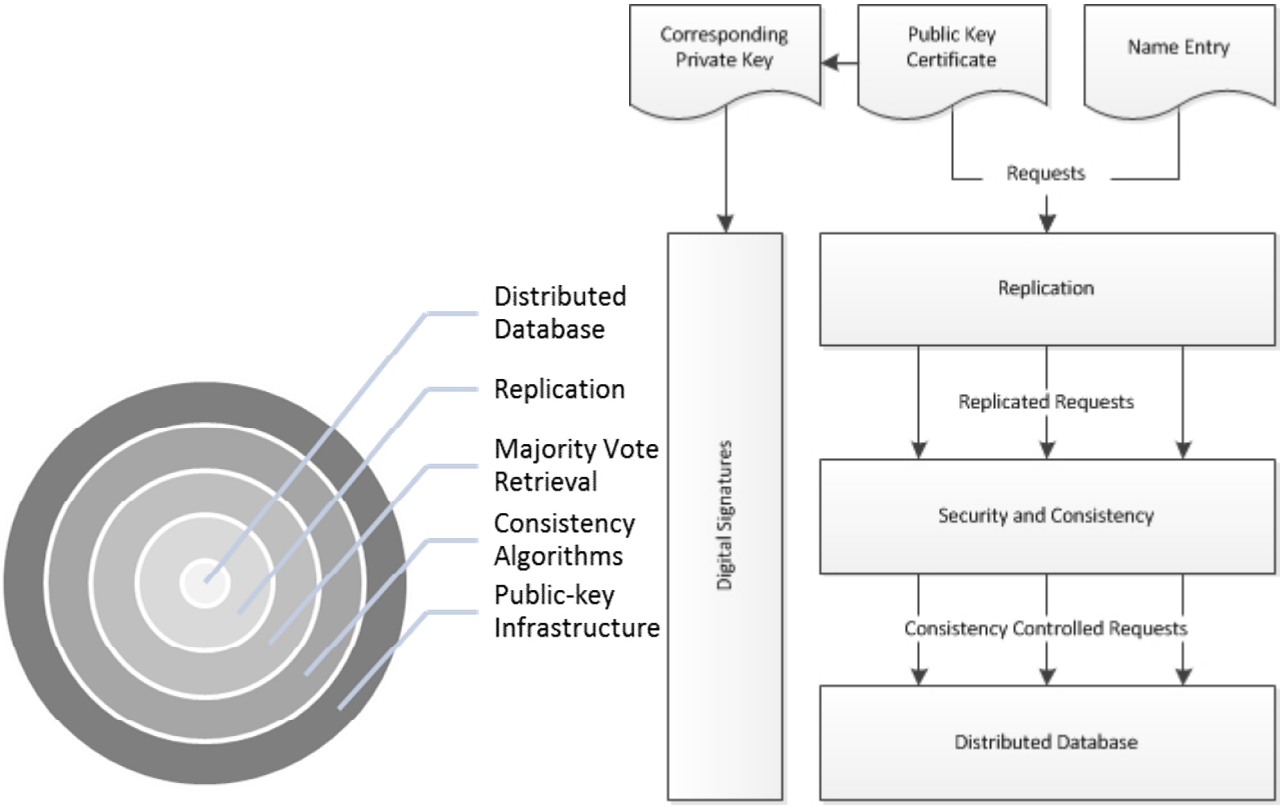

Figure 2: A conceptual overview of the proposed model.

Informed by the presently available work detailed above, the following model is proposed:

- At the core of the system, a database is implemented using a distributed hash table. The distributed database stores both name entries and public-key certificates, both signed by the publisher.
- Entries in the database are replicated over several nodes to ensure availability.
- Majority vote "Byzantine agreements" are introduced to promote security and consistency in the replicated distributed database during retrieval and modification of entries.
    - Mechanisms for querying all replicas of an entry and ensuring the correctness of the returned response are introduced.
    - Consistency algorithms are employed to ensure the atomicity of publishing and updating of entries. When publish or update commands are invoked, all replicas of the published or modified entry collaborate to ensure consistent modification.
- Finally, building off of majority vote algorithms, a distributed public key infrastructure is implemented, allowing for the use of public-key certificates and digital signatures.

The concentric circles on the left of Figure 2 are intended to highlight the progressive building nature of the model, and serves to structure the remainder of this section. Each layer is a requirement for the layers above it. The layered diagram on the right provides a framework, illustrating the placement of, and relationship between, each of the investigated technologies supporting a decentralised name service. The diagram illustrates the steps taken by a client in order to access a name entry or certificate in the model.

3.1. Preliminary Concepts and Assumptions

3.1.1. Definition of Preliminary Concepts

Before continuing with the discussion of technologies, the following concepts need to be clarified:

In the proposed model, cryptographic hash functions are used to create unique identifiers or keys for entries and nodes on the network. They are also used in the creation of message digests for use in digital signatures, discussed below. Cryptographic hash functions are a specific form of one-way functions mentioned in prior literature. A cryptographic hash function may be referred to as simply a “hash function”, and its output may similarly be called a “hash”. A cryptographic hash function has the following properties (National Institute of Standards and Technology, 1993; Stallings, 2007, pp. 64 – 65):

- A hash function can be applied to a message or object of any size.
- A hash function produces a fixed length output.
- A hash function is easy to apply to any message or object, for efficiency purposes.
- Given a hashed value, it is computationally not feasible to reverse the function and recover the message. This is known as the “one-way property”.
- It is not feasible for an adversary to find a set of any two messages where the hash of one message is identical to the hash of the other. This is known as “strong collision resistance”.

Digital signatures are used as an authentication mechanism to secure integrity, authenticity and non-repudiation in the proposed model. Digital signatures have the following properties (Diffie & Hellman, 1976; Rivest, Shamir, & Adleman, 1978; Goldwasser, Micali, & Rivest, 1988):

- A digital signature is created by creating a digest of the signed message using a cryptographic hash function, encrypting this digest with the creator’s private-key and appending this to the message.
- A signature can only be created using a private-key. Without knowledge of a user's private key, that user's signature cannot be created. Trust is placed in the strength of the hash function and encryption algorithm to maintain this assumption. For simplicity, it is assumed that breaking of either algorithm is sufficiently difficult to be considered impossible, and an honest user’s private-key remains a secret.
- As the signature is a function of both the message and the creator’s private key, an attacker cannot modify the message or object without violating the integrity of the signature. That is, any changes in the message require changes in the signature. Additionally, signatures are unique to messages, a signature cannot be removed and appended to another message.
- The identity of the signer can be determined by recalculating the digest of the message or object, decrypting the signature with the creator’s public-key and comparing the results.
- As signed messages may not be modified, nor may signatures be generated without knowledge of the creator’s private-key, the creator cannot deny the sending of a signed message, as no other party would be able to create this message.

3.1.2. System Model

The proposed model consists of a network of *n* nodes, each node owned by a specific entity. An entity may be an individual or organisation. An entity defines the owner of a private-key, which identifies each node uniquely. An entity may be in control of multiple nodes, owning multiple private-keys.

The model is designed to store name-address mappings, “name entries”, and public-key certificates. These are collectively referred to as “entries”. Nodes are identified by a “node ID”, and entries are identified by a “key”. Further discussion of IDs and keys is provided in **Section 3.2. Distributed Hash Tables as a Decentralised Name Database**.

Name entries have the structure:

{*Key*, *Name*, *Address*, *Sequence Number*, *Publisher Node ID*}

Certificates have the structure:

{*Publisher Node ID*, *Publisher Public-Key*, *Sequence Number*, *Time of Publish*, Other information}

Both name entries and certificates are digitally signed by their publisher.

Certificates are valid from their *Time of Publish* or “timestamp”, until a defined time-to-live has expired.

Entries may be published and retrieved. Name entries may be updated to change their *Address*, and certificates may be updated to change their optional identifying information. Name entries may also be transferred, or republished under a new publisher or certificate, while the old publisher forfeits ownership of the name entry.

Each entry is stored redundantly on *r* nodes in the network, these *r* nodes are referred to as “replica nodes” for a specific entry. Operations publishing, accessing or modifying entries communicate with all replica nodes for that entry. These operations require a quorum of 67% of *r* replica nodes for the entry to honestly perform the operation for it to succeed. The value of *r* is defined at system creation and does not change. Higher values of *r* provide greater resiliency to attacks and failures, but suffer in performance and scalability. Further discussion of *r*, its selection and use is provided in **Section 3.3. Replication**.

In contrast to the hierarchical structure of the DNS, the namespace allowed by the proposed model is flat. Names may be of any length and of any alphabet or syllabary. Names are passed through a cryptographic hash function to become network usable “keys”; the only limitation on names is their ability to be passed through a cryptographic hash function.

The protocol operates on top of the User Datagram Protocol (UDP; Postel, 1980). As such, no assumptions are made for the guaranteed delivery or sequencing of messages. Messages are not acknowledged unless explicitly stated; even then, acknowledgements are in the form of a response to the message sent.

The proposed model requires periodic upkeep and heartbeat messages to be sent. An integer constant *m* is defined. *m* represents the time, in seconds, between periodic maintenance events. These events include maintenance of the network, and the maintenance of the entries stored on the network.

An abstract “honest” node is defined. To any query, the honest node responds with the objectively correct answer based on local knowledge. All honest nodes provide similar, non-conflicting responses to any specific request unless otherwise stated. Similarly, if any honest node executes a command, all honest nodes execute the same command (Défago, Schiper, & Urbán, 2004). Honest nodes do not modify messages, send incorrect or conflicting messages, fail to send or forward messages, or delay messages in any manner.

Of the population of *n* nodes, *t* may experience Byzantine faults. Byzantine faults imply that a node may not simply fail and stop; it may continue to operate in an arbitrary manner, including malicious behaviour or collusion with other faulty nodes. Malicious behaviour is considered a subset of faulty behaviour. Faulty nodes may exhibit the following behaviour, adapted from Cristian, Aghili, Strong, and Dolev’s fault definition in their work on Atomic Broadcast (1995):

- Omission faults, a faulty node may refuse to initiate or forward messages, or may send messages only intermittently.
    - Faulty nodes may also experience excessive delays on the delivery of their messages. Messages arriving after an operation has terminated are functionally identical to omission faults.
- Link faults, a minority (<33%) of links connecting nodes may fail, causing messages across these links to become delayed indefinitely. Any node experiencing complete link failure is functionally identical to an omission fault.
- Integrity faults, a faulty node may have their messages modified in transit. It is assumed that the computationally bound adversary cannot correctly recreate a digital signature for another node.

Additionally, based on literature surrounding Byzantine fault tolerance (Lamport, *et al.*, 1982), the following faults are defined:

- Incorrectness, a faulty node may return an incorrect response to a query; differing from the expected response under perfect conditions. Alternatively, a node may respond to a message that should be forwarded, masquerading as the destination node. These messages are correctly signed, and thus are undetectable by authentication mechanisms such as digital signatures.
- Message conflict, all commands are directed to all replica nodes, a faulty node may issue differing messages to each replica. A faulty replica node may also issue differing messages when communicating with each other replica node. This is a deliberate attempt to subvert the protocol and is considered provably malicious behaviour.

Adversaries are considered to be computationally bounded, and $t < n / 3$, this is referred to as the honest majority assumption. Restated, 67% or more nodes in the network behave as honest. 67% is the minimum number of nodes that must be honest to ensure a majority decision (Lamport, *et al.*, 1982; Bracha & Toueg, 1985). This is explained further in **Section 3.4. Byzantine Agreement**.

This assumption presumes that a brute force Sybil attack (Douceur, 2002) is not possible; the number of honest nodes is so large that an adversary cannot insert enough faulty nodes to become the majority. Additionally, it is assumed that any pseudo-random subset of nodes selected will contain 67% or more honest nodes, and an Eclipse attack (Singh, *et al.*, 2006) is similarly not possible. Discussion on securing this honest majority is presented in **Section 4.6. Attainment of the Honest Majority**. The rest of this thesis defines 67% of r or more nodes as the “majority”, and the remaining 33% or less as the “minority”.

It is assumed that all malicious nodes may be under the control of a single entity, and thus may act in collusion. The adversary may manipulate the answers of faulty nodes freely. Additionally, it is assumed that the adversary is in partial control of the network and may modify, insert, delay or delete messages where this does not violate the honest majority assumption, or where it is specifically prohibited by specific security measures.

3.2. Distributed Hash Tables as a Decentralised Name Database

Upon analysis of existing distributed name services, the *de-facto* implementation of a distributed database for the storage of name entries is the distributed hash table. The following sections attempt to synthesise commonalities found in all distributed hash table implementations reviewed.

Distributed Hash Tables (DHTs) are highly scalable, efficient data structures operating over a large number of nodes, typically on a peer-to-peer network. The concept and problem statement was introduced by Plaxton, Rajaraman, and Richa (1997), and many varied implementations have been proposed since. These implementations have been theoretically proven to provide O(log *n*) routing time or better and require O(log *n*) degree routing tables or less (Stoica, *et al.*, 2001; Rowstron & Druschel, 2001; Maymounkov & Mazières, 2002; Zhao, *et al.*, 2004). Routing time, routing table degree and message complexity differ between implementations and the design decisions in each. DHTs have also been proven empirically scalable through their use in global content distribution networks such as Napster (Fanning & Fanning, 1999) and Freenet (Clarke, *et al.*, 2010).

Distributed Hash Table implementations proposed in the literature invariably contain the following components (Dabek, *et al.*, 2003):

- A key based routing mechanism by which nodes are assigned IDs and messages are given a key. Messages are forwarded recursively to nodes with IDs closer to the desired key.
- An overlay network, providing a structured organisation of nodes and links and defining routing tables of individual nodes to ensure reliable message delivery in an acceptable time.

3.2.1. Key based routing

Key based routing (KBR) forms the lowest layer in structured overlay networks such as DHTs (Dabek, *et al.*, 2003). It is the means by which messages are forwarded from one node to another, to their final destination. Key based routing provides a mechanism for message delivery that does not require a source know the network address of the destination, rather it needs only an address of a node nearer to the destination than itself. This removes the requirement of global network knowledge. Each node need only know a fraction of the network for efficient routing, enabling typical O(log *n*) degree routing tables.

Hash tables work on the principle of key → object bindings. A key is created for an object, the object is then stored in the hash table at a location determined by the key (Black, 2009). Locally, using an associative array indexed by key, hash tables can provide lookup times bounded by O(1). Distributed hash tables work on a similar premise; object keys are created, and objects are stored on a node with the ID most similar to the key.

Objects on the network are assigned keys by taking the cryptographic hash of the object's identifying information. The process is similar for the assignment of IDs to nodes. For objects, identifying information may be a hash of a subset of the object's content or name; for nodes this may be a public-key, IP or MAC address. The use of a cryptographically secure hash function has the advantage of pseudo-randomly selecting keys, distributing the load evenly and promoting fairness (Fabian, 2009).

The number of unique keys, or keyspace, is determined by the length of cryptographic hash function's output. A cryptographic hash function with output length *s* bits is able to create a keyspace of size 2^*s*. That is, 2^*s* unique node IDs or object keys.

Nodes are responsible for objects with keys that are "closer" to their node ID than to any other node ID. A node responsible for an object is referred to a "home" node or a "root" node in the literature. The definition of "close" is different in each implementation, but takes on a basic form, similar to the following:

- Chord (Stoica, *et al.*, 2001) defines closeness between a key and a node by the largeness of the difference between the node ID and the object key. The closest node for a key is determined by:

  minimum_positive_value(node ID – object key)

- Kademlia (Maymounkov & Mazières, 2002) defines closeness as the largness of the bitwise exclusive or of the node ID and the object key. The closest node for a key is determined by:

  min(node ID XOR object key)

- Protocols such as Tapestry (Zhao, *et al.*, 2004) define closeness as the number of contiguous bits the node ID and the object key have in common, reading from left to right. The closest node, or in Tapestry's case, nodes for a key is determined by:

  max(length of the common prefix of the node ID and object key)

Each forms offers similar key distribution and routing characteristics. The selection between each possibility is dependent on the design of the overlay network, discussed subsequently.

Messages sent using the KBR contain the key of the object they wish to publish, update, retrieve or otherwise interact with. When a node receives a message with a key for which it is not the home node, it searches its local routing tables and forwards the message to the node it deems closest to the key. This close node may or may not be the destination, if it is not the destination, that node performs the same process. The message is recursively forwarded to the closest known node, until it reaches its destination.

Routing is reduced to a recursive Greedy algorithm. Greedy algorithms are simple for the same reason that they are limited – they do not "think ahead" (Black, 2005). Each node makes the best possible decision with only locally known information. Well-constructed overlay networks, defined in the next section, allow a Greedy algorithm to make efficient choices without sacrificing simplicity.

3.2.2. Overlay Network

Overlay networks are network structures created on top of physical networks; introducing a layer of abstraction in order to change a basic property of the underlying network. An alternative to an overlay network would be to change the underlying network directly, however this is not a goal attainable in public peer-to-peer networks where nodes may join and leave on an ad-hoc basis. There are two classes of overlay networks in use, unstructured and structured (Lua, Crowcroft, Pias, Sharma, & Lim, 2005; Tsoumakos & Roussopoulos, 2006).

3.2.2.1. Unstructured Overlay Networks

Unstructured overlay networks organise peers into a random graph, with little or no control of the links that appear between peers. Peers have no knowledge of the rest of the network and are thus forced to make only local-optimal decisions. Messages are typically broadcast or pseudo-randomly "walk" around the network until they reach their destination (Tsoumakos & Roussopoulos, 2006) corresponding to a breadth-first and depth-first search respectively. Broadcast techniques flood the network in a small area, they send an inefficiently large number of messages and if the desired node is more than a short distance away, the node may never be found. Random walk techniques take less-than-optimal paths through the network; they may also never reach their destination when the network naturally groups nodes into clusters (Chen, *et al.*, 2007). Empirically, broadcast's shortcomings are observed in the Gnutella protocol; random walk's disadvantages can be seen in Microsoft's PNRP, discussed below.

Gnutella (Kirk, 2003) is an overlay network used for file sharing and distributed search. It operates through the broadcasting of messages to locally known peers, who fulfil the request, or continue to broadcast if they cannot. In this way each request generates a flood of queries from a node. Ritter (2001) provides a mathematical explanation of the implication of Gnutella's broadcast nature; in short, even small queries generate an exponentially increasing amount of traffic as they propagate through the network. In a heavily used system like a name service, these floods of queries are unacceptable.

Microsoft's (2006) PNRP makes use of random walks around the network to locate name entries. PNRP performs backtracking if a dead-end is reached and a query cannot be forwarded, leading to unpredictable and unnecessarily long times to locate destinations. Due to the fact that a random walk may never find the desired node, as unstructured networks make no attempt to remove network partitions and clusters, these systems do not promise global connectivity or uniqueness (Chen, *et al.*, 2007).

These shortcomings make unstructured networks unpredictable and unscalable, making inefficient use of the network they operate on (Lua, *et al.*, 2005). Structured overlay networks emerged to rectify these flaws. For the remainder of this thesis, only structured overlay networks are referred to.

3.2.2.1. Structured Overlay Networks

DHTs implement overlays to structure the routing of messages between peers. By structuring the routing in a network, applications are allowed to locate objects in a "probabilistically bounded, small number of network hops" (Wallach, 2003, p. 1), while only maintaining a relatively small number of connections to other nodes (Wallach, 2003; Lua, *et al.*, 2005).

Structured overlay networks aim to create "small-world networks" in which any two nodes are only separated by a small number of links. This is done in by maintaining links to many nearby nodes, and a few far away nodes (Watts & Strogatz, 1998). Messages to nodes some distance away can traverse a large portion of the network in a single hop, to be routed more precisely by the nodes closer to the destination.

Overlay networks can be categorised by a geometric representation of their routing tables. Existing implementation include Ring (Chord: Stoica, *et al.*, 2001), Hypercube (CAN), Butterfly (Viceroy), Tree (PRR: Plaxton, *et al.*, 1997; Tapestry: Zhao, *et al.*, 2004) and Exclusive-OR based (Kademlia; Maymounkov & Mazières, 2002) structures (Gummadi, *et al.*, 2003). The choice of these geometries has effects on the performance, routing table degree and resilience to failure of the DHT. The typical tradeoff seen is that between the size of routing tables and the static resilience to simultaneous node failure.

Structured overlay network models vary in the strictness of their structure (Gummadi, *et al.*, 2003; Liu, *et al.*, 2004). Implementations such as Chord (Stoica, *et al.*, 2001) suggest strictly organised routing tables with each entry in the routing table refers to a single node at a specific position in the keyspace. Routing is done by forwarding messages to a single node closest to the destination node. Nodes closer to the destination are more likely to know the address of the destination, and routing is done recursively.

Later overlay networks such as Pastry (Rowstron & Druschel, 2001), Kademlia (Maymounkov & Mazières, 2002) and Tapestry (Zhao, *et al.*, 2004) allow flexibility in their routing tables. These implementations group nodes into "buckets" based on their IDs, each bucket representing a specific area in the keyspace. Nodes are free to fill these buckets with any and as many node addresses in that area as they wish, up to a pre-defined maximum. These are then ordered by latency (in Tapestry), time last seen (in Kademlia) or other characteristics. Routing is performed by forwarding messages into the desired area of the keyspace by selecting any node from the desired bucket, typically the node with the lowest latency or highest uptime. Messages are recursively forwarded into smaller and more accurate areas.

The flexible approaches have the advantage of more efficient routing as the faster or more reliable links are chosen at each hop. Their disadvantages however lie in the increased complexity involved with routing table maintenance and dealing with node insertions, and their increased susceptibility to Eclipse attacks (Singh, *et al.*, 2006). Strictly structured overlay networks are less susceptible to Eclipse attacks, as they do not include measures of proximity or latency which may be corruptible by an adversary (Castro, Druschel, Ganesh, Rowstron, & Wallach, 2002; Wallach, 2003). However, removing these measures disallows the optimisation of routing, resulting in poorer performance.

3.2.3. Distributed Hash Table Selection Criteria

Relating back to the five defined goals of the proposed model defined in **Section 2.3. Goals of Decentralised Name Services**, nearly all structured DHT implementations will allow for the storage of human-readable entries in a decentralised and scalable manner. Availability and reliability of entries is discussed in **Section 3.3. Replication**. Peer-to-peer networks, DHTs in particular, have specific security concerns relating to the presence of untrusted nodes in the network. Castro, *et al.* (2002) and Wallach (2003) provide two similar checklists to assist in assessing the security attributes of overlay networks. They identified the following security issues affecting structured overlay networks:

- Secure assignment of node IDs.
- Secure maintenance of routing tables.
- Secure forwarding of messages.

3.2.3.1. Secure Assignment of Node IDs

As described in the above section, node IDs are used when determining where to place entries in the DHT, they are also used when determining the set of nodes in a finger table. Without the secure assignment of node IDs, an adversary may select node IDs corresponding to the key or keys of a particular entry, or corresponding to the IDs of routing table nodes. This allows an adversary to obtain higher than average membership in a defined set of nodes or force an honest node to communicate exclusively through these adversarial nodes. This violates the assumption of an honest majority, allowing an adversary to gain control of an entry or node using only a small number of resources. This is defined in literature as the Eclipse attack (Singh, *et al.*, 2006).

Recommended solutions include the establishment of a central authority to regulate the assignment of node IDs (Castro, *et al.*, 2002; Wallach, 2003). While this may curb the free selection of node IDs, it is not feasible in a pure peer-to-peer environment. Other solutions include the use of a cryptographic hash function for the generation of keys and node IDs, an adversary would need to attempt a large number of hash functions in order to select a key or ID. Finding an input that produces a hash output with *z* number of bits common to a target output has been found to take O(2^*z*) operations (Back, 2002).

It is recognised that systems that allow nodes to occupy the same position in the keyspace for an indefinite amount of time are non-survivable (Awerbuch & Scheideler, 2004). In these systems, adversaries may slowly infiltrate the network, selecting node IDs through trial and error, regardless of the difficulty in doing so. A conceptually simple solution is “churning” of nodes (Awerbuch & Scheideler, 2004). If a node may be removed from the overlay, even against its will, it is possible to force a node to rejoin the network at a different position. This forces a node to give up its node ID. If this churning is done at a rate faster than the reversal of the function assigning node IDs, the infiltration is thwarted.

3.2.3.2. Secure Maintenance of Routing Tables

Even in the case that node IDs are assigned in a secure manner, it is still assumed that an adversary controls *t* nodes in the network. As such, every routing table or other pseudo-randomly selected group of nodes is expected to contain a minority of faulty nodes. Several overlay network architectures (Pastry, Kademlia, Tapestry, etc.) employ flexible routing tables, to obtain performance benefits. With flexible routing tables, nodes are selected based on some measureable network characteristic, typically aiming to minimise routing time.

Through the manipulation of the network, an adversary may delay or prevent the delivery of messages from honest nodes. An adversary thus makes their nodes appear as more attractive alternatives to honest nodes. This increases the rate at which messages are passed through adversarial nodes, threatening the honest majority assumption (Singh, *et al.*, 2006).

While the use of flexible routing tables provides improved performance, they are a recognised vulnerability to Eclipse attacks (Sit & Morris, 2002; Singh, *et al.*, 2006); it is typically recommended that tightly constrained routing tables be used for security reasons (Castro, *et al.*, 2002; Wallach, 2003). These constrained routing tables only consider node ID when determining routing paths or group membership, thus reducing the problem of securing routing to secure node ID assignment, solved in the previous section.

One last possible attack on routing tables is an incorrect node responding to keepalive messages (Sit & Morris, 2002). Should a node with an ID similar to the destination node of a keepalive message respond to that message, the incorrect node may become indexed in the finger table of the DHT. This issue can be solved by ensuring the correct destination receives the message, and is discussed in the next section.

3.2.3.3. Secure Forwarding of Messages

In an ideal system, any message sent would be forwarded flawlessly from source to destination without need for additional mechanisms. Unfortunately, due to the unreliable nature of networks and the presence of faulty nodes, this is not possible. Messages may become delayed, corrupted or lost entirely as they are forwarded.

Additionally, without global knowledge of the network, it is impossible to determine if a response originates from the node with the node ID closest to the key contained in the corresponding request. It is feasible that dishonest nodes near the destination node in the keyspace may refuse to forward messages and instead respond to each message, masquerading as the destination node.

The following issues, specific to the key-based routing mechanism, are identified:

- Forward messages to incorrect or inefficient nodes, or delaying message forwarding.
- Refuse to forward messages altogether, creating a link fault.
- Respond to messages when they should rather be forwarded, masquerading as the destination node.

Forwarding messages to inefficient nodes increases the length of the path taken by messages. This may break the O(log *n*) path length bound promised by structured overlay networks. Other than creating an inefficient path for a message, this has no damaging effect on the network. Should a message be delayed long enough, the message will time out and be considered lost, creating a link fault.

The two remaining security concerns are correctable by utilising multiple, redundant, unique routes to a destination (Castro, *et al.*, 2002; Wallach, 2003). This requires the overlay to maintain multiple diverse routes to any node. If the most desirable route to a node fails or if the security of a message is in question, forwarding the message to multiple entries in the routing table should find alternative routes, circumventing faulty nodes. This may be less effective in systems such as Tapestry where messages to the same node tend to converge along the same path for efficiency in caching (Zhao, *et al.*, 2004).

Summarising the previous two sections, the following matrix is provided:

	Unstructured		Structured	
	Random Walk	Flooding	Constrained	Flexible
Message Complexity (number of messages sent)	Unpredictable, only optimal if first random path is correct.	Exponentially increasing with each network hop.	Optimal O(log *n*)	Optimal O(log *n*)
Routing Hops	Unpredictable, only optimal if first random path is correct.	Optimal for unstructured overlay. Typically fixed, Gnutella = 7.	Optimal O(log *n*)	Optimal O(log *n*)
Routing Time	Unpredictable	Optimal for unstructured overlay.	Function of physical distance between nodes.	Minimises effects of physical distance.
Simplicity	Optimal	Optimal	Simple algorithms for routing table management.	Complex algorithms for routing table management, requires measures of proximity.
Reliability (probability of object being found, assuming it exists)	Not guaranteed	Guaranteed if object is within a fixed number of hops. Gnutella = 7.	Guaranteed Ring-based overlays provide best resilience.	Guaranteed
Secure node ID assignment.	Achieved with a cryptographic hash function.	Achieved with a cryptographic hash function.	Achieved with a cryptographic hash function.	Achieved with a cryptographic hash function.
Secure routing tables.	Ad-hoc routing tables.	Ad-hoc routing tables.	Resilient to routing table attacks.	Susceptible to routing table attacks.
Secure message forwarding.	Unpredictable	Optimal	Varies with network geometry, Ring-based overlays provide diverse routes.	Varying with network geometry.

Table 2: A summary matrix of four Distributed Hash Table classifications.

3.2.4. Distributed Hash Table Selection: Chord

To partially deal with the issue of secure node ID assignment, a cryptographic hash function is used to assign IDs to nodes and keys to entries. All discussed DHT implementations are compatible with this approach. The problem of secure node ID assignment is further alleviated through the periodic churning of nodes, discussed in **Section 3.7. Node Churn**.

To maintain the security of routing tables, a design decision is made to sacrifice the performance benefits offered by proximity routing strategies offered by overlay networks such Tapestry and Kademlia. Rather, a strictly structured implementation is selected, to obtain the security offered by constrained routing tables.

Secure and reliable message forwarding requires the maintenance of multiple redundant and unique routes from any source to any destination. The comparative study of overlay network geometries by Gummadi, *et al.*, (2003) recommends Ring-based overlays as achieving the greatest resilience to node or link failure. Ring-based overlay networks provide O(log *n*) connections to the overlay at every node (Stoica, *et al.*, 2001) and maintain O((log *n*)!) unique routes from any source to any destination (Gummadi, *et al.*, 2003). Furthering the comparative study, Loguinov, Kumar, Rai, and Ganesh (2003) indicate that Chord-based DHTs are "*k*-connected", where *k* is the number of links maintained by each node. This suggests that Chord maintains no weak-points and the most likely means of causing network partitions is the simultaneous failure of all of a node's O(log *n*) finger table nodes. Although the Loguinov, *et al.* study (2003) recommends De-Bruijn-based DHTs, Liu, Chen, Yuan, Lu, and Xu (2004) provide empirical evidence indicating Chord's superiour static resilience in comparison to De-Bruijn graph networks. Al-Kassimi, (2005) provides empirical evidence that even while suffering from 30% and 50% simultaneous node failure, a Chord DHT can still ensure delivery of every message. The only adverse effects of the massive node failure were an increased average path length, and the requirement that requests be retried if the timed out.

As such, Chord (Stoica, *et al.*, 2001) has been selected for use in the proposed model. Chord is an early ring-based overlay network and is widely referenced and critically analysed by other DHT implementations. The rich literature surrounding Chord is attractive when considering possible enhancements and optimisations. Chord has the advantages of simplicity and relative efficiency in routing. Its key-based routing is compatible with cryptographic hash functions for secure node ID assignment. Chord also offers intrinsic security and reliability based on tightly constrained routing tables and its support for many distinct routing paths.

3.2.4.1. Chord Key-Based Routing

Keys for name entries are generated through the cryptographic hash of the name selected. Node IDs are generated by taking the cryptographic hash of the node's public-key. Keys for certificates are the node IDs of their owning node. Chord defines "closeness" between keys and IDs by the magnitude of the clockwise integer distance between a node ID and an entry key.

Normally, the home node for a key is the node directly succeeding it in the keyspace. Chord then locates the key by contacting nodes closer to, but preceding, the home node. The information of the home node is retrieved from its direct predecessor, and the home node is then contacted (Stoica, *et al.*, 2001, pp. 4 – 5).

Chord routing naturally contacts predecessors for keys, contacting the home node then requires an extra hop. Thus, as a slight adjustment, entries are stored on the predecessor for their key, rather than the successor. Thus after the predecessor for a key is found, no extra hop is required.

The size of the keyspace is 2^*s*, where *s* is the number of bits in the output of the cryptographic hash function. The Chord keyspace is circular, the first key and last key are adjacent.

The closest node to an entry is determined by:

Closest Node ID = minimum_positive_value((entryKey – nodeID) mod keyspaceSize)

Where:

- nodeID is the current node's identifier.
- entryKey is the key of the desired entry.
- keyspaceSize is the maximum number of keys possible in the keyspace.
- minimum_positive_value() is a function determining the smallest of a set of values.

To illustrate, consider a system catering for a maximum of 8 nodes (keyspaceSize = 8) shown in Figure 3. The hash function creates IDs and keys 3 bits in length to index nodes and entries on the network. Four nodes are present in the network, altogether responsible for the storage of eight entries.

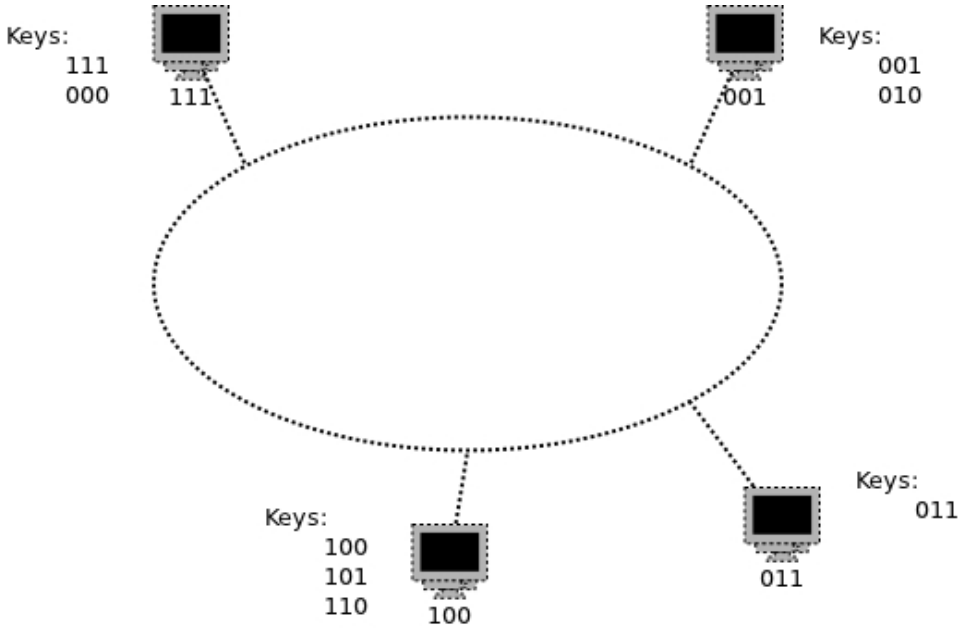

Figure 3: An example of a Chord circular keyspace with a maximum of 8 keys and nodes. The node ID is listed underneath each nodes, with the keys each node is responsible for listed alongside each node.

Lookups are done recursively by locating the node with the node ID closest to, but preceding or equal to the given key in the current node's routing table. Messages are forwarded to the node listed, which will in turn forward the message to the node closest to the key in its routing table. This repeats until the home node for the key is found. This node cannot forward the message, and thus responds.

3.2.4.2. Chord Overlay Network

Chord's routing table is known as a "finger table". It consists of a series of ideal node IDs in the keyspace, and the addresses of the nodes closest to those IDs. It is maintained using the following formula:

fingerTable[*i*] = lookup((nodeID + 2^(*i*-1)) mod keyspaceSize)

Where:

- *i* is an integer indexing the fingerTable array. *i* ranges from 1 to the number of bits in the node key and object identifiers.
- lookup(key) is a function for locating the network address of the node responsible for the key defined by parameter "key", defined in the previous section.

To illustrate, consider a system catering for a maximum of 8 nodes (similar to the system in Figure 3) shown in Figure 4. The entries in **Node 000**'s finger table are indicated by solid black arrows. Each other node maintains a similar set of links.

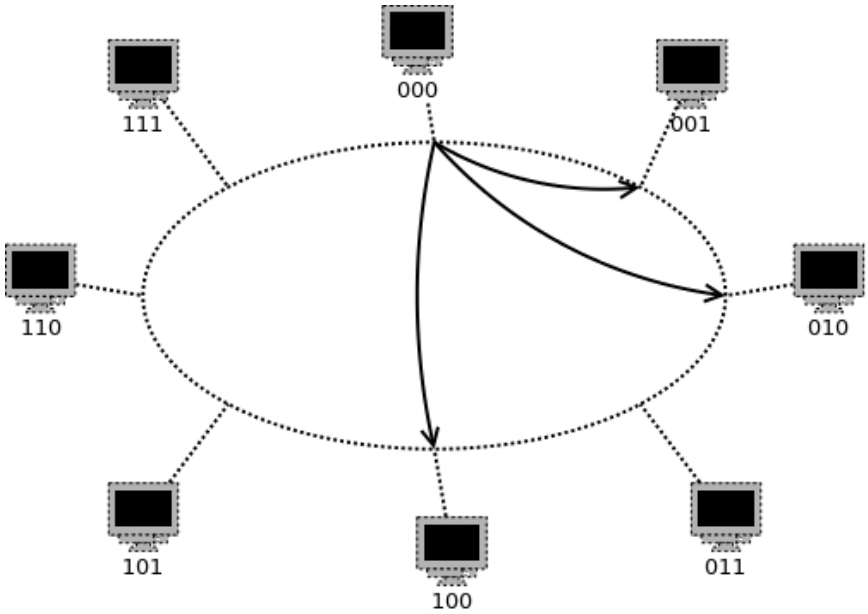

Figure 4: Node 000's finger table, the black arrows represent the links the routing table.

The finger table is maintained by periodic "keepalive" messages, directed at the keys determined by the above function. A keepalive message contains information identifying the sender, such as node ID and address. After every period of time *m*, a keepalive message is directed to each key in the finger table. The node receiving a keepalive message is expected to return a similar keepalive message, indicating it is alive and providing its network address and node ID. This is a slight modification of the Chord stabilisation algorithm. Keepalive messages serve as an explicit notification to a node that it is being indexed by another node; these messages thus replaces Chord's "stabilize" and "notify" procedures (Stoica, *et al.*, 2001, p. 7).

Keepalive messages are digitally signed, and contain a sequence number that monotonically increases with every message sent and received. Certificates for the finger table nodes are retrieved and stored with the node's information and each keepalive message is verified for integrity and authenticity. This prevents against integrity faults, inauthentic messages and replay attacks.

Keepalive messages are never routed directly to the destination node, but rather through a different node in the finger table each period *m*. Only the first hop is changed, and thus the message is still delivered in O(log *n*) hops. This ensures that keepalive messages travel multiple distinct paths to their destination. This secures against omission and link faults, as well as incorrectness faults (Castro, *et al.*, 2002; Wallach, 2003). Nodes along the path to the destination cannot omit every keepalive message or impersonate the destination by responding to every keepalive request when a closer node exists. As the message is routed through a different set of nodes each period *m*, the faulty node or link will eventually be circumvented, the actual finger table node will be notified, and will respond normally.

If a keepalive response is received from a node less close to the ideal node ID than the node already in the finger table, the original node is sent a keepalive message. If the original node responds, the less close node is deemed faulty and not stored in the finger table. If the original node does not respond, it has failed, and the less close node replaces the original. If a keepalive response is received from a node closer to the ideal node ID than the node already in the finger table, the closer node's certificate is retrieved. The keepalive message is verified for authenticity and integrity. If valid, the closer node replaces the original.

Nodes wishing to join the network contact any existing "entry" node. New nodes use their entry node to publish a certificate and begin sending keepalive messages, populating their finger table. New nodes then alert other nodes of their presence. The finger table function is reversed and unsolicited keepalive response messages are sent to nodes whose finger tables should contain the new node:

reverseKeepalive[i] = lookup((nodeID – 2^(i-1)) mod keyspaceSize)

As suggested by Stoica, *et al.* (2001), node joins should not aggressively modify the network. Rather, some nodes should be notified of the new node's presence, and the keepalive messages will stabilise the network over time. As such, the unsolicited keepalive responses do not require responses, and some may fail; as long as some nodes know of the new node's presence, future keepalive messages will stabilise the network.

3.2.5. Chord optimisations

In order to achieve performance gains, Bi-Chord (Jiang, Pan, Liang, & Wang, 2005) is recommended. In Chord, messages are only forwarded in a single direction around the keyspace; messages to nearby but preceding nodes are required to traverse the entire keyspace clockwise, rather than take the shorter route anti-clockwise. Bi-Chord increases the efficiency of routing by allowing messages to traverse the ring in either direction, while not sacrificing the simplicity of Chord. The performance gains achieved by Bi-Chord have been empirically proven (Jiang, *et al.*, 2005; Zheng & Oleshchuk, 2009).

The addition of bi-directional edges to the Chord overlay adds an additional O(log^2 *n*) finger table to each node (Jiang, *et al.*, 2005). While this creates an additional storage overhead, the additional links provide increased resiliency, requiring more links to fail before network partitions occur. Additionally, the reverse links significantly reduce the effect of inefficient routing techniques implemented by faulty nodes (Xuan, Chellappan, & Krishnamoorthy, 2003).

As a node sends heartbeat keepalive messages to all nodes in its finger table, nodes have knowledge of nodes that index them on the network. By storing the node ID and address of these heartbeat messages, Chord finger table links become bi-directional.

To maintain the security provided by the strict structure of the Chord-based DHT, before entering a node into the reverse finger table, the node receiving the keepalive message should use the sending node's node ID to calculate a portion of its finger table. If the ideal node ID in the keepalive message is not in the sending node's finger table, the sending node is malicious. Similarly to ordinary finger table, the certificate of the sending node can be retrieved, and used to validate the authenticity and integrity of the received keepalive messages.

The routing algorithm remains the same, but can now select nodes in both directions around the network. This reverse finger table should periodically be maintained, removing nodes from which a keepalive messages has not been received in some time. All other Chord operations remain untouched.

3.3. Replication

DHTs provide an API for a scalable distributed database; they do not however, guarantee high availability. When a node disconnects, data may be lost from the network. This is at odds with the design goal of availability. In order to improve the reliability of the system, objects may be replicated at multiple nodes, making the network resilient to single node disconnection (Milojicic, *et al.*, 2003).

Replication also allows the use of majority agreement algorithms, removing single points of authority (Sit & Morris, 2002). These majority agreement algorithms are introduced in the next section.

Two classes of replication algorithms will be considered (Leslie, Davies, & Huffman, 2006):

- Sequential replication
- Replica enumeration

3.3.1. Sequential Replication

Sequential replication algorithms define replica nodes as nodes nearby the home node for a given key. Introduced in DHash Systems (Dabek, *et al.*, 2001), replica nodes are defined as the *r* successor nodes of the home node for an object. Thus a small block of nodes stores any object redundantly.

Beehive (Ramasubramanian & Sirer, 2004b) is a massively efficient replication scheme for use in networks using prefix key-based routing (Kademlia, Pastry, etc.). Beehive defines replica nodes as the nodes with keys closest to the original key, somewhat similar to the DHash approach.

In these schemes, all replica nodes share very similar IDs to the home node and to each other, differing only in the least significant bits. This places them close together on the overlay network and allows requests for an object and its replicas to be reduced to only a single message to the object's home node. The home node then determines the IDs of the replica nodes forwards request. The closeness of replica nodes drastically improves the efficiency of locating all replicas by requiring only a single message to traverse the full O(log *n*) diameter of the network.

The disadvantage of this approach is that individual replicas are not locatable from any node in the network without knowledge of the entire system. Only the home node and the replica nodes know the IDs of each other. The home node for an object is queried, and is expected to forward the request to the replica nodes (Leslie, *et al.*, 2006). A malicious home node may not allow for replica nodes to be located, and locating replica nodes from elsewhere in the overlay becomes a matter of trial-and-error by changing the last few bits of a key, hoping to land a hit on a replica node.

3.3.2. Replica Enumeration

The alternative to sequential replication is the definition of a globally known function for calculating replica keys based on the original key. Waldvogel, Hurley, and Bauer (2003) introduced the technique as “replica enumeration”, defining a function that takes the original key and an integer index and returns a replica key. Replica enumeration is a family of algorithms, rather than a single algorithm. Changing the defined function creates entirely different sets of replica nodes with different relationships and routing characteristics.

An illustrative example is provided by Knežević, Wombacher, and Risse (2009). They propose the creation of replica keys using the same hash function that created the original key. The input to the cryptographic hash function is the original key concatenated with a “Replica Ordinary Number” to serve as a replica index. Using a cryptographic hash function with a low collision rate generates keys in a pseudo-random manner, spreading the replicas around the network, promoting fairness. This approach ensures that replicas are unrelated and possibly far away from one another in the keyspace. However, the unrelated nature of the replica nodes introduces the possibility that messages between replica nodes may be forced to traverse the entire diameter of the network. This significantly decreases the efficiency of inter-replica communication.

The primary goals of these techniques include the ability for any node in the network to locate a replica without consulting other nodes (Waldvogel, *et al.*, 2003). Using a well-known function, the original key and a monotonically increasing integer to index replicas, keys are easily locatable using only knowledge of the original key. No knowledge of the state of the network is required, and no trust is placed in the home node. However this has the disadvantage that replica messages cannot be conglomerated into a single message.

3.3.3. Replication Selection Criteria

Castro, *et al.* (2002) and Wallach (2003) suggest that the use of replication schemes provide inherent routing security in structured overlay networks. The sending of messages over many unique paths increases the likelihood of messages reaching replica nodes. While a faulty node may prevent the delivery of one replicated request, utilising *r* different paths ensures secure delivery. As such, it is suggested that replicas not be placed too close together in the keyspace, to ensure that multiple unique paths are taken.

Keeping replica nodes far apart in the keyspace has the additional advantage of reducing replica collisions. A collision occurs multiple replica keys for an object are close together in the keyspace, and a single node becomes responsible for more than one replica key for the same entry (Leslie, *et al.*, 2006).

Note that the previous concerns are only present in replica enumeration schemes. In sequential replication, replicas are not located separately and only a single request is sent. Additionally, in sequential schemes, collisions are impossible; per definition, replica nodes are the *r* nodes closest to the home node for a specific key, regardless of what their IDs are.

This security consideration, in addition to the two design goals highlighted in the previous discussion yields the following checklist:

- All replicas should be easily locatable using only knowledge of the original key for an entry.
- Replicas should be able to locate each other in a small number of hops for efficiency.
- Replica nodes should not be close together in the keyspace, to reduce collisions.

3.3.4. Replication Selection: Finger placement

In order to find a compromise between the goals of efficiency, ease of location, and collision prevention, a "finger placement" scheme is adopted (Leslie, *et al.*, 2006). Finger placement is a replication enumeration scheme for use on Chord-based overlay networks. Replica nodes are defined as nodes in the finger table of the home node with the ID equal to the original key of the object; regardless of whether such a node exists.

Replica keys are created using the following formula:

replicaKey[RON] = (originalKey + 2^(hashLength – RON)) mod keySpaceSize

Where:

- originalKey is the original key for the object to be replicated.
- RON is an integer value, ranging from 1 to *r*, increasing monotonically increasing by 1, indexing each replica uniquely.

This function is similar to the one used in the generation of Chord's routing tables. In a completely full network, this formula will guarantee that each replica is only a single hop away from at least one other replica. In non-complete networks this single hop is not guaranteed, however with a high probability each replica node will only be a few hops away from other replica nodes (Leslie, *et al.*, 2006). This efficiency is again drastically improved by the bi-directionality of the Bi-Chord implementation selected in the previous section.

Some modifications are required to the indexing of entries. Rather than indexing an entry at the node with the ID closest to the entry's key, the key is put through the above function, and *r* replica keys are produced. The entry is indexed at the home nodes of each of these *r* keys. Similarly every request for an entry is directed to each of these *r* replica keys.

To illustrate, consider the system catering for a maximum of 8 nodes shown in Figure 5. An entry with key 000 is indexed into the network, three replicas are desired. Using the proposed formula, replica keys of 100, 010 and 001 are found, represented by arrows on the diagram.

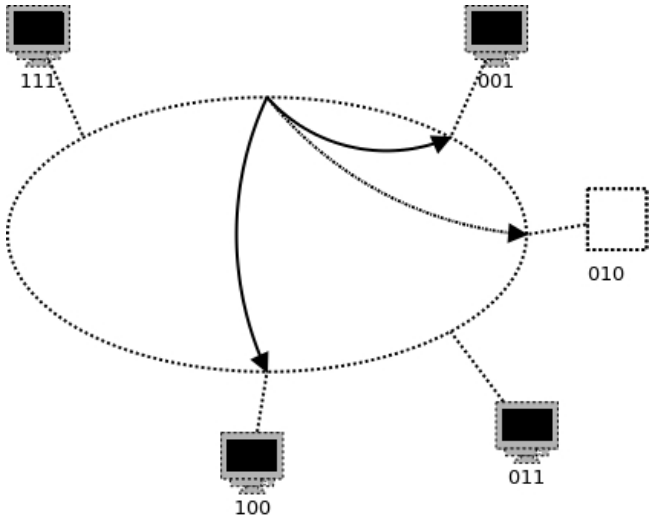

Figure 5: The replica mappings of an object with a key of 000.

It can be seen that the replica mappings for the entry with key 000 mimic the finger table of the node with an ID of 000, as illustrated in Figure 4. Thus replicas of the entry will be stored on node 001, node 011 and node 100. In the bi-directional Chord implementation selected, each replica node is only a small number of hops away from each other replica node (Leslie, *et al.*, 2006).

This technique retains the ability for any node to independently calculate replica keys with no knowledge of the rest of the network. This removes the threat of a malicious home node subverting the protocol and removes the trial-and-error from generating replica keys.

Replica nodes are selected from the far end of the ideal finger table, ensuring that the gaps between replica nodes are large. This reduces the likelihood of collisions, and allows for the use of *r* unique paths around the network, allowing more secure routing (Castro, *et al.*, 2002; Wallach, 2003).

3.3.5. Replica Maintenance

The above section discusses how the model can be made resilient to a small number of failed nodes. However replicas of entries need to be maintained and replaced after failures.

Replica node failures are easy to detect. When a node receives a query which it cannot forward nor fulfil, either the entry does not exist on the network or a replica of that entry has failed. As all replicas are locatable using only knowledge of the original key, the node queries the replica nodes of the queried entry. The query will return either the desired entry, or an indication that the entry does not exist. The node can now index the entry if it exists, stabilising the database in the time taken for a single query. Thus single node failures are detected and corrected, each time a query is made. Entries can only be lost from the network if all replicas fail simultaneously.

Entries that are frequently queried will obtain greater reliability with no modification to the protocol. In order to keep less popular entries reliable, each node periodically makes requests for entries it is responsible for. This will ensure that when node failures occur, the new replica nodes are quickly found, irrespective of the time between normal queries to that entry. This is known as a “replica check”. The more frequent the replica checks, the faster replicas will be replaced and less likely it becomes that all replicas will fail between two replica checks.

Replica checks have other useful features, by having each node responsible for an entry query all other replicas of that entry at regular intervals, the consistency of replicas can be maintained. Each time a replica check is made, replicas that are incorrect or out of sync can be detected and corrected or removed.

Finally, replica checks allow for the gradual re-indexing of entries onto newly joined nodes. This removes the need for a new node to determine a list of keys it is responsible for and make a large number of retrievals from the DHT. Rather, the node listens for requests or replica checks, indexing entries as needed. In this way, nodes that join the network but promptly leave do not cause instability. Entries that do not receive requests or replica checks for long periods of time should be de-indexed; another node has become responsible for the entry and is receiving the messages.

3.3.6. Replica Maintenance Optimisation

The system's availability is thus a function of the number of replicas (*r*), and the time between replica checks (*m*). The larger the number of replicas and less time between replica checks, the less likely it becomes that all replicas fail between checks.

Frequent replica checks allow a much smaller number of replicas to be maintained and still ensure availability. Greater replication creates performance losses, node disk space is used less efficiently and the protocols ensuring consistent database reads and writes discussed in the next sections incur greater overheads when operating on many replicas. It is thus recommended that the number of replicas for each object is kept to a minimum, while the time between replica checks is shortened to compensate.

A detailed discussion on the appropriate selection of replica numbers is provided by Knežević, *et al.* (2009); it is based on the probability of a node failing between replica maintenance.

3.4. Byzantine Agreement

Replication of entries in a DHT with multiple paths to each replica fulfils the design goal of availability. It however introduces the challenge of maintaining the consistency of the database when entries are replicated across distributed and untrusted nodes. It also presents an opportunity to remove the trust in any single node (Sit & Morris, 2002). As requests are issued to untrusted nodes, no assumptions can be made about the correctness of the responses received. Responses may be lost, corrupted, out-dated or incorrect; this may be deliberate malicious activity, or a benign fault. By distributing queries to the DHT across all *r* replica nodes for an entry, no single node needs to be trusted to deliver an honest response.

The problem of reaching a correct response in spite of a minority of Byzantine faults was introduced by Pease, Shostak, and Lamport (1980) and was coined “the Byzantine Generals Problem” (Lamport, *et al.*, 1982). The protocol used to solve the Byzantine Generals Problem is commonly referred to as the “Byzantine agreement”. The classes of “Byzantine” faults the following algorithms are designed to deal with are described in **Section 3.1.2. System Model**.

A Byzantine agreement consists of an interaction between a number of “generals” or nodes making a joint decision. In all cases of the agreement in use in this thesis, the group of generals is the set of replica nodes for a specific entry. The general structure of Byzantine agreements is presented below. The algorithms in use in this thesis are slight variations of the theme introduced here, and are discussed in later sections.

1. To begin an operation, an “initiating” node creates a message. This message is replicated to create a copy for each general. Each replicated copy of the message should be identical, so that all generals operate on the same message.

2. When each general receives a message, it undertakes some message-specific processing and arrives at a "decision". A decision is the outcome based on the processing of the received message at any general. A decision is not acted upon unless a majority of generals reach the same decision. All honest generals perform the same operation on the same message, arriving at the same decision.
3. After a decision is made, both the original message and the decision are broadcast to all other generals. From this point on, each unique message received by a general is diffused to all other generals. Based on the assumption that only a minority of links may fail, rebroadcasting ensures a message will be delivered over a remaining majority of links. Thus every general receives at least one copy of every message.
4. Generals receive up to *r*-1 copies of every message, one from the original general, and *r*-2 from every other general rebroadcasting the message. As additional copies of a message are received, they are checked for consistency with the original copy. If two copies of a message disagree, the certificate of the sender is retrieved and the integrity of both messages is verified. Modified or inauthentic messages are discarded. If a general is found to issue more than a single decision, that general is deemed malicious.
5. Each general waits for a quorum of at least 67% of *r* decisions from other generals before continuing. If all generals have reached the same decision, the protocol terminates, no generals are in disagreement and no special processing is needed. If disagreements occur, one or more generals have suffered from a Byzantine fault and the set of received messages is analysed further.
6. Every message containing a decision differing from the majority or differing from the local decision is marked as suspicious. Certificates of the senders of suspicious messages are retrieved, and the integrity and authenticity of each suspicious message is verified. Modified and inauthentic messages are removed.
7. If any general is found to have sent multiple messages containing conflicting decisions, that general is identified as malicious, and can be removed. The digital signatures provide non-repudiation preventing the malicious general denying their actions.
8. At this point, all faulty or malicious messages have been removed. The remaining responses may be used to make a majority decision. If 67% of *r* messages remain, a majority decision is made. Else the protocol waits for more responses to be received before continuing.

Byzantine agreements only guarantee a decision when $t < n / 3$, when at least 67% of all generals are honest (Lamport, *et al.*, 1982). This is due to the fact that agreements require a quorum of responses over twice as large as the number of possibly faulty nodes to ensure an honest decision (Bracha & Toueg, 1985). This can be proven by considering a group containing a maximum of 33% faulty generals. The decision requires a quorum of 67% of responses; as only 33% of the quorum of responses may be faulty, the remaining 34% the quorum is guaranteed to be honest. As only 33% of generals may faultily withhold their responses, reaching the 67% quorum is also guaranteed. If more than 33% of nodes are faulty, these arguments do not hold. If 34% of generals are faulty, a quorum of 69% is required; as 34% of generals may withhold responses, reaching this quorum cannot be guaranteed. As such, in this thesis, the assumption is made that a maximum of 33% of generals involved in an operation may be faulty simultaneously. This assumption is proven reasonable in **Section 4.6. Attainment of the Honest Majority**.

As rebroadcasting of messages over many distinct links ensures that every honest general receives at least a single copy of every message, proven in the following sections, every honest general works with an identical data set. As every honest general performs an identical function on an identical dataset, every

honest general arrives at an identical decision. As it is assumed that a majority of generals are honest, a majority of generals receive a set of messages containing a majority decision. Thus agreement is reached, despite a minority of faulty participants. Honest generals update their local decision to reflect the majority decision if required.

The use of rebroadcasting for the diffusion of messages makes the Byzantine agreement an expensive operation, requiring O(*g*^3) messages to be sent among generals, where *g* is the number of generals involved in the agreement. As all cases of the agreement in use in this thesis define the group of generals as the set of replica nodes for a specific entry; *g* = *r*. This bound does not account for routing messages.

As the operation is expensive, three precautions are taken:

- The number of replica nodes for an entry is kept small, and replicas are placed only a few hops away from one another, discussed in **Section 3.3.4. Replication Selection: Finger placement** and **Section 3.3.6. Replica Maintenance Optimisation**.
- All implementations of the Byzantine agreement in use in this thesis make use of signed messages, ensuring the integrity and authenticity of messages. This removes the need for a Byzantine agreement on every message.
- The Byzantine agreement is reserved for faulty behaviour or when consistency is unobtainable in another manner. Thus the agreement is not employed unnecessarily. If faulty behaviour is the exception, not the norm, protocols will tend to terminate without requiring any rebroadcasting.

3.4.1. Evaluation of Byzantine Agreement

As the Byzantine agreement is in use throughout the rest of this paper, it shall be evaluated here, demonstrating its usefulness. The agreement is used to make majority decisions that allow the model to operate, in spite of the presence of a minority of nodes experiencing Byzantine faults. Byzantine faults were defined in **Section 3.1.2. System Model**. Each of these faults is discussed in the following sections, as well as an explanation of how the Byzantine agreement achieves resilience to each fault.

3.4.1.1. Omission Faults

An omission fault refers to a case where a general ceases to respond to requests, or responds only intermittently. This may be benign, caused by the general failing or disconnecting, voluntarily or not. It may also be malicious, where a general deliberately refuses to participate, or only participates selectively. The omission of messages has two potential effects on consensus algorithms:

- It may delay the execution of the protocol as other generals wait for the faulty general to respond.
- It may cause messages forwarded through the faulty general to fail to be delivered.

In order to reach an honest decision, the Byzantine agreement requires a quorum of responses over twice as large as the number of possibly faulty generals (Lamport, *et al.*, 1982; Bracha & Toueg, 1985). Even if the quorum contains the maximum number of faulty responses, a majority agreement still results in an honest response. As proven in the previous section, this is only attainable when 33% or fewer generals are faulty simultaneously. As such, it is assumed that, at maximum, 33% of all generals in an operation are faulty.

Given that 33% of generals may be faulty, a quorum of 67% of generals is required. Even if all 33% of faulty generals experience omission faults simultaneously, this 67% quorum will still be achieved as the remaining 67% of generals are honest and respond normally. This allows the agreement to proceed and eventually terminate. Thus a minority omission faults does not affect the execution of the Byzantine agreement.

A node failing to forward messages due to an omission fault is functionally equivalent to a link fault, discussed subsequently.

3.4.1.2. Link Faults

A link fault refers to the failure of the links connecting generals. This includes network failures due to the use of the efficient but unreliable User Datagram Protocol, or loss of messages caused by a node suffering from an omission fault along the path of the message.

In order to allow for the use of majority agreement, and to allow for the identification of faulty or provably malicious nodes, every unique message received by a general is rebroadcast to every other general. Thus, *r*-1 paths are used for every message, one from the original general, and *r*-2 from every other general rebroadcasting the message.

As generals are placed some distance from each other in the keyspace, due to the finger placement replication scheme, these paths are unlikely to all contain the same link or node. By using multiple distinct paths, messages are thus routed around a minority of faulty nodes or links, and through the majority of honest nodes and links (Castro, *et al.*, 2002; Wallach, 2003).

A general only needs to receive a valid message a single time for it to be included in the agreement. As this can be guaranteed for every honest general, all honest generals receive at least a single copy of every message sent. As all honest generals work with the same set of messages, they will all come to the same decision. Thus the Byzantine agreement is unaffected by a minority of link faults.

3.4.1.3. Integrity Faults

Integrity faults refer to the modification of messages during routing, or the creation of inauthentic messages. Modification of messages may be the result of a benign failure in the links connecting generals, or it may be the result of deliberate malicious action of a node along the forwarding path. The forgery of inauthentic messages is the result of an adversary attempting to impersonate another general.

In network-based systems, the integrity and authenticity of a message can be secured using a digital signature or message authentication mechanism (Rivest, *et al.*, 1978). Properties of digital signatures are discussed in **Section 3.1.1. Definition of Preliminary Concepts**.

During the Byzantine agreement, as each of the *r*-1 copies of each message are received during rebroadcasting, they are compared to the original message received. If any differences are noticed, the message is considered suspicious. The certificates of the senders of these messages are retrieved, the authenticity and integrity of each message is verified, and invalid messages are discarded. As messages are rebroadcast over at least *r*-1 unique paths, a faulty link or node causing the modification will be circumvented; at least one valid message will be delivered to each general. Thus the Byzantine agreement ensures that if a minority of messages are modified during routing, they will be detected and removed.

After a quorum of 67% of *r* responses are received, messages containing a decision disagreeing with the majority decision, or disagreeing with the locally held decision at each general, are marked as suspicious by that general. Similar to the above, the certificates of the senders of suspicious messages are retrieved and the validity of the messages is verified. Thus Byzantine agreements are resilient to a minority of decisions being modified or forged by an adversary.

By limiting integrity and authenticity checks to only suspicious messages, the number of retrieval of certificates is kept low. Certificate retrievals are an O(*r* log *n*) operation, as proven in the next section. As they are performed by every general, they contribute significantly to the running time of agreements.

Limiting retrievals to only the senders of suspicious messages allows the protocol to run more efficiently when few or no generals are faulty. This feasibly allows messages to be modified to the majority decision without detection; as the majority decision is assumed correct, this has no adverse effect on the protocol.

3.4.1.4. Incorrectness

Incorrectness involves two separate issues:

- A general may issue an incorrect decision. This may be caused by a benign corruption of the general's local knowledge, or may be the effect of malicious action, attempting to force an incorrect decision.
- A node may incorrectly believe itself to be a general, and thus attempt to participate in the agreement rather than forward the message to the correct general. This may be caused by a faulty routing table, or it may be a malicious action to gain undue influence over the agreement.

As previously discussed, Byzantine agreements require a quorum of responses over twice as large as the maximum number of possibly faulty nodes (Lamport, *et al.*, 1982; Bracha & Toueg, 1985). By waiting for this number of responses, it is guaranteed that more than half the responses in the quorum are honest. A majority agreement is guaranteed to yield an honest response. Byzantine agreements are thus resilient to a minority of incorrect responses.

In the proposed model, care has been taken to construct a strictly structured DHT, and to ensure that all replica nodes are locatable using only local knowledge. Due to these efforts, every general knows the replica keys for the entry involved in the agreement. Thus during a Byzantine agreement, messages are always routed directly to the actual generals (replica nodes) for an entry. Based on the discussion presented in **Section 3.4.1.2. Link Faults**, at least a single copy of message will reach every general. Thus an adversary cannot prevent an honest general participating in the agreement.

Upon receiving messages from two generals claiming responsibility for the same replica key, the authenticity and integrity of each message is verified by retrieving the certificate of each general. Only the node with the authentic node ID closest to the replica key is considered a general. Messages from the faulty node will simply be ignored. Even if the faulty node continues to participate, eventually acting on the majority decision, future retrievals can detect these fake generals using the same mechanism. Thus a faulty node cannot replace an honest general.

As an honest general cannot be prevented from participating, nor be replaced, the Byzantine agreement always communicates with all honest generals. This makes the agreement resilient to incorrect responders. Thus the agreement is secured against both incorrect responses, and incorrect responders.

3.4.1.5. Message Conflict

Message conflict refers to the situation where a faulty initiating node issues a different message to each general, or where a faulty general issues more than a single decision. This is typically done to cause two subsets of otherwise honest generals to work with different datasets. This causes the generals to disagree, or suspect each other of faulty behaviour. Conflicting messages are correctly signed and otherwise valid, but differ in content. This action is always considered malicious, as no benign corruption can cause an initiating node or general to correctly sign conflicting messages.

During Byzantine agreement, every unique message received by a general is rebroadcast. As proven in **Section 3.4.1.2. Link Faults**, this ensures that every honest general receives at least one copy of every message. Upon receiving one of possibly many conflicting messages, a general rebroadcasts it to every other general, as it is considered unique. As an adversary cannot block this rebroadcasting, every honest general will receive a copy of every other conflicting messages received by other generals.

Based on the discussion in **Section 3.4.1.4. Integrity Faults**, conflicting messages will be checked for integrity and authenticity, and discarded if these are violated. If multiple conflicting messages are correctly signed, then a single source has issued multiple decisions. These signed messages serve as proof of malicious activity, the malicious general cannot deny the sending of correctly signed messages. As all honest nodes receive all conflicting messages, all honest nodes will detect the malicious behaviour.

Thus, the Byzantine agreement is resilient to malicious behaviour. Furthermore, after detecting this behaviour, the conflicting messages may be sent to the replica nodes maintaining the malicious node's certificate. After these messages are validated, the malicious node's certificate may be revoked.

3.4.2. Byzantine Agreement Conclusions

Based on the above discussion, it can be seen that Byzantine agreements allow for correct decisions to be made, even in the presence of a minority of arbitrarily faulty nodes. The algorithm relies on the fact that an adversary is computationally bounded and cannot prevent the delivery of all messages to a general, cannot forge digital signatures, and does not control more than one third of all generals in an agreement.

Byzantine agreements remove the need for trust to be placed in any single party. Thus they may be used as a decentralised alternative to central authorities to maintain security or consistency. Byzantine agreements may handle the secure retrieval from and modification of the name database. Additionally, the Byzantine agreement may be used to replace the centralised public key infrastructure in use in the common approach to decentralised name services. Thus the Byzantine agreement is employed in the following three cases:

- To maintain the correctness of entries as they are retrieved from the distributed database. Secure retrieval is discussed in **Section 3.5. Secure Retrieval of Entries**.
- To maintain the consistency of the distributed database during the publication and modification of entries. Consistency is discussed in **Section 3.6. Consistency**.
- To create a decentralised public key infrastructure, discussed in **Section 3.7. Key Management**.

3.5. Secure Retrieval of Entries

When an entry is retrieved from the DHT, the original key of the desired entry is passed through the finger table replication scheme defined in **Section 3.3.4. Replication Selection: Finger placement**. The replicated request messages are issued to all *r* of the replica nodes for the desired entry.

Each replica node responds with the entry for the corresponding replica key. Additionally, the replica node rebroadcasts the request to all other replica nodes. This is done to deal with omission and link faults (Castro, *et al.*, 2002; Wallach, 2003), to ensure all honest replica nodes receive and respond to the request.

The next two sections detail the specific processing involved in the retrieval of a certificate and a name entry, respectively. Following that, a discussion on the Byzantine agreement as it relates to the retrieval of entries is presented. The use of a full Byzantine agreement is not required for secure retrievals, but serves to identify and correct faulty replica nodes. In systems implementing the proposed model, use of the full Byzantine agreement on retrievals is optional; it results in stricter security, but worse performance.

3.5.1. Retrieving a Certificate

A certificate is retrieved using the node ID of the owner of the certificate as the original key. As each certificate response is received from the replica nodes, the initiating node verifies the following:

- The node ID of the certificate is the cryptographic hash of the public-key in the certificate.
- Using the public-key in each certificate, the digital signature of that certificate is verified.

The node ID of any node is the cryptographic hash of that node's public-key. Both are contained in the certificate. As such, the node ID and the public-key of a certificate are cryptographically linked. To cause an incorrect certificate to be considered valid, an adversary must create a public-key that creates the same node ID when hashed. As it is assumed that a computationally bounded adversary cannot predetermine the output of a hash function, this is impossible. This makes public-keys naturally resilient to Byzantine faults. The remaining checks are used to ensure the remainder of the certificate is equally secure.

Using the public-key in each certificate, the certificate is verified for integrity and authenticity to secure against integrity faults. As the public-key and node ID are securely linked, this ensures that no modification has occurred, and that the certificate has been authored by the node with the requested node ID.

When the initiating node receives a quorum of 67% of *r* valid certificates, they are compared. A maximum of 33% of *r* responses may be faulty, thus 34% of *r* received responses must be correct, and the majority decision is taken as honest. Waiting for agreement allows incorrectness and conflict faults to be detected. These faults are caused by a faulty publisher illegally altering their certificate without performing an update operation, or authoring many differing, but otherwise valid, certificates with the same sequence number. Agreement ensures honest replica nodes respond with the correct certificate, and thus faulty certificates are detected. If multiple valid certificates, signed by the publisher, are present, the publisher is malicious.

The certificate with the highest sequence number, correctly signed by the publisher, and in agreement with the majority of responses is accepted. Thus, certificates retrievals are secure against Byzantine faults.

Revocation messages may be present in the set of responses, discussed in **Section 3.7.6. Revocation of Certificates**. If included, these messages are checked for authenticity and integrity using the accepted certificate. If any of these messages is valid, or if the timestamp indicates that the certificate's time-to-live has expired, the certificate is considered revoked. As rebroadcasting ensures that every honest replica node participates in the operation, an adversary cannot prevent these revocation messages from being returned.

3.5.2. Retrieving a Name Entry

Original keys for name entries are obtained through the cryptographic hash of the desired name. Unfortunately, the content of a name entry, the address, is not bound to its key as in a certificate. This means that a verification of key to address is not possible. Thus external mechanisms are in place.

Rebroadcasting of the original request ensures that all honest replica nodes respond. However if more than a single response is received for any single replica key, an incorrectness fault has occurred and an incorrect node is responding to the request. The certificate of the node claiming to be closest to the replica key is retrieved, using the process described above. If the message from this node is authentic and unmodified, it is accepted as the response for that replica key. If it is invalid, it is discarded. Thus, only responses from the actual replica nodes are accepted and incorrect responders are removed.

When the initiating node receives a quorum of 67% of *r* responses from replica nodes, they are compared based on the publisher ID in each name entry. As only 33% of *r* responses are faulty, this guarantees an honest response and secures against incorrect responses.

Using the publisher ID in the majority of responses, guaranteed to be correct, the publisher's certificate is obtained. This publisher's public-key is then used to verify the authenticity and integrity of each response. Invalid responses are discarded, securing against integrity faults. Conflict faults are detected if multiple, differing, correctly signed name entries are received, each with the same sequence number.

The name entry with the highest sequence number, correctly signed and with a publisher ID in agreement with the majority of responses is accepted. Thus, name retrievals are secure against Byzantine faults.

3.5.3. Replica Node's View of Retrieval

If a unanimous decision is not obtained during retrieval, the initiating node may send the set of signed responses received to every replica node for the entry requested. This triggers a Byzantine agreement. As the initiating node already obtains a correct response to its request, this agreement is used to identify malicious nodes, or to correct incorrect replica nodes. Correcting incorrect replica nodes can also be accomplished by replica check messages, described in **Section 3.3.5. Replica Maintenance**. As such, this operation is not critical for the operation of the proposed model. Its ommision yeilds slightly looser security, as malicious nodes may not be identified, but results in increased performance as the expensive rebroadcasting in the agreement is not performed. The agreement is detailed here for completeness and it is left to future implementations of the proposed model to decide if and when the agreement is used.

During the agreement, the set of responses received from the initiator is rebroadcast to ensure all replica nodes initiate the agreement on the same set of responses. Additionally, every replica sends the response it originally sent to the initiator to every other replica, these are then also rebroadcast.

At this point every replica node receives up to *r* copies of every response, one from the initiator, one from the replica node issuing the response, and *r*-2 as they are diffused by each other replica node. If any of these copies of a single message are in conflict, the certificate of the sender is retrieved, and the messages are checked for integrity and authenticity. Corrupted messages are removed. If two valid messages from the same source are found, the sender is deemed malicious.

Once a quorum of 67% of *r* responses is received, responses disagreeing with the majority response, or with the local response, are marked as suspicious. Suspicious responses are checked for integrity and authenticity and corrupted responses are removed. Finally a majority agreement is performed, and all replica nodes update their local decision to reflect the majority. Including all rebroadcasting, and a maximum of *r* retrievals made by each of *r* replica nodes for the certificate of each other replica node, O((*r*^3 * *c*) + (*r*^3 * log *n*)) messages are sent to identify faulty nodes. Where *c* is the number of hops replicas are away from one another. Based on the finger placement scheme selected in **Section 3.3.4. Replication Selection: Finger Placement**, *c* is a near constant number, less than (log *n*) (Leslie, *et al.*, 2006).

3.6. Consistency

Every entry within the proposed model is stored across *r* replica nodes. Based on the discussion on the retrieval of entries above, at least 67% of *r* replica nodes need to honestly maintain an entry for a retrieval of that entry to succeed. Entries thus need to be published and updated in a manner that takes into account this replication, ensuring that entries are kept consistent across all honest replica nodes.

In the traditional DNS, changes are only made to an authoritative server for each zone. Each caching server below the authoritative server in the hierarchy updates its own database by mimicking these changes. Caching servers periodically query the authoritative server to receive new and updated entries. The central authority ensures the consistency of modifications across all replicated servers.

Peer-to-peer networks cannot rely on a central point for controlling change; all replicas of a name entry are considered authoritative and thus all replicas require publishing and updating simultaneously. If replicas are published or updated individually, the system may be left in an inconsistent state while the modifications propagate, with some nodes returning incorrect values when queried.

If no security measures are in place, it cannot be guaranteed that publish and update messages reach all replica nodes. Faulty nodes may cause Byzantine faults to occur, leaving the system with too few replica nodes, or with replica nodes that are not correct.

Thus, publish and update processing must either be atomic, interruptible and indivisible, or a method of determining a correct answer in spite of an inconsistent database is required.

3.6.1. Atomic Broadcast Protocols

Atomic broadcast protocols (Cristian, *et al.*, 1995) are used to provide consistency in distributed databases. They are defined as providing the following four properties (Défago, *et al.*, 2004):

- Validity: If an honest node broadcasts a command, the command is delivered to all honest nodes involved in the command.
- Uniform Agreement: If an honest node receives and executes a command, all honest nodes receive and execute the same command.
- Uniform Integrity: If a command is broadcast, it is executed once at most.
- Uniform Total Order: Commands are executed in the same order at all involved nodes.

Atomic broadcast protocols share much in common with Byzantine agreements, specifically the rebroadcasting of messages to all involved nodes to ensure all nodes are notified and operate on the same message. As such, problems solvable with one protocol may be solved with an implementation of the other (Kursawe & Shoup, 2002). Thus, much of the discussion presented previously on the Byzantine agreement, including its security and scalability characteristics, are applicable to atomic broadcast protocols.

Original work in this area assumed a reasonable synchronisation of clocks across the network, within a small acceptable deviation. These assumptions are reasonable in an environment where a central time server exists by which clocks can be synchronised by the Network Time Protocol (Mills, 1985; Mills, Delaware, Martin, Burbank, & Kasch, 2010). The restrictions of a pure peer-to-peer network architecture make the use of a central time server infeasible.

Kursawe and Shoup (2002) created an optimistic asynchronous atomic broadcast protocol to overcome this limitation. It has been successfully used in prior attempts at distributed name services (Cachin & Samar, 2004). Their protocol makes no synchronisation assumptions, making it ideal for use on a peer-to-peer network. It works by broadcasting messages to all involved nodes, who each sign and rebroadcast each message. In the case of delays or non-responsive nodes, the protocol attempts to roll back to a safe sequence number or “watermark” and continue from that point. If a majority of peers respond favourably, the command succeeds, otherwise it fails definitively. It has been proven secure against even a powerful adversary in control of the network (Kursawe & Shoup, 2002).

The uniform total order property is useful in situations where reads and writes may occur in any order, and may be dependent on one another. A typical scenario demonstrating the need for ordering is as follows:

Alice reads the value of a variable called “balance”. Bob reads the same value from “balance”.

Alice then adds some value to “balance”. Bob subtracts some value from “balance”.

Alice stores her new value for “balance” to the database, overwriting the previous value.

Bob stores his new value for “balance” to the database, overwriting Alice’s value.

In this scenario, different versions of the replicated database may be stored at each replica node, or updates may be overwritten and lost. These inconsistencies are caused by the fact that records are accessed and written to by multiple parties, and writes may be dependent on prior reads. These issues may be solved enforcing a uniform total ordering on operations, allowing for the use of atomic transactions and locking mechanisms to prevent multiple users accessing the same data simultaneously.

In a name service however, publishing to the database is not dependent on prior reads and only a single party, the publisher, is able to update an entry. The database is thus considered intrinsically atomic at the record level and locking mechanisms are not required. This negates the requirement for total ordering of commands. It is possible to remain consistent with only an indication of which command arrives first.

Update commands may be ordered by sequence numbers. Simultaneous publish requests may be ordered by majority agreement. Replica nodes will refuse any publish requests beyond the first for a specific name, thus the protocol will continue using the entry received first by the majority of replica nodes.

In addition, related to the honest majority assumption, unanimous decision is not required. Only a majority of nodes need execute a command for the system to be left in a consistent state. Finally, the assumption that only a minority of links or nodes may be faulty allows for the implementation of simpler algorithms.

Based on the above assumptions, the optimistic phase of the algorithm proposed by Kursawe and Shoup (2002) is sufficient to reach uniform agreement. Functionally, Kursawe and Shoup’s (2002) optimistic phase is a “Bracha Broadcast” (Bracha & Toueg, 1985). This implementation does however assume reliable links. As a minority of links are assumed to fail, the protocol is modified to include additional rebroadcasting of messages along multiple diverse paths to circumvent faults, as suggested by Castro, *et al.* (2002) and Wallach (2003), and defined by Segall (1983). This more robust, but expensive, version of the algorithm works as follows:

1. The original message is sent from the initiating node to all replica nodes for an entry. A minority of these messages may fail for any reason.
2. Every replica node receiving this message performs some processing based on the message. The local processing performed at each node is dependent on the operation being initiated.
3. If the replica node agrees to complete the operation, it creates a confirmation message. The confirmation message also contains the original message, identifying the operation.
4. This confirmation is broadcast, effectively rebroadcasting the original message along with an indication of willingness to continue the operation. Again a minority may fail for any reason.
5. Upon receiving a unique confirmation message, a replica node stores and rebroadcasts this message. Again a minority may fail for any reason.
6. Each replica node receives the original message through r links, one from the initiator, and r-1 from every other replica node. Similarly each confirmation is received through r-1 links, one from the replica node originating the confirmation, and r-2 from every other replica.

7. Should any of these copies be different from the others, the sender's certificate is retrieved, and they are checked for authenticity and integrity. Inauthentic messages are removed, and any remaining conflicting messages identify the sender as malicious.
8. Once a replica node receives confirmations from a majority of other replicas, it compares all confirmations to ensure all nodes are operating on the same message. If any disagreements are noticed, messages are marked as suspicious and checked for integrity, authenticity, and conflict, as detailed in the previous section on Byzantine Agreement.
9. After all faulty responses are removed, if a majority of confirmations remain, a replica can be sure that all honest replicas are working with the same request. This allows the replica to execute the command knowing a majority of other replica nodes are doing similar.

In the original algorithm defined by Bracha and Toueg (1985), only the initial message is rebroadcast. Thus in order to ensure that all honest nodes operate on the same message, the algorithm included an extra step where nodes waited for "ready" messages from a quorum of nodes more than twice as large as the number of possibly faulty nodes. This ensured that more honest nodes than faulty nodes were included in the quorum. As communications between honest nodes were assumed to be reliable, this ensured that if some honest nodes agreed, all would eventually agree as the decision propagated.

In the modified protocol, as only a minority of links may fail, all rebroadcasted confirmation messages are guaranteed to be received by every honest replica node. Thus if any honest replica node receives a majority of confirmations messages, all honest replica nodes will receive a majority of confirmation messages as they are rebroadcast. This allows the "echo" and "ready" messages in the original algorithm to be compressed into the single "confirmation" message in the modified algorithm.

3.6.2. Epidemic Consistency Algorithms

Demers, *et al.*, (1987) recommend epidemic consistency algorithms for more efficient diffusion of updates in environments where a hierarchical database cannot be established, such as peer-to-peer replicated databases. They state performance benefits over previous deterministic consistency algorithms and the removal of any assumptions or requirements for network quality.

The common theme of these protocols is the pseudo-random selection of another replica node with which the update is shared. Each updated replica node repeats the algorithm at regular intervals, spreading the update at a near exponential rate as more replica nodes are informed, and begin informing others. This approach is robust, should the sharing of an update fail due to a fault, the update will be shared again soon by another replica node. This approach is efficient in comparison to atomic broadcasts; replica nodes are not blocked broadcasting updates to all other replica nodes in order to complete updates. Rather, the update is shared with another node periodically, after which both nodes continue to operate normally.

Epidemic consistency algorithms allow for an update at a single site to propagate the entire database. In a pure peer-to-peer environment, a single site may not be trustworthy; security needs to be enforced through the use of authentication mechanisms such as digital signatures. Before accepting an update, a node may use the certificate of the publisher, and verify the authenticity of the update. The majority agreement required in atomic broadcast protocols is effectively replaced with authentication mechanisms.

Epidemic consistency algorithms have the disadvantage of not propagating updates instantly. In a distributed system, this may lead to inconsistent results when query is made before the update has fully disseminated. Similar to the above solution, nodes retrieving entries may use the publisher's public-key to verify the authenticity and integrity of the updated messages, and accept the newest message.

Should a faulty publisher create multiple update messages, each correctly signed and having the same sequence number, both messages will be accepted as correct by the replicas that receive them. In this manner a faulty publisher may attempt to undermine the stability of the system by having replicas return inconsistent name entries. However, when replicas propagate the change, this inconsistency will be noticed. As each replica has a valid, signed update message and the signatures prove authorship of each message, the publisher can be identified as creating inconsistent update messages.

3.6.3. Consistency Selection Criteria

From the discussion above, it can be observed that the correctness of a message needs to be verified before that message is accepted, either through cryptographic means, or through majority agreement.

When updating, the identity of the publisher of an entry is tied to the private-key used to sign the original entry. The authenticity and integrity, and thus the correctness, of an update are easily verified through the use of digital signatures. Before any update is accepted, the public-key of the publisher of the original entry is obtained. If both the original and the updated entries are correctly signed by the same private-key, it can be ensured that the update is authentic, from the publisher of the original entry, and has not been modified. The same digital signatures prevent conflicting messages from a malicious publisher.

The above solution relies on the fact that an update can be verified through comparison of the key signing the original entry and the key signing the update. When publishing an entry, nothing is known about the entry in advance; publish requests cannot be verified as correct without contacting other replica nodes. As such, an atomic broadcast technique is required to ensure a consistent and secure publishing of entries.

Following this observation, three alternatives present themselves, all offering consistency. The selection between them is thus one of the efficiency of each operation, and the length of the stored entries. These alternatives are summarised below and in Table 3.

- Atomic broadcast protocols are used for both publish and update operations. These algorithms ensure consistency, but are expensive. They do however ensure that all updates are performed simultaneously, simplifying the logic used in the retrievals of entries.
- Updates make use of epidemic consistency algorithms, with the certificate of the publisher being requested at each update and retrieval operation.
- Updates make use of epidemic consistency algorithms, with the certificate of the publisher being stored with the name entry during the original publish operation.

Alternative	Advantages	Disadvantages
Publish: Atomic Broadcast. Update: Atomic Broadcast.	Name entry retrievals do not have to obtain the publisher's public-key to check for updates.	Updates are significantly less efficient, as they require Byzantine agreement.
Publish: Atomic Broadcast. Update: Epidemic Consistency, retrieving public-keys on request.	Name entries do not include the publisher's public-key, and are thus smaller. This improves the efficiency of all operations.	Publisher's certificate is retrieved on every update and retrieval, reducing efficiency.
Publish: Atomic Broadcast. Update: Epidemic Consistency, storing public-keys with entries.	Updates and name entry retrievals already have the publisher's public-key, and do not have to retrieve it.	Increased entry length, the publisher's public-key is included in every entry.

Table 3: Summary of consistency alternatives.

If the network that the proposed model is constructed upon has low latency, the rebroadcasting of messages during atomic broadcast becomes more acceptable. Alternatively if the throughput of the network is high, the increased size of entries is not an issue. For completeness, this thesis details the epidemic consistency algorithm. No assumptions are made as to where the public-key of the publisher is stored, and thus the key is simply "obtained", either from the entry itself, or through an explicit retrieval using the publisher's ID. Three scenarios are considered when modifying the distributed database:

- The publishing of a new name entry or certificate.
- The transference of ownership of a name entry to a new publisher.
- The updating of an existing name entry.

It is assumed that, in name services, publish and transfer operations occur less frequently than update operations, which in turn occur significantly less frequently than retrieval operations (Mockapetris, 1983). As such, maintaining the efficiency of retrieval operations should be given highest priority, followed by update operations, finally followed by publish operations.

3.6.4. Consistent Publishing and Transferring Selection: Atomic Broadcast

As no information on the publisher of an entry is known in advance, publish requests cannot make use of the more efficient epidemic consistency algorithms. As such, publishing employs the modified version of the optimistic phase of the algorithm informed by Bracha and Toueg (1985), Segall (1983), and Kursawe and Shoup (2002), as described above.

To publish a name entry, the publisher selects the desired name and uses the cryptographic hash function on this name to create a key. The publisher then selects the address to which the name is bound. The key, name and address are placed into a message to serve as a name entry, along with a sequence number and the node ID of the publisher. The name entry is digitally signed using a private-key corresponding to the public-key in the publisher's certificate. Finally the name entry is replicated and sent to each of the *r* replica nodes responsible for its storage.

At each replica node, some local processing is done before a confirmation is issued. A uniqueness check is performed, to ensure that the entry has not been published previously. The certificate of the publisher is retrieved using the included publisher node ID, and the authenticity and integrity of the message is verified. Additionally, if the public-key of the publisher is included in the name entry, the replica node ensures that the included public-key is identical to the public-key in the retrieved certificate. If the message is valid, each replica node issues a confirmation message.

These confirmations are rebroadcasted, and the Byzantine agreement continues, checking for message authenticity and integrity, and removing faulty and malicious nodes. Finally, After all faulty nodes are removed, if a quorum of 67% of *r* valid confirmations remain, the entry is published.

While publishing a certificate, the publisher's public-key cannot be verified through a retrieval, as the required certificate is the one being published. In addition, the publisher's ownership of the claimed public-key must be verified. While similar to the atomic broadcast protocol, the process for publishing a certificate is detailed in **Section 3.7.5. Key Management Selection: Byzantine Agreement on Certificates**.

When the ownership of name entries is transferred from one entity to another, the certificate used to sign the entry needs to be replaced. As such, the validity of the update cannot be objectively verified through the use of digital signatures, and requests made before the update is fully propagated cannot be resolved. Thus, name transfers require atomic broadcast to ensure consistency. During the local processing, in addition to the checks done during publishing, each replica node performs verifies the following:

- The transfer command is signed by the original publisher. This is done by obtaining the original publisher's public-key, and verifying the digital signature of the message. This ensures that the transfer operation is initiated by the original publisher and is unmodified.
- The transferred name entry is correctly signed by the new publisher, and contains the correct publisher node ID.

As the atomic broadcast algorithm follows the same approach as the Byzantine agreement, proof of its resiliency against Byzantine faults is presented in **Section 3.4.1. Evaluation of Byzantine Agreement**.

3.6.5 Consistent Updating Selection: Epidemic Consistency

Entries in the proposed model are digitally signed by their publishers. When updating, the publisher of the entry can prove its ownership of an update message by signing the updated entry with the private-key that signed the original entry. The integrity and authenticity of the update can be verified objectively at any single node without the need for a Byzantine agreement. As a computationally bounded adversary cannot forge a digital signature, and the publish operation is secure, correctly signed messages are considered secure, and updates may only originate from the original publisher.

Name entries may have their address updated, while certificates may have their optional identifying information updated. The updated entry must contain a sequence number greater than the sequence number in the original entry. By including a sequence number each entry, the relative age of each entry can be determined, and the entry with the highest sequence number deemed as newest. Use of sequence numbers has the beneficial effect of preventing replay attacks by which an update can be reverted.

The updated entry is shared by the publisher to as many replica nodes as possible. Each replica node compares the updated entry to the original one. If there is no difference, the update is discarded. Otherwise the replica node ensures that only the correct fields are updated, and verifies the sequence number to ensure the updated is newer, and not replayed by an adversary.

Each replica node obtains the publisher's public-key, and the validity of the update is verified. For name entries, the publisher's certificate is obtained using the publisher's ID in the original name entry. The public-key may be obtained from the original name entry itself if it is included. For certificates, the public-key in the original certificate is used. If the update is valid, the original entry is overwritten with the updated one.

After accepting the update, updated replica nodes begin periodically sharing the updated entry with other replica nodes, and the above processing repeats. This ensures that as long as a single honest node receives the original update, all honest nodes will eventually receive it as it is shared.

Epidemic updates are resilient to omission and link faults; the update propagates through *r* unique links as all replica nodes share the update, thus faulty nodes or links will be circumvented. They are resilient against integrity faults, as all update messages are checked for authenticity and integrity using the publisher's public-key before being accepted. They are resilient against incorrectness, as all messages are signed and verified, it is impossible for an incorrect update message to be authored, and the use of sequence numbers prevents replay attacks. Even if an incorrect node shares the update, the update cannot be incorrect without invalidating the digital signature. Finally, they are resilient to message conflict, if a malicious publisher issues two conflicting updates with the same sequence number, this will be detected as the update is propagated amongst replica nodes. Thus epidemic updates are resilient to Byzantine faults.

Section 3.3.5 Replica Maintenance details the need for periodic checks to ensure the consistency of replicas. By weighting these checks towards recently updated entries, both functions can be performed with no increase in overhead. A replica node simply performs the above processing on receipt of a replica check maintenance message; if the replica node does not have an original entry to update, it is retrieved and indexed. Thus both replica maintenance and updates are performed simultaneously.

3.7. Key Management

Throughout the previous sections, it has been assumed that a digital signature mechanism is available, and that certificates are stored in the DHT and may be retrieved as required in order to validate these signatures. Digital signatures provide for authenticity, integrity and non-repudiation in the proposed system. They are created through the encryption of a digest of the sent message, which is attached to the sent message. They verified through the decryption of this digest at the destination.

In order to facilitate the use of digital signatures, a means of managing public-keys is required. It should allow for the secure storage and retrieval of public-keys. To allow the removal of malicious nodes and assist in securing an honest majority, a certificate revocation mechanism should be available. Additionally, to run on the peer-to-peer network, it should require no central point of trust.

3.7.1. Centralised Public Key Infrastructure

In traditional DNS offerings and the common approach to the implementation of a distributed naming system, DNS security extensions (DNSSEC) are used to ensure the integrity and authenticity of messages. DNSSEC (Eastlake, 1999) uses public-key authentication to create digital signatures, identifying senders and preventing the unauthorised modification of messages. Solutions implementing DNSSEC typically rely on a central certification authority to manage the issuing of public-key certificates. The central authority signs each certificate with its private key; certificates may then be verified by checking the digital signature with the authority's well-known public-key. The centralised model for public-key management is simple and efficient, trust in the central authority means that certificates issued by that authority can be verified and trusted. However pure peer-to-peer networks lack the central trust and authority required for this model.

Threshold cryptosystems (Desmedt & Frankel, 1990) are an attempt to allow multiple parties control over the signing of messages. It has seen use in previous distributed name system attempts (Cachin & Samar, 2004). A *K* out of *L* threshold cryptography scheme is a protocol where *K* peers out of *L* trusted peers can perform a cryptographic function, but less than *K* cannot (Desmedt, 1994). It is important to note that some of the *L* trusted peers may be faulty or malicious.

In threshold cryptography schemes, a single private key is created and distributed over *L* peers; certificates may then be signed collaboratively by *K* of these peers. In this manner, responsibility for certificate management is shared by many nodes, in contrast with the centralised approach where the private key is owned by a single entity. While this is superior to a centralised authority, it still requires the concept of a "super peer", or a peer with greater than average authority.

Users of Cachin and Samar's system (2004) are required to trust that the majority of the super peers are honest. Even after the super peers disconnect, their signatures remain valid; the user is required to trust an abstract entity that may no longer exist. As these peers are selected at key creation, the solution does not scale as users join the network. A central entity is required to create and distribute the private-key to the *L* nodes. In order to verify the signature created by the super-peers, a public-key is required, this requires that the public-key be widely known or a central authority made responsible for the management and distribution of certificates. This system cannot be considered decentralised.

3.7.2. Handshaking Key Exchange

An alternative to a public key infrastructure is the exchange of keys during handshaking; seen in Baumgart's distributed name service implementation, P2PNS (2008). During "handshaking", each party provides the other with their public-key during the initial message exchange. This has the advantage of simplicity and independence from centralised services. This approach secures the integrity of messages; by verifying each received message with the received public-key, it is ensured that messages cannot be modified in transit.

Baumgart (2008) attempts to bind node IDs to their corresponding public-keys by enforcing that node IDs be cryptographic hashes of their public-keys. As hashes are irreversible, this approach makes it impossible to select a node to impersonate. However it only creates a weak binding between a node and its public-key.

As no verifiable certificate storage exists, certificate revocation is impossible. Certificate revocation requires the storage of revocation messages, which are then returned with the certificate as it is retrieved. This is not possible in a system where each node supplies its own certificate during handshaking. Thus if malicious activity is detected, the offending node cannot be excluded.

As only the node ID and public-key are cryptographically bound , the correctness of certificates cannot be secured. With the exception of the node ID and public-key, a certificate may be freely edited by its publisher. The publisher may edit the timestamp indicating a certificate's time-to-live, or issue multiple conflicting, but otherwise valid, certificates for the same node ID and sequnce number. As only the publisher maintains the certificate, this cannot be detected.

In **Section 3.2.3.1. Secure Assignment of Node IDs**, it is stated that the slow churning of nodes is required to prevent gradual Eclipse attacks (Singh, *et al.*, 2006) and ensure secure node ID assignment (Awerbuch & Scheideler, 2004). Later in this section, it is suggested that certificates be revoked over time to obtain this slow node churn. This is done using the timestamp in each certificate, which is then used to determine a time-to-live of that certificate. As the timestamps may be edited, and no other means of certificate revocation is available, this periodic churning of nodes is not possible. This prevents secure node ID assignment, and undermines the assumption of an honest majority.

Stronger bindings between node ID and public-key are needed for use in majority-trust based protocols. If malicious activity is detected, it is important to revoke certificates and remove the offending parties, re-enforcing the honest majority. Similarly, to revoke certificates over time, it is important that the time-to-live of a certificate is not modifiable. To prevent incorrectness and conflict faults, and to secure certificate revocation, the publisher of the certificate should not have sole authority over its modification. This requires the online storage and maintenance of certificates. Once again, the problem returns to the storage, maintenance, and distribution of public-keys in a Public Key Infrastructure (PKI).

This thesis requires a decentralised alternative to the centralised PKI. Two alternatives for online distributed key management are identified from a review of the literature:

- Web-of-trust.
- Byzantine agreement on certificates.

3.7.3. Web-of-trust

The web-of-trust is a model for the propagation of public-keys, first introduced by Zimmerman in his implementation of Pretty Good Privacy (Zimmerman, 2001; originally 1994). The model has been cited as a decentralised alternative to traditional centralised certification authorities.

Web-of-trust models rely on mutual trust between nodes involved in each transaction. This trust can be achieved by three means (Wilson, 1998; Stallings, 2007, pp. 146 – 147):

- Two parties physically exchange keys, each verifying the identity of the other.
- An already trusted peer introduces and endorses a new peer.
- Trust a certification authority to endorse a new peer.
    - Impossible due to the lack of central authority in a pure peer-to-peer network.

Physical exchange of keys provides for absolute trust in the identity of each party, this is how “completely trusted” certificates are entered into each node's database. By authenticating each other through some external channel, both parties are assured of the other's identity. If every node is expected to trust every other node in the network, O(*n*^2) key exchanges are required. Physically exchanging keys when the number of nodes requiring authentication is large becomes unreasonable.

To facilitate scalability, a system of endorsement was introduced. Peers owning a trusted key may vouch for the authenticity of another key, a first-order introduction. If the endorsement is trusted, the new key becomes trusted without the need to physically authenticate that key. Endorsement introduces risks however, not only must a peer trust the endorsing peer, it must also trust that the endorsing peer’s evaluation of the endorsed peer is correct (Wilson, 1998). Each degree of separation between two nodes decreases the trust between parties. First order introductions introduce some risk, second order more so and so on.

To reduce the risk of endorsements, multiple trusted peers may endorse each key, allowing for greater trust within a cluster. As the number of nodes in the system increases, so too does the degree of separation between any two keys, thus introducing increasing the risk of trusting that key. Thus the number of trusted endorsers required increases, to reduce these risks (Wilson, 1998).

Web-of-trust models effectively run on top of the random links created through the exchange of keys between peers. Unstructured networks have been proven inefficient in **Section 3.2.2.1. Unstructured Overlay Networks**. With the structured overlay network in place to ensure scalability, an unstructured PKI becomes a significant bottleneck in the protocol.

These scalability issues are unacceptable in a global network and the increasing risk of fraudulent keys as the distance between nodes increases also jeopardises the security of the system. These issues prompt the investigation of more scalable architecture for a distributed PKI.

3.7.4. Byzantine Agreement on Certificates

Pathak and Iftode (2006) created a model for a public key infrastructure utilising the concept of the Byzantine Agreement (Lamport, *et al.*, 1982). Their work proposed a protocol for authenticating nodes based on proven ownership of a private-key. New nodes create a certificate, aiming to join the trusted group of a "commander" node by having this certificate challenged and accepted by nodes already in the commander node's trusted group. Untrusted nodes prove their ownership of their private-key through a series of challenge-response messages initiated by each already trusted node. The commander requests the results from the trusted group and a majority decision is made. The untrusted node joins the trusted group if it has successfully proven ownership of its private-key. As messages are signed, malicious parties can be identified and removed from the trusted group. This protocol has been proven Byzantine fault tolerant and capable of handling node failures and collusion between malicious nodes.

Takeda, *et al.*, (2010) present a recently attempted Chord-like implementation of a distributed public key infrastructure. Their implementation stores the public-key certificates of a node on *r* nodes preceding and succeeding that node, known as "authenticator nodes". The trusted group for each authenticator node is all other authenticator nodes. Nodes enter the network by having an introducer node, already authenticated on the network, vouch for the new node's identity and public-key. After the introducer node obtains a new node's public-key, it sends the key to the authenticator nodes, which can verify the new node's ownership of the corresponding private-key. Takeda, *et al.* (2010) provide empirical proof of scalability of the system.

Ultimately however, the work of Takeda, *et al.* (2010) lacks precise details on how ownership of a private-key is verified. It is presumably done in the same manner as described by Pathak and Iftode (2006). It is also not stated how malicious nodes are handled. It is reasonable to assume a Byzantine agreement is used.

With these two assumptions, this model allows for the creation of a scalable PKI for use in structured overlay networks. The following section aims to flesh out this model, to create a functioning decentralised alternative to a centralised certification authority. By removing the need for a centralised authority, the shortcomings seen in the common approach to distributed name services can be successfully alleviated.

3.7.5. Key Management Selection: Byzantine Agreement on Certificates

Based on the discussion in the previous work in this thesis, a structured DHT is already in place with the ability to store small chunks of information in a scalable manner. By simply extending this model to allow certificates to be published and retrieved in a nearly identical manner, and relying on Byzantine agreement to verify ownership of certificates, a decentralised PKI can be achieved.

The process for retrieving and updating certificates has been discussed in **Section 3.5. Secure Retrieval of Entries**, and **Section 3.6.5. Consistent Updating Selection: Epidemic Consistency**, respectively. The process for publishing the certificates is similar to the techniques discussed in **Section 3.6.1. Atomic Broadcast Protocols**, with additional processing to prove the publisher's ownership of the certificate.

A public-key set is created and a node ID is obtained by taking the cryptographic hash of the public-key. The node ID and public-key are placed into a message to serve as a certificate, along with a timestamp, sequence number, and any other optional identifying information. This certificate is then digitally signed

using the private-key corresponding to the public-key it contains. The certificate is replicated in the same manner as name entries, and sent to each of the replica nodes responsible for its storage.

Similar to name entries, every honest replica node check for the uniqueness of the certificate, denying the request if the node ID is, or was previously, occupied. The replica nodes ensure that the timestamp is in the past but not already expired, as discussed in the next section on node churn. Finally it is verified that the certificate is correctly formed and correctly signed by the private-key corresponding to the public-key in the certificate itself.

If the certificate is valid, every replica node performs an instance of Pathak and Iftode's (2006) protocol, while serving as a member of the trusted group for each other replica node. Replica nodes issue a challenge encrypted with the publishing node's public-key. The publishing node returns a signed copy of both the encrypted and unencrypted challenge, proving ownership of the claimed private-key to that replica node.

These challenge-response pairs, along with the expected unencrypted challenge, are diffused to the other replica nodes, in a manner identical to atomic broadcasting while publishing names. This serves to ensure that all replica nodes are publishing the same certificate, allows corrupted or inauthentic responses to be removed, and allows a majority agreement to be performed as to the ownership of the private-key.

Each received pair is analysed to ensure that the publisher's response is correctly signed using the certificate being published. The unencrypted challenge supplied by the replica node is encrypted with the publisher's public-key and compared with the challenge the publisher claims to have received. If there are any discrepancies, the certificate of the replica node is retrieved and the authenticity and integrity of the challenges are verified. If both are valid, the replica node issued a different challenge to the publisher than the one included in the challenge-response pair. The replica node is identified as malicious.

After 67% of *r* of these challenge-response pairs have been received, they are analysed for a decision. If the publisher has correctly decrypted the challenge, it has proven ownership of the certificate. If all decisions agree, and the publisher has proven their ownership of the private-key to a majority of replica nodes, the protocol terminates with the certificate being published. If any disagreements are found, each replica node requests the set of signed challenge-response messages received and issued from the publisher.

The two sets of challenge-response messages, one from the other replica nodes and one from the publisher, are compared. Messages with the same apparent source but differing content are checked for authenticity and integrity by retrieving the certificate of the sender. After removing corrupted responses, if conflicting messages remain, the sender of the conflicting messages is identified as malicious.

Messages containing decisions that disagree with the majority decision or with the decision of the current replica node are marked as suspicious. Certificates of the senders of the suspicious messages are retrieved and modified messages are discarded. Any nodes masquerading as replica nodes are also removed by verifying that only the node with the node ID closest to the replica key is acting as a replica node.

After removing all faulty nodes, a majority decision is made. If less than 67% of replica nodes remain in the agreement, the protocol terminates with the certificate being rejected. The publishing node may attempt to publish their certificate again, provided they were not found malicious previously.

As this algorithm follows the same approach as the Byzantine agreement, proof of its resiliency against Byzantine faults is presented in **Section 3.4.1. Evaluation of Byzantine Agreement**. This algorithm additionally includes a challenge-response step. Messages are rebroadcast during this step, and the checks

for integrity, incorrectness and conflict are repeated, to account for the additional messages. Thus, the algorithm remains Byzantine fault tolerant.

3.7.6. Revocation of Certificates

An important feature of most PKI implementation is the ability to blacklist or revoke previously valid certificates. This must be achievable, even when the owner of the certificate does not agree with the revocation. Certificates are typically revoked by the publisher if the certificate becomes compromised.

To voluntarily revoke a certificate, the publisher sends a signed message requesting revocation of the certificate. The authenticity and integrity of this request is verified using the certificate to be revoked. If valid, this signed message is then stored with the certificate, and returned by every honest node upon retrieval of the certificate. This is irreversible, the certificate is permanently revoked. This prevents an adversary from imitating the publisher, using the compromised certificate.

Certificates may also be revoked when malicious activity is detected. Nodes are proven malicious when they issue conflicting messages during operations. Conflicting signed sets of messages may be used as a replacement for a revocation request. This set of signed messages is then returned with the certificate upon request. Nodes initiating the retrieval of a certificate receive this proof of malicious activity, verify the messages using the retrieved certificate, and accept the certificate as revoked. Revoking a malicious node's certificate prevents any further malicious activity from affecting the future operations of the network.

As both revocation requests and proofs of malicious activity are correctly signed by the certificate in question, they may be objectively verified as authentic and unmodified at any single node. As such, these are disseminated by epidemic consistency protocol.

3.8. Node Churn

Awerbuch and Scheideler (2004) state that systems allowing nodes to occupy the same position in the keyspace for indefinite amounts of time are non-survivable. They state that it is possible for an adversary to gradually infiltrate groups, regardless of the difficulty of doing so. These static systems allow an adversary to, through the mass generation of node IDs over an indefinite period of time, place nodes in strategic positions in the keyspace. This allows the adversary to gradually infiltrate finger tables of victim nodes or the sets of *r* replica nodes for an entry and conduct an Eclipse attack (Singh, *et al.*, 2006).

Awerbuch and Scheideler (2004) propose "churning" nodes; constantly excluding them from the overlay, even against their will. This forces every node to periodically create new node IDs, rejoining the network at a different position in the keyspace. This churn constantly shuffles the membership of groups, randomising the membership of honest and faulty nodes. This randomisation of groups is also suggested by Pathak and Iftode (2006), however they simply delete and reconstruct their trusted groups periodically.

This churn is attainable by placing a timestamp in the certificates used to identify nodes in the proposed model, indicating the initial time the certificate is valid from. After the time-to-live of a certificate expires, verified by checking the timestamp, the certificate becomes revoked automatically and the owning node is forced to publish a new certificate and rejoin the overlay network at a different position in the keyspace. Before this expiry occurs, a node should publish a new certificate and transfer the ownership of all name entries they have published to the new private-key. Certificate home nodes ensure the timestamp is in the past, but still valid, before publishing. They do not allow the updating of this timestamp. Thus no entity, not even the certificate's publisher, can edit the timestamp to prevent revocation. Node churn is thus secure.

If the churning occurs at a rate faster than an adversary can force nodes into a group, then any pseudo-randomly selected subset of nodes can be assumed to contain a number of honest and faulty nodes proportionate with the number of honest and faulty nodes in the entire network. As node IDs are generated through the use of a cryptographic hash function, the difficulty of generating the desired ID becomes exponentially difficult due to the has function's one way property (Back, 2002). The sheer difficulty in generating the required node IDs ensures that infiltration of any group is likely to happen at a slow pace, thus the time between certificate expiries can be kept high without much risk.

3.9. Summary of Technologies

The previous sections detailed technologies supporting the creation of a decentralised name service, their alternatives, implementations and optimisations if any. Referring back to Figure 2, a conceptual diagram was presented for the placement of technologies. This diagram is revisited below, placing each selected technology into the framework.

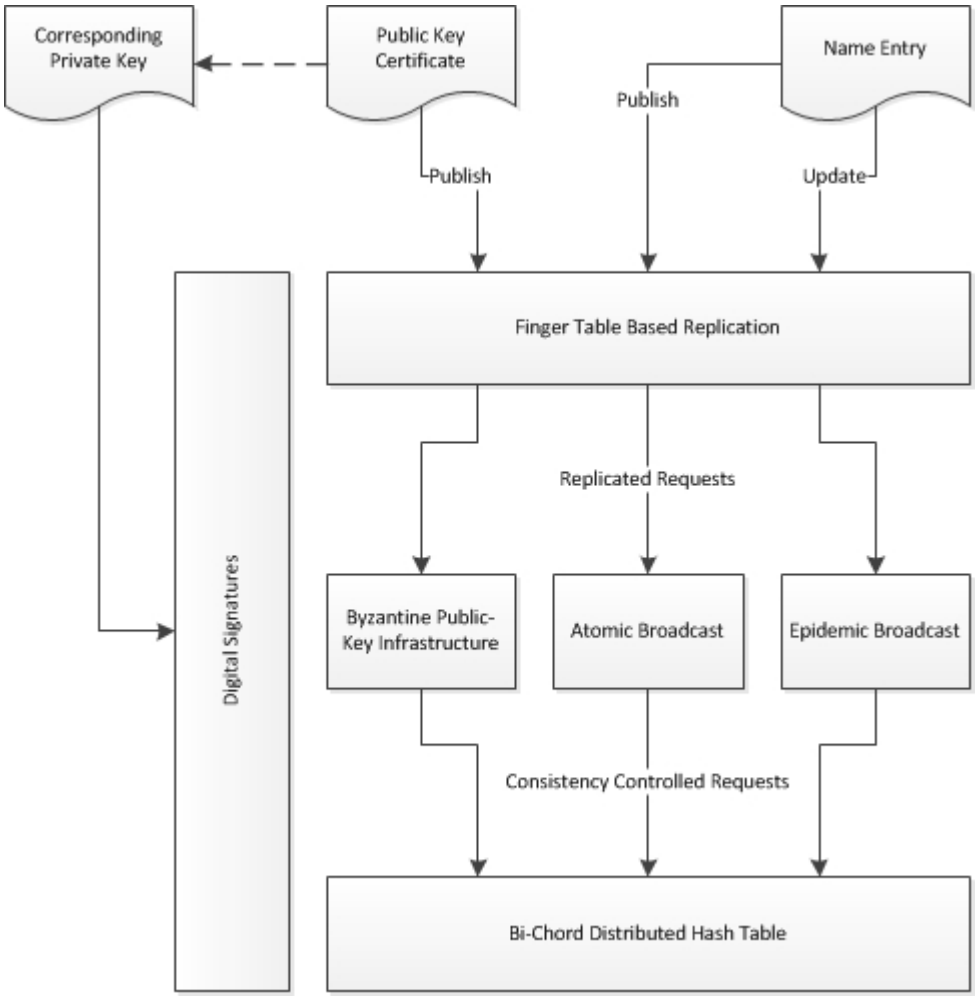

Figure 6: The conceptual model presented in Figure 2, populated with each technology selected to support the proposed model.

Summarising the previous sections and populating the conceptual framework proposed in Figure 2, the following technologies are selected for use in the proposed model for a decentralised name service:

- A scalable and efficient distributed database for the storage of name entries and certificates is established using Bi-Chord:
    - Node IDs are assigned through the cryptographic hash of the node's public-key.
    - Original keys for names are assigned through the cryptographic hach of the name.
    - Original keys for certificates are the same as the node ID of the owning node.
    - The overlay network is maintained through periodic, signed, keepalive messages. These serve both as heartbeats, and as notifications, allowing for the bi-directionality of links.
- Entries are replicated over several nodes for availability and security purposes.
    - Original keys for entries are replicated by placing them through a finger placement algorithm.
    - Entries are stored at home nodes for each of the replica keys, called replica nodes.
    - Messages directed at entries are directed to all replica nodes.
    - Replicas are maintained through the periodic "replica check" messages.
- Byzantine fault tolerant algorithms remove the need for central control of the distributed database.
    - Retrieval of entries is handled in O(*r* log *n*) messages for certificates, based on the fact that node IDs cannot be freely selected, and O(*r*^2 log *n*) messages for name entries.
    - Publishing of entries is performed through an O((*r*^3 * *c*) + (r^3 * log *n*)) message Atomic Broadcast protocol, to ensure consistency of the replicated database.
    - Updates to entries are handled by a more efficient O(*r* log *n*) Epidemic Broadcast protocol, which operates on top of the existing replica check messages.
- The Byzantine Public Key Infrastructure is in place to ensure the validity of certificates, provable through challenge-response messages and provable ownership of private-keys.
    - Certificates are published similarly to name entries, with additional processing to determine ownership of the claimed private-key.
    - Certificates can be revoked either voluntarily by the publisher, involuntarily through proof of malicious behaviour in a Byzantine algorithm, or automatically if a time-to-live expires.
- Using the published certificates, digital signatures are applied to all messages using the public key infrastructure, to ensure authenticity and integrity, and to provide non-repudiation. The digital signature mechanism supports, and it supported by, the rest of the model.

4. Evaluation and Attainment of Goals

Five design goals for decentralised name services were defined and discussed in **Section 2.3. Goals of Decentralised Name Services**; means of measuring these goals were proposed in **Section 2.3.1. Design Evaluation Metrics**. The following section will discuss how the proposed model meets each defined goal, with respect to the supporting technologies discussed above. Finally, although not an explicit goal of decentralised name services, the assumption of the honest majority is discussed.

The proposed model represents a synthesis of several existing technologies, selected through the search process detailed in the previous section. Due to the diversity of the defined design goals and the technologies selected, mathematical analysis is substituted for a more flexible logical reasoning approach. Descriptive and analytical methods of evaluation are employed, informed by existing research to provide rigour, as recommended by Hevner, *et al.* (2004).

4.1. Attainment of Decentralisation

In the above discussion of supporting technologies, many centralised or partially centralised technologies have been identified as *de facto* standards, specifically in the implementation of a public key infrastructure. By taking care to find alternatives to these centralised technologies, decentralisation is obtained.

The selection of Bi-Chord for storage and location of stored entries (Stoica, *et al.*, 2001; Jiang, *et al.*, 2005) ensures a decentralised distribution of entries. Key-based routing and the addition of an appropriate replication scheme based on Chord finger tables (Leslie, *et al.*, 2006) allows all entries to be located, without the need for a centralised tracking server to maintain pointers to content.

The use of Byzantine agreement algorithms to reach majority consensus (Lamport, *et al.*, 1982) removes the need for central control of the database and allows of asynchronous agreements to be reached, negating the need for a centralised time server or a trusted authority.

Finally, replacement of the centralised public key infrastructure with a decentralised alternative based on trust in a majority (Pathak & Iftode, 2006; Takeda, *et al.*, 2010) removes the central authority that plagued the many previous attempts at a decentralised name service, discussed in **Section 2.6.1. The Common Approach to Decentralised Name Services**.

All essential functions of a database of signed name entries (publishing, updating, transferring ownership, and retrieving) are provided without the introduction of a centralised authority or component.

4.2. Attainment of Human-Readability

The proposed model converts names to keys through the use of a cryptographic hash function. From that point, the underlying key-based routing mechanisms work solely with these keys. At no point are these keys exposed to the end user. No restrictions are placed on what names may be contained in name entries, thus allowing the selection of human-readable names in any language, alphabet or syllabary. This allows the publisher to freely select names, allowing names to remain human readable and memorable.

The only restriction on the selection of names is their ability to be input into a cryptographic hash function. Hash functions typically operate successfully on arbitrary binary data, including binary representations of character data such as ASCII and Unicode characters (National Institute of Standards and Technology, 1993; Stallings, 2007, pp. 64 – 65). Any names constructed from Unicode characters can reasonably be expected to be suitable for use in the proposed model. Additionally, any future binary representations of names are expected to be suitable for use in the model with no modifications.

4.3. Attainment of Security

The individual components of security in name services are discussed in **Section 2.3. Goals of Decentralised Name Services**. These general security goals are augmented to be more specific to peer-to-peer systems in **Section 3.2.3. Distributed Hash Table Selection Criteria**, based primarily on the work of Castro, *et al.* (2002) and Wallach (2003). A description of the Byzantine faults that threaten these security goals are identified in **Section 3.1.2. System Model** based on the work around Atomic Broadcast protocols (Cristian, *et al.*, 1995) and the Byzantine agreement (Lamport, *et al.*, 1982).

4.3.2. Uniqueness of Entries

Uniqueness is maintained at each individual node during publish requests. Honest nodes will refuse any publish request making use of a name that already exists within the node's database. As a majority of nodes are assumed to be honest, a majority of duplicate publish requests will fail. In conjunction with the atomic broadcast consistency algorithm (Kursawe & Shoup, 2002), a majority of nodes will not agree to publish a duplicate name entry, and the operation will fail.

4.3.3. Updates from Publisher only

All entries are digitally signed by their publisher. As entities are identified through the ownership of a private-key, a publisher may prove their identity by creating a digital signature using the same private-key that created the signature for the entry in question.

During updates, nodes storing entries ensure that the updated entry and the original entry are correctly signed using the same private-key. Only after this verification is the update accepted. As certificates are stored in the distributed database, signatures can be verified objectively by retrieving the sender's certificate and decrypting the digital signature. With the assumption that the digital signature scheme is sufficiently hard, an adversary cannot create a digital signature for an update message. With the addition of sequence numbers, updates cannot be reversed by replaying previous update messages.

Thus, no party other than the publisher can create a valid update message, and by doing so, cause an unauthorised update to be accepted. Updates are thus secure.

4.3.4. Correctness of Entries

An entry is deemed correct for its key if it is the first entry published for that key, modified only by the publisher. Correctness is maintained through Byzantine agreement protocols, allowing a decision to be made in the presence of faulty nodes (Lamport, *et al.*, 1982).

Entries are published using a modified version of the Atomic Broadcast protocol (Cristian, *et al.*, 1995; Kursawe & Shoup, 2002). Updates are made consistent through the use of the more efficient, but still cryptographically secure, Epidemic Consistency algorithms (Demers, *et al.*, 1987). As such, it is unlikely that any non-malicious node maintains any incorrect entries, however incorrectness is not immediately deemed malicious. It is feasible that an entry become corrupted through benign faults.

To cope with incorrect entries during retrievals, majority agreements are performed should any of the returned entries disagree with others. As a quorum of 67% of r responses are required, and a maximum of 33% of r nodes are assumed to be faulty, 34% of r responses must be honest. The node initiating the request may thus perform a majority agreement to obtain the correct entry.

Additionally, certificate retrievals are naturally resilient to incorrect responses. A computationally bounded adversary cannot forge a certificate with the same node ID as the requested certificate; this would require the reversal of the cryptograpic hash function, assumed impossible. Thus, in addition to the majority vote,

the correctness of a certificate is further secured by verifying the digital signature of the certificate, and checking that the node ID for the certificate owner is the cryptographic hash of the public-key.

In short, entries are securely published and securely retrieved using majority agreement. Proof that these algorithms are secure against Byzantine faults is presented in **Section 3.4.1. Evaluation of Byzantine Agreement**. Entries are securely updated, using digital signatures to maintain correctness. Proof that this algorithm is secure is presented in **Section 3.6.5 Consistent Updating Selection: Epidemic Consistency**.

4.3.5. Secure Node ID Assignment

Every node forming a part of the overlay network maintains a finger table of O(log *n*) degree; these finger table nodes are located using their node IDs. Every operation modifying an entry in the distributed database is required to be executed upon a quorum of 67% of the *r* replica nodes for that entry; these replica nodes are also determined by their node ID.

Because of this, node IDs can be said to determine "membership" to the above two groups. In this thesis, it has been assumed that every group has membership characteristics in proportion to the characteristics of the entire population of *n* nodes. That is, every group consists of at least 67% honest nodes. Allowing an adversary to select node IDs at will, and thus affect group membership, violates this assumption (Castro, *et al.*, 2002; Wallach, 2003). This is defined in literature as the Eclipse attack (Singh, *et al.*, 2006).

To secure against this, node IDs are assigned using a cryptographic hash function. Due to the properties of cryptographic hash functions defined in **Section 3.1.2. System Model**, a cryptographically bounded adversary cannot predetermine the output of a cryptographic hash function. As such, they cannot select node IDs freely. To create any specific node ID, an adversary is required to generate O(2^*z*) separate node IDs, where *z* is the number of bits in the node ID (Back, 2002).

Despite the exponentially large amount of time it may take to generate node IDs that allow an adversary to infiltrate a group, long lived systems that allow nodes to occupy the same position in the keyspace for indefinite periods of time become non-survivable (Awerbuch & Scheideler, 2004). Over extended periods of time, an adversary may slowly generate the required node IDs through trial and error. The proposed model implements Awerbuch and Scheideler's (2004) suggestion of "churning" nodes, removing the node and forcing it to rejoin at a different position. Churning of nodes is discussed in **Section 3.8. Node Churn**. Provided nodes are churned faster than node IDs can be selected, node ID assignment is secured.

4.3.6. Secure Routing Table Maintenance

Routing tables may become corrupted due to manipulation of flexible routing tables, or corruption of the messages used in routing table maintenance (Castro, et al., 2002; Singh, et al., 2006; Sit & Morris, 2002).

Use of optimisation techniques that modify group membership based on proximity, response time or similar metric is a recognised vulnerability to Eclipse attacks (Castro, et al., 2002; Singh, et al., 2006). An adversary may delay honest messages, forge addresses or take similar action to make faulty nodes seem more attractive, allowing larger than average group membership in honest routing tables.

To prevent this corruption of routing metrics, the proposed model makes use of strictly structured routing tables, providing resilience to the Eclipse attack (Castro, *et al.*, 2002; Wallach, 2003; Singh, *et al*., 2006). A single node is defined to process any single message, and as proven in the next section, this can be secured. By not making use of any metric corruptable by an adversary, routing table maintenance is reduced to ownership of node IDs. Based on the discussion in the previous section, this is also proven to be secure.

The only remaining avenue for attack on routing tables is the corruption of the keepalive messages used in routing table maintenance (Sit & Morris, 2002). An adversary may attempt to place itself in routing tables by responding to keepalive messages incorrectly. Keepalive messages may be omitted, corrupted, forged, replayed, or may originate from an incorrect node. Without global knowledge of the network, it cannot be known if a response to a message is from the correct node. These faults are dealt with in the next section.

4.3.7. Secure Message Forwarding

Secure message forwarding involves delivering a message, unaltered by any unauthorised party, from the source to the correct destination. Based on the Byzantine faults defined in **Section 3.1.2. System Model**, this goal is threatened by link faults, link faults created by omission faults along a message's path, integrity faults, or messages being responded to by an incorrect node, rather than the actual destination.

Means of dealing with these faults are discussed in **Section 3.4.1. Evaluation of Byzantine Agreement**, and the respective sections for each operation. Summarising those sections, omission and link faults are dealt with by making use of many distinct paths between any source and any destination (Castro, *et al.*, 2002; Wallach, 2003). During retrievals, publishing and transferring, this is achieved through the rebroadcasting of messages between replica nodes. During updates, it is achieved by the periodic sharing between replica nodes. As the Chord DHT maintains $O((\log n)!)$ paths between any two nodes (Gummadi, *et al.*, 2003), messages may use many unique paths to circumvent a minorty of faulty nodes with high probability.

During publish, transfer and name retrieval operations, integrity faults are detected as r copies of every message are received as they are rebroadcast. Messages from the same source that differ in content are checked for authenticity and integrity, removing invalid messages. The same is done during updates using the public-key of the publisher, obtained using the publisher's ID or the public-key contained in the original entry. Finally, during certificate retrievals, as node IDs and public-keys are cryptographically linked by a hash function, a certificate cannot be forged with the same node ID, but different public-key. As the public-key is thus secure, it may be used to check the the rest of the certificate for authenticity and integrity.

To cope with incorrect responders in any operation, if at any point differing, valid responses are received from multiple nodes, each claiming to be the home node for a specific key, the certificate of the node with the ID closest to the key is retrieved. Using the public-key in the certificate, the authenticity and integrity of the response is verified. If the response is valid, it is accepted as coming from the correct node, else it is discarded and the process is repeated for the response from the next closest node. Certificate retrievals will likely never require this, as multiple valid certificates for a single node ID are assumed impossible to forge.

Where only a single destination exists, agreements or epidemic sharing cannot be used. In these situations, messages are not automatically rebroadcast, and multiple paths are not automatically used. The only single destination messages in the proposed model are the keepalive messages used to maintain routing tables.

Omission and link faults are handled by explicitly making use of a different first hop for each keepalive message. By selecting a different node to forward the keepalive messages to, many distinct paths are used, and thus faulty nodes are circumvented (Castro, *et al.*, 2002; Wallach, 2003).

The issue of corrupted or inauthentic messages is corrected through the retrieval of a node's certificate before placing that node into the finger table. By maintaining the certificate of every node in the finger table, the integrity and authenticity of the keepalive messages can be verified as they are received. A sequence number is maintained, and thus an adversary cannot replay old responses to keep a faulty or disconnected node in a finger table longer than it should be.

Incorrectness faults, caused by a faulty node responding to a keepalive, rather than forwarding it, are also solved through the use of multiple distinct paths. By forwarding keepalive messages along different paths, the faulty node will eventually be circumvented. If a keepalive response is received from a node less close than the expected node stored in the finger table, the original node is sent a keepalive message. If the original node responds, the less close node is deemed faulty and not stored in the finger table. Else the less close node replaces the original node if the keepalive response is correctly signed. Similarly, if a closer node responds to a keepalive message, it replaces the previous node if the response is correctly signed.

4.4. Attainment of Scalability

The use of the Chord DHT, specifically the strictly structured overlay network, allows any node or entry to be located in O(log *n*) messages (Stoica, *et al.*, 2001). Enforcing the bi-directionality of links through the implementation of Bi-Chord has been empirically proven to reduce the length of message paths (Jiang, *et al.*, 2005; Zheng & Oleshchuk, 2009). Bi-directional links additionally provides improved performance in the face of a concerted attack on routing (Xuan, *et al.*, 2003).

These bounds hold in the case of a perfectly honest network. In reality, replication of entries is required to ensure availability and provide security. This replication introduces redundancy, and requires the use of special algorithms to read to or write from this distributed database.

Maintaining consistency while publishing to the replicated database requires the use of expensive Atomic Broadcast (Kursawe & Shoup, 2002) and Byzantine agreement (Lamport, *et al.*, 1982) algorithms. The rebroadcasting involved in these algorithms increase the message count drastically for publish operations and requests that utilise the Byzantine agreement. Thus the following precautions are taken.

Expensive Atomic Broadcasts are avoided where possible. Updates make use of the simpler and more efficient Epidemic Consistency algorithm (Demers, *et al.*, 1987). This algorithm allows eventual consistency as updates spread to each replica while not requiring message rebroadcasting. Digital signatures and sequence numbers ensure that gradual updates do not cause conflicts.

As retrievals are expected to be the most common operation, effort has been taken to improve their efficiency. They do not require full Byzantine agreement with rebroadcasting to ensure a secure response, only a majority agreement. As it is impossible for an adversary to forge a certificate for a specific node ID, certificate requests require no expensive processing to ensure correctness. Name requests, which cannot rely on this assumption, retrieve certificates to ensure secure message forwarding, and thus have higher bounds. However, as certificate retrievals are efficient, these higher bounds remain reasonable.

To facilitate efficient routing between replica nodes, replica keys are chosen based on the same algorithm used to create Chord finger tables. This approach allows replicas to be located from anywhere in the network and places each replica within only a few hops of each other (Leslie, *et al.*, 2006). This effectively reduces routing time between replicas from O(log *n*) to O(*c*), where *c* is a small, nearly constant value.

In order to minimise the negative effects of replication, the value of *r* needs to be chosen carefully. Larger values of *r* lead to poorer performance, but greater availability. A means to increase reliability and availability while not increasing *r* is discussed in the next section on availability.

If all parties involved in an operation behave in an honest manner, the protocols in use terminate quickly. Care is taken to limit more expensive processing to worst-case scenarios. Majority agreement algorithms, requiring multiple rebroadcasts of received messages, are only employed if faulty behaviour is detected. It is assumed that faulty behaviour is the exception, not the normal case. Certificates are only retrieved for messages marked as suspicious for disagreeing with expected results, rather than for every message.

Table 4 presents the maximum number of messages sent to complete each operation supported by the proposed model. These represent the worst-case scenarios, where the maximum 33% of *r* nodes in each operation are faulty, messages are routinely lost, corrupted, incorrect or in conflict, and every message is required to traverse the full log *n* number of hops around the keyspace.

Operation		Total Order
Certificate Retrieval	Initiator: (*r* request messages) * (log *n* routing messages) + (*r* response messages) * (constant checks on responses)	O(*r* * log *n*)
	Replica node: (*r* request messages) * (*r* rebroadcasts) * *c*	O(*r*^2 * *c*)
Name Entry Retrieval	Initiator: (*r* request messages) * (log *n* routing messages) + (*r* response messages) + (*r* certificate requests)	O(*r*^2 * log *n*)
	Replica node: (*r* request messages) * (*r* rebroadcasts) * *c*	O(*r*^2 * *c*)
Retrieval Byzantine agreement	(*r* response messages) + (*r* agreement messages) * (*r* rebroadcasts) + (*r* replica nodes) * (*r* replica pair messages) * (*r* rebroadcasts) * *c* + (*r* replica nodes) * (one certificate request made for each of *r* replicas).	O((*r*^3 * *c*) + (r^3 * log *n*))
Certificate Publish	(*r* request messages) * (log *n* routing messages) + (*r* challenge messages) + (*r* response messages) + (*r* replica nodes) * (*r* proof messages) * (*r* rebroadcasts) * *c* + (*r* replica nodes) * (one certificate request made for each of *r* replicas)	O((*r*^3 * *c*) + (r^3 * log *n*))
Name Entry Publish	(*r* request messages) * (log *n* routing messages) +(*r* replica nodes) * (*r* confirmation messages) * (*r* rebroadcasts) * *c* + (*r* replica nodes) * (one certificate request made for each of *r* replicas)	O((*r*^3 * *c*) + (r^3 * log *n*))
Name Transfer	(*r* request messages) * (log *n* routing messages) + (*r* replica nodes) * (*r* confirmation messages) * (*r* rebroadcasts) * *c* + (*r* replica nodes) * (one certificate request made for each of *r* replicas)	O((*r*^3 * *c*) + (r^3 * log *n*))
Update	(*r* update messages) * (log *n* routing messages) + (publisher's certificate retrieval)	O(*r* * log *n*)
Certificate Revocation	(*r* update messages) * (log *n* routing messages)	O(*r* * log *n*)
Node Join	(log n finger table nodes) * ((1 keepalive message) * (log *n* routing messages) + (1 keepalive response) + (1 certificate request))	O((r+1) * log^2(n))
Keepalive	(1 keepalive message) * (log *n* routing messages) + (1 keepalive response) + (1 certificate request for both nodes)	O(*r* * log *n*)

Table 4: Message complexity for each operation supported by the proposed model.

In reality, it is unlikely that these worst-case bounds will be realised. Additionally, the order represents the total number of messages sent across the entire network. In operation, each replica node will only see 1/*r* of these messages; the remainder are directed to other replica nodes. While publish and transfer operations are expensive, they do not grow in relation to the number of nodes in the network. Rather, they remain static, based on the selected value of *r*. Coupled with the fact that publish operations are expected to be relatively uncommon, these bounds are deemed acceptable.

4.5. Attainment of Availability

The use of the Bi-Chord DHT provides O((log *n*)!) unique paths around the network (Gummadi, *et al.*, 2003). The network is fully *k*-connected, where *k* = O(log *n*) (Loguinov, *et al.*, 2003). Thus, a co-ordinated failure of all O(log *n*) finger table nodes of a victim node is required to create a network partition, a feat assumed nearly impossible in any reasonable time in **Section 4.6. Attainment of the Honest Majority**.

Up to 50% simultaneous node failure is withstood, if every node is equally likely to fail (Loguinov, *et al.*, 2003; Al-Kassimi, 2005). Link failures do not significantly affect the availability of the model; an alternative path is found, utilising alternative nodes. More detailed discussion of the use of multiple paths is provided in the security section above.

The availability and reliability of entries is otherwise entirely controlled by the replication of entries. Larger values of *r* allow a greater number of individual replica nodes to fail simultaneously without an indexed entry becoming lost. Entries are then re-indexed through maintenance messages sent every *m* seconds. Smaller values of *m* create higher overheads as more maintenance messages are sent, but create a smaller window of opportunity for all *r* replicas to fail before entries are reindexed. It is recommended that *m* be kept small, allowing high availability of entries while keeping *r* small to maintain adequate performance for algorithms requiring rebroadcasting.

As node IDs are assigned through a cryptographic hash function, the *r* replica nodes likely share no common detail. As such, in an orchestrated attack, an adversary must organise the failure of *r* geographically dispersed nodes, within a period defined by *m*. If any of the *r* nodes remain connected to issue maintenance messages, or if a request is made for the victim entry during the attack, the attack not only fails, but the victim entry is reindexed on *r* different nodes, requiring the entire attack to be repeated.

4.6. Attainment of the Honest Majority

While not a goal of decentralised name services, the major assumption made in this thesis is that the adversary is computationally bounded, and that no more than 33% of nodes or links may be faulty at any time. The assumption states that the network contains enough honest nodes that a bounded adversary cannot muster enough resources to execute a Sybil attack (Douceur, 2002) and become the majority. However infiltrating a small subset of nodes requires significantly less resources and is reasonable for even a computationally bounded adversary.

Every node forming a part of the overlay network maintains a finger table of O(log *n*) degree. Should an adversary gain control over a majority of these outbound links, the adversary may control the communication of the victim node. Every operation modifying an entry in the distributed database is required to be executed upon a quorum of 67% of the *r* replica nodes for that entry. Should an adversary gain control of a majority of any set of *r* nodes for an entry, the adversary can control operations upon that entry. Thus, two groups are defined:

- The set of O(log *n*) outbound links maintained in the finger table at each node.
- The set of *r* replica nodes for any entry.

Ideally, these groups should share the same faulty node distribution as the entire network, and thus allow the honest majority assumption to apply to any group. However naively making this assumption is not possible. Singh, *et al.* (2006) defined the Eclipse attack, where an adversary attempts to place its nodes into a small and well known group of nodes, such as the ones defined above, as many times as is possible in order to gain control of the group.

Group membership is determined by node ID, **Section 4.3.5. Secure Node ID Assignment** describes how node IDs are securely assigned, and how node churning secures against slow infiltrations. **Section 4.3.6. Secure Routing Table Maintenance** describes how the use of a strictly structured routing table ensures that no corruptible metric is used during group membership. **Section 4.3.7. Secure Message Forwarding** describes how the correct senders of messages can be determined, preventing an adversary from masquerading as a group member.

Based on the above precautions, an adversarial node cannot place itself into a group by selecting an appropriate node ID, move itself into a group by altering some measurable metric, or masquerade as a group member. Thus, it can be assumed that Eclipse attacks are effectively impossible for a computationally bounded adversary, and that any pseudo-randomly selected group of nodes to have the same faulty node distribution as the entire network.

As such, the only means for an adversary to gain a majority membership of a group is to perform a full-scale Sybil attack. A Sybil attack requires resources that a computationally bounded adversary is assumed to not possess. Thus group membership is secured.

5. Comparison of Approaches

As this thesis has strived to demonstrate, both the peer-to-peer network approach and the standard hierarchical approach to name services can achieve human readable names in a secure manner. However, the peer-to-peer approach to security is significantly more complicated due to the lack of any central trust or authority. The two approaches are thus compared based on the goals of performance and availability.

Pappas, Massey, Terzis, and Zhang (2006) provide a detailed discussion of the advantages and disadvantages of both the hierarchical and peer-to-peer network approach. Their study concluded that DHT based solutions are considerably more resilient to orchestrated attacks due to their flat structure and the lack of authority. However their study also criticised this flat structure and the requirement that all nodes must maintain equality, disallowing the “fine-tuning” of the system for performance benefits.

The Pappas, *et al.* (2006) study echoes this thesis's concerns over the reliability of the incumbent DNS. It suggests that peer-to-peer alternatives provide significant availability benefits in the face of orchestrated attacks. This is due to the lack of centralised power that may be the target of these attacks.

While their comparison raises concerns over the performance of the proposed system, the scalability measures employed by the proposed model ensure that while an implementation system may not perform as rapidly as the hierarchical DNS, its performance will not degrade as more users enter the system. The use of similar technologies, specifically distributed hash tables, in prior attempts such as the DDNS (Cox, *et al.*, 2002) and CoDoNS (Ramasubramanian & Sirer, 2004a) and in real-world systems such as Freenet (Clarke, *et al.*, 2010) suggest that these distributed data structures provide acceptable performance.

The Pappas, *et al.* (2006) study focused only on the performance and resilience to attacks of the two classes of systems. The model proposed in this thesis also improves upon the existing DNS in terms of greatly reduced administrative costs, as every honest user of the system donates resources to the model. Additionally, without any point of power or “ownership”, systems based on the proposed model are resilient to the abuse of power or political pressure.

The proposed model provides other minor advantages over the DNS, including a flat name space, rather than mandatory hierarchical naming. This has the benefit of separating functionality and operation; in the DNS, names are hierarchical and separated by periods (“.”), partially revealing the internal structure of the

name database. The proposed model does not have this restriction. Finally, the proposed model provides support for names consisting of any data suitable for input into a cryptographic hash function, this provides intrinsic support for any current and future writing system capable of being represented in a binary format.

5.1. Affected Stakeholders

Three sets of stakeholders are identified for name services:

- Administrative organisations or registrars providing the name service.
- Name owners, who register their domain names with a registrar.
- Name service users, everyday Internet users who depend on name services to translate human-readable names to machine-readable addresses.

The proposed model removes the burden of the growing name database from registrars. Administrative costs of peer-to-peer systems are amortised across every peer, handling the progressively increasing costs of name services. As the model runs on end-user machines, no administrative organisation is required. This is both an advantage and a disadvantage to administrative organisations. Expenses are cut, however control over the name database is lost.

Name owners electing to use systems based on the proposed model gain unquestionable ownership over their domain names. This is in contrast with the DNS, where governmental agencies may remove domain names of sites they disagree with from the database (Holder, 2010; Intellectual Property Act of 2011, 2011). Organisations whose web presence may be encumbered by local or international politics may re-establish themselves.

The proposed model also offers resilience against technical attacks, benefitting all name owners. Published entries remain available, even in the face of orchestrated attacks (Ripeanu, *et al.*, 2002). By preventing both political and technical attacks, both recognised weaknesses of the DNS (Ariyapperuma & Mitchell, 2007), the proposed model offers superior availability of published entries.

Smaller organisations without the funds to purchase domain names may benefit from the availability of free domain names. Additionally, as no restrictions are placed on the format of names, name owners may select names consisting of any alphabet or syllabary.

Users of the proposed models experience an Internet unfettered by politics, and uncontrolled by any organisation using political or technical means to gain undue influence over the name service. This promotes the ideals of free speech and freedom of expression. Users in countries with restrictive governments will likely see the greatest benefits. However users all over the world stand to benefit from the availability of websites that may have been removed due to a single country's laws. This leads to a free and more reliable Internet for all users.

The disadvantage of the proposed model from a user's perspective is the increased latency when performing name lookups, and the requirement that the service be running on their machines, consuming some amount of processing power and bandwidth.

5.2. Interoperability

To promote the widespread adoption of systems based on the proposed model, the cost of switching away from the DNS needs to be kept low. This increases a potential user's perceived ease of use and increases the likelihood of the technology being accepted (Davis, 1989).

The framework proposed by Cox, Muthitacharoen, and Morris (2002) suggested the implementation of a loopback server, running on the client's machine to translate conventional DNS queries into a format suited for the peer-to-peer implementation. This approach also allows flexibility; the server can intercept every request, analyse it, then either forward it to the DNS as normal, or to the system based on the proposed model. This allows the two systems to co-operate, easing transition to the proposed model and allowing users to experience the benefits of both systems. Sites that require extremely robust storage of their name entries may use the proposed model, while other sites may choose the performance of the traditional DNS.

By intercepting ordinary DNS queries and translating them, existing applications and infrastructure can interact with the proposed model with no modifications. This significantly reduces the cost of migration by preserving the transparency of name lookups found in the DNS. The server may additionally listen to the network for incoming messages from the rest of the name service.

Future implementations of the model proposed in this thesis are recommended to follow this approach, but further research is required to determine the most efficient or easiest implementation approach.

6. Conclusions

Three problems are identified with the existing DNS in use on the Internet: the hierarchical nature of the system leads to multiple points of authority that are vulnerable to orchestrated attacks, the administrative costs of maintaining the massive name database are growing, and the centralisation of power over domains allows for abuse of the service. Well-connected peer-to-peer networks have been found to be massively fault tolerant, distribute the costs of maintaining the system across all users of the system, and the lack of centralised authority makes them immune to organisational politics.

Through analysis of name services and peer-to-peer networks, five design goals were defined: Decentralisation, human-readable names, security, scalability and availability.

A model for a peer-to-peer based name service was defined. A bi-directional Chord distributed hash table forms a scalable distributed database in which to store name-address mappings and public-key certificates. A replication algorithm is employed to allow for the deterministic duplication of name-address mappings and certificates. The replication algorithm selected provides a compromise between allowing efficient location of any individual replica node without global network knowledge, and efficient communication between replica nodes.

Entries are retrieved, published and updated through the use of Byzantine fault tolerant protocols. Retrievals query all replicas of an entry and perform a majority agreement algorithm to obtain a correct response. An optimistic atomic broadcast protocol is employed to ensure consistent publishing of name-address mappings. An epidemic consistency protocol provides a more efficient means of maintaining consistency during updates. With the assumption that 67% of all nodes on the network are honest, Byzantine agreement algorithms ensure agreement amongst all replicas of an entry. This protects the system against benign node or link failure, or a strong but computationally bounded adversary.

Finally, continuing from majority-vote type algorithms, a decentralised public key infrastructure is created, allowing for the secure publishing and retrieval of public-key certificates. These certificates provide for the integrity, authenticity and non-repudiation of messages in the system; all peers require a certificate, and all messages are signed using a cryptographically secure digital signature algorithm.

Previous work in the area of decentralised name services neglected the implementation of a distributed public key infrastructure. Previous attempts either made use of a centralised certification authority, or implemented inadequate certificate distribution methods. Other common problems observed in prior literature included the lack of a structured overlay network, leading to unpredictable performance and a lack of scalability. Secure implementations often included a small number of designated super-peers, creating a degree of centralisation that limits the growth of the system and provides a central point of failure or power to be exploited.

The logical model proposed in this thesis achieved the creation of decentralised, human-readable name service which remains secure in the face of a computationally bounded adversary. Byzantine faults are accounted for, and attacks on both the infrastructure of the distributed database and the name service itself are considered and secured against. With a small set of reasonable precautions, the scalability of the system remains reasonable, with routing times growing only logarithmically to the total number of users. The proposed mode promises strong availability of the entries it maintains, dependant on the selection of two variables, defining the amount of redundant storage and the frequency with which these redundant copies are maintained.

6.1. Contributions

The primary contribution of this thesis is a model for a decentralised alternative to the incumbent DNS, maintaining human-readable names, strong security in the face of Byzantine faults, scalability to a global number of nodes, and unparalleled availability. The model is based on a structured peer-to-peer network, making it massively fault tolerant, even in the face on orchestrated attacks (Ripeanu, *et al.*, 2002; Al-Kassimi, 2005), a recognised weakness of the hierarchical DNS (Ariyapperuma & Mitchell, 2007). Administrative costs of running a system based on the proposed model are spread across all users, ensuring the system's resources grow to meet the demands of n increased user base (Coulouris, *et al.*, 2005; Milojicic, *et al*., 2003). As no central power exists, the model is immune to organisational politics.

The decentralised Public Key Infrastructure used to ensure the security of name entries detailed in this thesis is a generalizable model. Should the model prove successful in empirical tests, the decentralised PKI may be implemented in any peer-to-peer application requiring public-key certificates.

The five goals of decentralised name services were defined through an observation of name services and peer-to-peer networks. As such it is agnostic of any specific technologies, and the proposed model itself. This this framework of goals may be applied to any peer-to-peer based name service, and serve as a checklist for future research in this domain.

6.2. Further Research

In order to verify the validity of the logical model proposed in this thesis, the development of a working prototype is required. This prototype would serve as a proof of concept, and would need to be tested for reliability, performance and security against faults and malicious behaviour. The prototype would need to be tested against each class of Byzantine fault defined in **Section 3.1.2. System Model**. This prototype would need to be tested using differing values of the variables defined in this thesis; namely *r*, the number of replica nodes for every entry and *m*, the period of time between maintenance messages. By testing the system and varying these variables, ideal values of *r* and *m* c*a*n be determined.

Much care has been taken in this thesis to maintain the honest majority assumption through secure assignment of node IDs and the removal of provably malicious nodes. However research needs to be conducted, verifying that the assumption of an honest majority of nodes holds in global distributed name services. This research would not only be of benefit to the proposed model, but also many problems solved through the use of the Byzantine agreement (Lamport, *et al.*, 1982).

Should the above prototype be found successful, further efforts are required to implement the system "in the wild". Research needs to be conducted on the interoperability of the proposed model with existing infrastructure. This research should aim to reduce the cost of switching away from the hierarchical DNS to the alternative peer-to-peer name service. Ideally, the outcome of this research should be widespread adoption of the new service, as well as satisfaction of users.

7. References

Al-Kassimi, S. (2005). *Evaluation of Chord: A scalable P2P Lookup Protocol for Internet Applications.* Swedish Institute of Computer Science. Kista: Swedish Institute of Computer Science.

Androutsellis-Theotokis, S., & Spinellis, D. (2004, December). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys, 36*(4), 335-371.

Ariyapperuma, S., & Mitchell, C. J. (2007). Security vulnerabilities in DNS and DNSSEC. *Proceedings of the The Second International Conference on Availability, Reliability and Security* (pp. 335-342). Washington: IEEE Computer Society.

Awerbuch, B., & Scheideler, C. (2004). Robust Distributed Name Service. *Third International Workshop on Peer-to-Peer Systems III* (pp. 237-249). La Jolla: Springerlink.

Back, A. (2002, August). *Hashcash - A Denial of Service Counter-Measure.* Retrieved July 10, 2011, from Hashcash: http://www.hashcash.org/

Baskerville, R., Pries-Heje, J., & Venable, J. (2009). Soft Design Science Methodology. *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology* (pp. 9:1-9:11). Philadelphia: ACM.

Baumgart, I. (2008). P2PNS: A Secure Distributed Name Service for P2PSIP. *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications* (pp. 480-485). Washington: IEEE Computer Society.

Black, P. E. (2005, February 2). *greedy algorithm*. Retrieved May 14, 2011, from National Institute of Standards and Technology, Dictionary of Algorithms and Data Structures: http://xlinux.nist.gov/dads/HTML/greedyalgo.html

Black, P. E. (2009, November 16). *hash table*. Retrieved May 19, 2011, from National Institute of Standards and Technology, Dictionary of Algorithms and Data Structures: http://xlinux.nist.gov/dads/HTML/hashtab.html

Bracha, G., & Toueg, S. (1985, October). Asynchronous consensus and broadcast protocols. *Journal of the ACM, 32*(4), 824-840.

Cachin, C., & Samar, A. (2004). Secure distributed DNS. *2004 International Conference on Dependable Systems and Networks* (pp. 423-432). IEEE Computer Society.

Castro, M., Druschel, P., Ganesh, A., Rowstron, A., & Wallach, D. S. (2002, December). Secure routing for structured peer-to-peer overlay networks. *Proceedings of the 5th symposium on ACM SIGOPS Operating Systems Review, 36*(SI), 299-314.

Chen, J., Ramaswamy, L., & Meka, A. (2007). Message Diffusion in Unstructured Overlay Networks. *Sixth IEEE International Symposium on Network Computing and Applications, 2007. NCA 2007* (pp. 126-133). IEEE Computer Society.

Clarke, I., Sandberg, O., Toseland, M., & Verendel, V. (2010). *Private Communication Through a Network of Trusted Connections: The Dark Freenet*. Retrieved May 19, 2011, from The Freenet Project: http://freenetproject.org/papers.html

Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). *Distributed Systems: Concepts and Design* (4th ed.). Upper Saddle River, New Jersey, United States of America: Addison Wesley.

Cox, R., Muthitacharoen, A., & Morris, R. (2002). Serving DNS Using a Peer-to-Peer Lookup Service. *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (pp. 155-165). London: Springer-Verlag.

Cristian, F., Aghili, H., Strong, R., & Dolev, D. (1995). Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. *Information and Computation, 118*(1), 158-179.

Dabek, F., Brunskill, E., Kaashoes, F. M., Karger, D., Morris, R., Stoica, I., et al. (2001). Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems* (pp. 81-86). Washington: IEEE Computer Society.

Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., & Stoica, I. (2003). Towards a Common API for Structured Peer-to-Peer Overlays. (M. Kaashoek, & I. Stoica, Eds.) *Lecture Notes in Computer Science: Peer-to-Peer Systems II, 2735*, 33-44.

Davis, F. D. (1989, September). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly, 13*(3), 319-340.

Défago, X., Schiper, A., & Urbán, P. (2004, December). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys, 36*(4), 372-421.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., et al. (1987). Epidemic algorithms for replicated database maintenance. *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing* (pp. 1-12). Vancouver: ACM.

Desmedt, Y. G. (1994). Threshold cryptography. *European Transactions on Telecommunications, 5*(4), 449-458.

Desmedt, Y. G., & Frankel, Y. (1990). Threshold Cryptosystems. *roceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology* (pp. 307-315). London: Springer-Verlag.

Deutch, P. L. (1973, December). *RFC 606 - Host Names On-line.* Retrieved May 20, 2011, from ITEF Documents: http://tools.ietf.org/html/rfc606

Diffie, W., & Hellman, M. (1976, November). New directions in cryptography. *IEEE Transactions on Information Theory, 22*(6), 644-654.

Dot-BIT Project. (2011, July 05). *Namecoin Project*. Retrieved July 7, 2011, from Dot-BIT Namecoin Project Wiki: http://dot-bit.org/Main_Page

DotP2P Project. (2010, December 5). *Draft Discussion Paper.* Retrieved May 9, 2011, from Dot-p2p: http://dot-p2p.org/index.php?title=Draft_Discussion_Paper

Douceur, J. R. (2002). The Sybil Attack. *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (pp. 251-260). London: Springer-Verlag.

Eastlake, D. (1999, March). *RFC 2535 - Domain Name System Security Extensions.* Retrieved May 14, 2011, from IETF Documents: http://tools.ietf.org/html/rfc2535

Fabian, B. (2009). Implementing secure P2P-ONS. *Proceedings of the 2009 IEEE international conference on Communications* (pp. 988-992). Dresden: IEEE Press.

Fanning, S., & Fanning, J. (1999). *Napster*. Retrieved May 19, 2011, from Napster: http://www.napster.com

Gillett, S. E., & Kapor, M. (1997). The self-governing Internet: coordination by design. In *Coordinating the Internet* (pp. 3-38). Cambridge, Massachusetts, United States of America: MIT Press.

Goldwasser, S., Micali, S., & Rivest, R. L. (1988, April). A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing, 17*(2), 281-308.

Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., & Stoica, I. (2003). The impact of DHT routing geometry on resilience and proximity. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (pp. 381-394). Karlsruhe: ACM.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004, March). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75-105.

Holder, E. (2010, November 29). *Attorney General Eric Holder Speaks at the Operation in Our Sites II Press Conference.* Retrieved May 23, 2011, from United States Department of Justice: http://www.justice.gov/iso/opa/ag/speeches/2010/ag-speech-101129.html

Internet Architecture Board. (2000, May). *RFC 2826 - IAB Technical Comment on the Unique DNS Root.* Retrieved May 25, 2011, from IETF Documents: http://tools.ietf.org/html/rfc2826

Internet Corperation for Assigned Names and Numbers. (2010, October). *Summary of the Impact of Root Zone Scaling.* Retrieved May 23, 2011, from ICANN | Internet Corperation for Assignment Names and Numbers: http://www.icann.org/en/topics/new-gtlds/summary-of-impact-root-zone-scaling-06oct10-en.pdf

Internet Corporation for Assigned Names and Numbers. (2009, November 16). *Internationalized Domain Names.* Retrieved July 2011, 2011, from ICANN | Internet Corperation for Assignment Names and Numbers: http://www.icann.org/en/topics/idn/

Jiang, J., Pan, R., Liang, C., & Wang, W. (2005). BiChord: An Improved Approach for Lookup Routing in Chord. In J. Eder, H.-M. Haav, A. Kalja, & J. Penjam (Ed.), *Lecture Notes in Computer Science, Advances in Databases and Information Systems. 3631*, pp. 338-348. Berlin: Springer Berlin / Heidelberg.

Kirk, P. (2003). *Gnutella - A Protocol for a Revolution*. Retrieved September 10, 2011, from SourceForge: http://rfc-gnutella.sourceforge.net

Knežević, P., Wombacher, A., & Risse, T. (2009). DHT-Based Self-adapting Replication Protocol for Achieving High Data Availability. In E. a. Damiani, R. Chbeir, & A. Dipanda (Ed.), *Advanced Internet Based Systems and Applications* (pp. 201-210). Berlin: Springer-Verlag.

Kursawe, K., & Shoup, V. (2002). *Optimistic Asynchronous Atomic Broadcast.* Zurich Research Laboratory, IBM Research. Rüschlikon: IBM Research.

Lamport, L., Shostak, R., & Pease, M. (1982, July). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems, 4*(3), 382-401.

Leslie, M., Davies, J., & Huffman, T. (2006). A Comparison of Replication Strategies for Reliable Decentralised Storage. *Journal of Networks, 1*(6), 36-44.

Liu, Z., Chen, G., Yuan, C., Lu, S., & Xu, C.-Z. (2004). Fault Resilience of Structured P2P Systems. *In proceedings of Web Information Systems Engineering* (pp. 736-741). Heidelberg: Springer-Verlag.

Loguinov, D., Kumar, A., Rai, V., & Ganesh, S. (2003). Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (pp. 395-406). Karlsruhe: ACM.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys Tutorials, 7*(2), 72-93.

March, S. T., & Smith, G. F. (1995, December). Design and natural science research on information technology. *Decision Support Systems, 15*(4), 251-266.

Maymounkov, P., & Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *IPTPS '01 Revised Papers from the First International Workshop on Peer-to-Peer Systems* (pp. 53-65). London: Springer-Verlag.

Microsoft. (2006, September 27). *Peer Name Resolution Protocol.* Retrieved May 09, 2011, from Microsoft TechNet: http://technet.microsoft.com/en-us/library/bb726971.aspx

Mills, D. L. (1985, September). *RFC 958 - Network Time Protocol (NTP).* Retrieved July 24, 2011, from IETF Documents: http://tools.ietf.org/html/rfc958

Mills, D. L., Delaware, U., Martin, J., Burbank, J., & Kasch, W. (2010, June). *RFC 5905 - Network Time Protocol Version 4: Protocol and Algorithms Specification.* Retrieved July 24, 2011, from IETF Documents: http://tools.ietf.org/html/rfc5905

Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., et al. (2003). *Peer-to-Peer Computing.* Retrieved May 2003, 2011, from HP Technical Reports: http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.html

Mockapetris, P. (1983a, November). *RFC 882 - Domain names: Concepts and facilities.* Retrieved May 14, 2011, from IETF Documents: http://tools.ietf.org/html/rfc882

Mockapetris, P. (1983b, November). *RFC 883 - Domain names: Implementation specification.* Retrieved May 14, 2011, from IETF Documents: http://tools.ietf.org/html/rfc883

Mockapetris, P., & Dunlap, K. J. (1988, August). Development of the domain name system. *SIGCOMM Computer Communications Review, 18*(4), 123-133.

National Institute of Standards and Technology. (1993, May 11). *Federal Information Processing Standards Publication 180-1.* Retrieved June 2, 2011, from National Institute of Standards and Technology Information Technology Laboratory: http://www.itl.nist.gov/fipspubs/fip180-1.htm

OpenNIC. (2011). *OpenNIC*. Retrieved May 25, 2011, from OpenNIC: http://www.opennicproject.org/

Pappas, V., Massey, D., Terzis, A., & Zhang, L. (2006). A Comparative Study of the DNS Design with DHT-Based Alternatives. *25th IEEE International Conference on Computer Communications. Proceedings* (pp. 1-13). IEEE Computer Scoiety.

Pathak, V., & Iftode, L. (2006, March). Byzantine fault tolerant public key authentication in peer-to-peer systems. *Computer Networks: The International Journal of Computer and Telecommunications Networking - Management in peer-to-peer systems, 50*(4), 579-596.

Pease, M., Shostak, R., & Lamport, L. (1980, April). Reaching Agreement in the Presence of Faults. *Journal of the ACM, 27*(2), 228-234.

Peffers, K., Tuunanen, T., Rothenberger, M., & Chatterjee, S. (2007, December). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems, 24*(3), 45-77.

Plaxton, C. G., Rajaraman, R., & Richa, A. W. (1997). Accessing nearby copies of replicated objects in a distributed environment. *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures* (pp. 311-320). Newport, Rhode Island: ACM.

Postel, J. (1980, August 28). *RFC 768 - User Datagram Protocol.* Retrieved June 2, 2011, from IETF Documents: http://tools.ietf.org/html/rfc768

Postel, J. (1994, March). *RFC 1591 - Domain Name System Structure and Delegation.* Retrieved May 20, 2011, from IETF Documents: http://tools.ietf.org/html/rfc1591

Preventing Real Online Threats to Economic Creativity and Theft of Intellectual Property Act of 2011, S. 968 (112th Congress May 26, 2011).

Qiang, Z., Zheng, Z., & Shu, Y. (2006). P2PDNS: A Free Domain Name System Based on P2P Philosophy. *Canadian Conference on Electrical and Computer Engineering* (pp. 1817-1820). IEEE Computer Society.

Ramasubramanian, V., & Sirer, E. G. (2004a, August). The design and implementation of a next generation name service for the internet. *SIGCOMM Computer Communications Review, 34*(4), 331-342.

Ramasubramanian, V., & Sirer, E. G. (2004b). Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays. *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1* (pp. 8-8). San Francisco: USENIX Association Berkeley.

Ripeanu, M., Iamnitchi, A., & Foster, I. (2002, January). Mapping the Gnutella Network. *IEEE Internet Computing, 6*(1), 50-57.

Ritter, J. (2001, February). *Why Gnutella Can't Scale. No, Really.* Retrieved June 7, 2011, from Darkridge Security Solutions: http://darkridge.com/~jpr5/doc/gnutella.html

Rivest, R. L., Shamir, A., & Adleman, L. (1978, February). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21*(2), 120-126.

Root Server Technical Operations Assn. (2011, July 11). Retrieved September 16, 2011, from Root Server Technical Operations Assn: http://root-servers.org/

Rowstron, A. I., & Druschel, P. (2001). Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Middleware '01 Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg* (pp. 329-350). London: Springer-Verlag.

Schoder, D., & Fischbach, K. (2003, February). Peer-to-peer prospects. *Commununications of the ACM, 46*(2), 27-29.

Schollmeier, R. (2001). A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. *Proceedings of the First International Conference on Peer-to-Peer Computing* (pp. 101-102). Washington: IEEE Computer Society.

Segall, A. (1983, January). Distributed network protocols. *IEEE Transactions on Information Theory, 29*(1), 23-35.

Shirky, C. (2000, November 24). *What Is P2P ... And What Isn't - O'Reilly Media.* Retrieved May 15, 2011, from O'Reilly Open P2P: http://openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html

Simon, H. A. (1996). *The Sciences of the Artificial* (3rd ed.). Cambridge: MIT Press.

Singh, A., Ngan, T. W., Druschel, P., & Wallach, D. S. (2006). Eclipse Attacks on Overlay Networks: Threats and Defenses. *Proceedings of the 25th IEEE International Conference on Computer Communications* (pp. 1-12). IEEE Computer Society.

Singh, M. (2001). Peering at Peer-to-Peer Computing. *Internet Computing, IEEE, 5*(6), 4-5.

Sit, E., & Morris, R. (2002). Security Considerations for Peer-to-Peer Distributed Hash Tables. *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (pp. 261-269). London: Springer-Verlag.

St. Juste, P., Wolinsky, D. I., Lee, K., Boykin, P. O., & Figueiredo, R. (2010). SocialDNS: A Decentralized Naming Service for Collaborative P2P VPNs. *6th International Conference on Collaborative Computing.*

Stallings, W. (2007). *Network Security Essentials, Applications and Standards* (3rd ed.). Upper Saddle River, New Jersey, United States of America: Pearston Prentice Hall.

Stewart, B. (2000). *DNS History*. Retrieved May 26, 2011, from Living Internet: http://www.livinginternet.com/i/iw_dns_history.htm

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001, August). Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Computer Communication Review, 31*(4), 149-160.

Takeda, A., Nakayama, S., Kitagata, G., Chakroborty, D., Hashimoto, K., & Shiratori, N. (2010). Hash-Based Distributed Public Key Infrastructure for Ubiquitous Environments. *Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems* (pp. 376-383). Washington: IEEE Computer Society.

Tsoumakos, D., & Roussopoulos, N. (2006). Analysis and comparison of P2P search methods. *Proceedings of the 1st international conference on Scalable information systems* (p. article number 25). Hong Kong: ACM.

Unifiedroot. (2011). *Unifiedroot - The Multilingual Internet*. Retrieved May 25, 2011, from Unifiedroot: http://www.unifiedroot.com/

Waldvogel, M., Hurley, P., & Bauer, D. (2003). *Dynamic Replica Management in Distributed Hash Tables.* IBM, Zurich Research Laboratory. Rüschlikon: IBM Research.

Wallach, D. S. (2003). A survey of peer-to-peer security issues. *Proceedings of the 2002 Mext-NSF-JSPS international conference on Software security: theories and systems* (pp. 42-57). Tokyo: Springer-Verlag.

Watts, D. J., & Strogatz, S. H. (1998, June 04). Collective dynamics of `small-world' networks. *Nature, 393*(6684), 440-442.

We Re-Build. (2010, December 28). *DNS - We Re-Build*. Retrieved May 09, 2011, from We Re-Build: http://werebuild.eu/wiki/DNS

Wilcox-O'Hearn, Z. (2003, September 22). *Names: Decentralized, Secure, Human-Meaningful: Choose Two*. Retrieved May 4, 2011, from zooko.com: http://www.zooko.com/distnames.html

Williamson, S., & Nobile, L. (1991, September). *RFC 1261 - Transition of Nic Services.* Retrieved May 20, 2011, from IETF Documents: http://tools.ietf.org/html/rfc1261

Wilson, S. (1998). Some limitations of web of trust models. *Information Management & Computer Security, 6*(5), 218-220.

Xuan, D., Chellappan, S., & Krishnamoorthy, M. (2003). RChord: An Enhanced Chord System Resilient to Routing Attacks. *Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing* (pp. 253-260). Washington: IEEE Computer Society.

Zahn, T., & Schiller, J. (2005). MAPNaS: A Lightweight, Locality-Aware Peer-to-Peer Based Name Service for MANETs. *Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary* (pp. 499-500). Washington: IEEE Computer Society.

Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., & Kubiatowicz, J. D. (2004, January). Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on, 22*(1), 41-53.

Zheng, X., & Oleshchuk, V. (2009). Improving Chord Lookup Protocol for P2PSIP-Based Communication Systems. *Proceedings of the 2009 International Conference on New Trends in Information and Service Science* (pp. 1309-1314). Washington: IEEE Computer Society.

Zimmerman, P. (2001). *PGP 7.0 User's Guide in English.* Retrieved May 21, 2011, from The International PGP Home Page: http://www.pgpi.org/doc/guide/

Appendix A: Operations of the Proposed Model

The following sections provide a detailed breakdown of the processing performed at each node during each operation. This is intended to guide the implementation of a system, based on the proposed model.

Each operation is performed upon a specific, single entry (either a name entry or a certificate), and involves an "Initiating" or "Publishing" node, and one or more "Replica" nodes. Each set of processing in an operation may be repeated as new messages arrive. This repetition is indicated by the headings for each set of processing; every time the predicate in the heading is true, the processing is repeated.

A.1. Notation

For the following operation descriptions, items in *italicised* text refer to specific messages within the proposed model. Unless otherwise specified, all *italicised* items within the same operation refer to the same message or an identical copy of that message. In cases where two identically names messages are not identical, one of the messages is faulty.

The exception to this rule is that messages named "*Replica-X*" will be named "*OtherReplica-X*" when received by a replica node other than the sender. In this case, "*Replica-X*" always refers to a message belonging or originating at the replica node currently performing the processing. Additionally messages may be named "*New-X*" or "*Old-X*", replacing "*X*". These differences should be noticeable by context.

The notation "{*Message*}*PrivateKey*" is used to indicate that *Message* is digitally signed using the private-key *PrivateKey*. Receiving node will decode with signature with the corresponding *PublicKey*.

The notation *Message[]* is used to represent an array of *Message* messages.

The function "hash(*Object*)" is used to represent the application of a cryptographic hash function on *Object*.

With the exception of terms referring to another technology, such as *FingerTable*, the terminology used below is not taken from other work. Variable names are selected to be descriptive, and easy to understand.

A.2. Key-Based Routing

A.2.1. Purpose of Operation

Messages are typically not delivered directly to their destination. Messages are delivered by recursively forwarding a message to a node closer to its destination. As such, nodes receive requests that need not be processed, only forwarded. All messages have a key, corresponding to a node, name entry or certificate, this key is examined at every node, and forwarded if that node is not the home node for that key.

A.2.2. Context of Operation

Whenever any operation detailed below mentions "sending" a message, Key-Based Routing is employed to ensure message delivery. The operations detailed below omit this processing, it is assumed that all forwarding is complete, and the receiving node is the final destination node.

Alternatively if the message is "directly" sent, the sending node knows the IP address of the destination node through prior communication. Every message in the system is assumed to contain the node ID and IP address of the sending node. Thus after the first message to a destination is routed via key-based routing, subsequent communication may make use of direct message sending.

A.2.3. Finger Table

The finger table is a local database of node IDs and corresponding information for use in routing. The finger table contains roughly O(log *n*) entries has the following format:

Index	Ideal Node ID	Node ID	Node IP Address	Node Public-key	Sequence Number	Time
1	Node ID + 1					
2	Node ID + 2					
3	Node ID + 4					
i	(Node ID + 2^(i-1)) mod keySpaceSize					

Where:

- Index is an integer value running from one to the length of the output of the cryptographic hash function selected to create node IDs.
- Ideal Node ID stores to the output from the finger table calculation formula. It is node ID of the ideal finger table node, if the network was completely full. Keepalive messages are routed to this node ID.
- Node ID is the node ID of the node which responds to the keepalive message sent to the Ideal Node ID. This should be close to, but greater than, the Ideal Node ID.
- Node IP Address is the IP address of the node which owns the Node ID.
- Node Public-key is the public-key of the node which owns the Node ID. Upon receiving a keepalive response from a node different to the one currently indexed in the finger table, a request for that node's certificate is made, the validity of the keepalive response is checked, and the key is stored in the finger table to verify all future keepalive messages.
- Sequence number is a monotonically increasing integer, included with every message sent to and received from the node which owns the Node ID. When sending any message the Sequence Number is retrieved, incremented by one and included in the message. Similarly, all received messages have this number checked. If the sequence number is greater than the currently stored number, the response is valid and the new sequence number overwrites the old one.
- Time is the local Unix timestamp, taken at the time the last message from the node which owns the Node ID is received. Finger table entries are discarded when no communication is received from the node which owns the Node ID for $X * m$, where X is an integer value defined at system creation.

In addition to the finger table defined by Chord shown above, two additional finger tables are kept:

- A finger table populated by nodes from which keepalive messages are received. The Ideal Node ID field is blank, and keepalive messages are only issued in response to received keepalive messages. This facilitates Bi-Chord bi-directional links in the overlay network, increasing routing efficiency.
- A small, O(*r*) entry finger table to contain the information of the nodes most recently communicated with. This serves as a cache for node information to increase the efficiency of majority-vote type algorithms. The public-key field is only filled as required, and ideal Node

A.2.4. Message Forwarding: Upon Receiving or Creating any message

1. A message (*Message*) is received or created by a node. Every message will contain a key (*Key*).
2. Using *Key*, the node performs a search on the Node IDs in its finger table for any node with an ID closer to *Key* than its own node ID.
    - Closeness is defined as the positive integer value found for (*Key* – NodeID)
    - Closer nodes are nodes with closeness values smaller than the closeness of current node.
    - The closest node is the node with the lowest closeness value found in the finger table.
3. If a closer Node ID is found, the node forwards the request on to the node with the Node ID closest to the desired key using the corresponding IP address stored in the finger table.
    - If the node expects a response and none is received after a period of time, the request is forwarded again, this time selecting the second best node to forward the message. This process is repeated until a response is received, or until the node gives up.
4. If no closer node exists, the current node processes the request.

A.2.5. Message Forwarding: Upon Receiving Messages from Two Nodes Claiming the Same Key

During the Byzantine agreement type operations in use in the proposed model, replica nodes author and rebroadcast several messages. The security of the model relies on each of these messages only coming from the actual replica node, and not another node masquerading as the replica node.

1. Two or more signed messages ({*Message*}*ReplicaPr*) have been received. Each message containing a different *ReplicaID*. The *ReplicaID* closest to the replica key for the entry the current operation requires is referred to as *ClosestReplicaID*:
    1. Retrieve the certificate (*ClosestReplicaCert*) using *ClosestReplicaID*. Extract *ClosestReplicaPu* from *ClosestReplicaCert*.
    2. Validate {*Message*}*ClosestReplicaPr* using *ClosestReplicaPu*. If the digital signature is invalid, discard {*Message*}*ClosestReplicaPr*. Repeat this algorithm with the next *ClosestReplicaID*.
    3. If the signature is valid, discard all other {*Message*}*ReplicaPr*.

A.3. Maintaining the Overlay Network

A.3.1. Purpose of Operation

The key-based routing mechanism described above operates on the assumption of a well-connected network, where forward progress to a destination is always possible. Key-based routing relies on properly maintained finger tables in order to find the next hop in a message's path. The following processing ensures finger tables are up-to-date and accurate.

Stoica, *et al.*, (2001) discuss using a new node's successor's finger table to use as hints in completing its own finger table. This can provide a significant improvement to the performance of node joins. However this requires that the new node's successor be completely trusted, and this cannot be guaranteed. The more expensive, more secure algorithm is presented here as a tradeoff.

A.3.2. Context of Operation

Based on the discussion of finger tables presented above, each node defines *ForwardFingerTable[]*:

ForwardFingerTable[i] = {*i*, *IdealNodeID*, *NodeID*, *IPAddress*, *PublicKey*, *SeqNum*, *Time*}

Additionally, each node defines *ReverseFingerTable[]*:

ReverseFingerTable[NodeID] = {*NodeID*, *IPAddress*, *PublicKey*, *SeqNum*, *Time*}

Node joins and node disconnects or failures affect the accuracy of other node's finger tables. Node joins require the population of the new node's finger table, and the insertion of the new node into existing finger tables. Similarly, node disconnects require finger table references to that node to be removed.

Preconditions: The initiating node owns a published certificate, or is in the process of publishing.

Postconditions: Every node's finger table contains the information of the nodes logically closest to the ideal finger table nodes. As this is an on-going operation, and finger tables are constantly affected by node joins and disconnects, this may never be fully realised.

A.3.3. New Node: Upon Joining the Overlay Network

1. Determine the IP address of any existing node in the overlay network (*EntryNode*).
2. By forwarding messages to *EntryNode*, using it as a surrogate, perform the processing detailed in **Section A.4. Publishing a Certificate**.
3. For *i* = 0; *i* < *HashLength*; *i*++:

   HashLength is the length in bits of the output of the cryptographic hash function.

   1. Calculate *IdealNodeID* by performing (*NodeID* + 2^(*i*-1)) mod *KeySpaceSize*

      NodeID is the node ID of the current node, determined by its certificate.

      KeySpaceSize is the maximum size of the keyspace, 2^*HashLength*.

   2. Select a sequence number (*SeqNum*), typically zero.
   3. Create the message (*Keepalive*)

      {*IdealNodeID*, "Keepalive" *NodeID*, *IPAddress*, *SeqNum*}

      IPAddress is the IP address of the current node.

   4. Sign *Keepalive* to create {*Keepalive*}*CurrentNodePr*

      CurrentNodePr is the private-key of the new node, corresponding to the public-key in the certificate published by the new node.

   5. *Send* {*Keepalive*}*CurrentNodePr* via *EntryNode* or any other node in *ForwardFingerTable[]*.
4. For *j* = 0; *j* < *HashLength*; *j*++:

   1. Calculate *ReverseNodeID* by performing (*CurrentNodeID* - 2^(*i*-1)) mod *KeySpaceSize*
   2. Select a sequence number *SeqNum*, typically zero.
   3. Create the message (*KeepaliveResponse*)

      {*ReverseNodeID,* "Keepalive Response", *CurrentNodeID*, *CurrentIPAddress*, *SeqNum*}

   4. Sign *KeepaliveResponse* to create {*KeepaliveResponse*}*CurrentNodePr*

5. *Send* {*KeepaliveReponse*}*CurrentNodePr* via any node in *ForwardFingerTable[]*.

A.3.4. Any Node: Every *m* seconds

1. For *i* = 0; *i* < *HashLength*; *i*++:
    1. Select *ForwardFingerTable[i]*, it has the form:

        {*i*, *IdealNodeID*, *NodeID*, *IPAddress*, *PublicKey*, *SeqNum*, *Time*}

    2. Select a sequence number (*NewSeqNum*), *NewSeqNum* = *SeqNum* + 1.
    3. Create the message (*Keepalive*)

        {*IdealNodeID*, "Keepalive" *CurrentNodeID*, *CurrentIPAddress*, *SeqNum*}

    4. Sign *Keepalive* to create {*Keepalive*}*CurrentNodePr*
    5. Select *FirstHopIndex*, *FirstHopIndex* = *SeqNum* mod *HashLength*.
    6. Send {*KeepaliveReponse*}*CurrentNodePr* via the node identified in *ForwardFingerTable[FirstHopIndex]*.
2. For every *NodeID* in *ReverseFingerTable[NodeID]*:
    1. Select *ReverseFingerTable[NodeID]*, it has the form:

        {*NodeID*, *IPAddress*, *PublicKey*, *SeqNum*, *Time*}

    2. Check *Time*, if *Time* is older than the threshold value, and a keepalive message has not been received from the node owning *NodeID* in some time, discard *ReverseFingerTable[NodeID]*. This is done to flush inactive entries from *ReverseFingerTable[NodeID]*.

A.3.5. Any Node: Upon Receiving a Keepalive message

1. A {*Keepalive*}*NewNodePr* is received, it should have the form:

   {*IdealNodeID*, "Keepalive" *NewNodeID*, *NewIPAddress*, *NewSeqNum*}*NewNodePr*

2. Check *ReverseFingerTable[NewNodeID]*, it has the form:

   {*OldNodeID*, *OldIPAddress*, *OldNodePu*, *OldSeqNum*, *Time*}

3. If *ReverseFingerTable[NewNodeID]* is null:
   1. Calculate *NewNodeFingerTable[]* by performing (*NewNodeID* + 2^(*i*-1)) mod *KeySpaceSize*

      for *i* < *HashLength*; *i*++.

   2. If *IdealNodeID* is not in *NewNodeFingerTable[]*, the node identified by *NewNodeID* is attempting to place itself into arbitrary finger tables. {*Keepalive*}*NewNodePr* is discarded.
   3. Retrieve certificate (*NewNodeCert*) using *NewNodeID*. Extract *NewNodePu* from *NewNodeCert*.
   4. Validate the {*Keepalive*}*NewNodePr* using *NewNodePu*. If the message is invalid, discard it.
   5. Complete the appropriate fields in *ReverseFingerTable[NewNodeID]* using *KeepaliveReponse*.
   6. Set *Time* to the current Unix timestamp.
4. Else if *ReverseFingerTable[NewNodeID]* is not null:
   1. Verify that *NewSeqNum* > *OldSeqNum*, if not, discard the message.
   2. If *OtherPublicKey* is available, validate {*Keepalive*}*NewNodePr* using *OldNodePu*. If the message is invalid, discard it and terminate.
   1. Otherwise, update *ReverseFingerTable[NewNodeID]*. Update *OldIPAddress* to *NewIPAddress* if required. Set *Time* to the current Unix timestamp. Set *OldSeqNum* to *NewSeqNum* + 1.
5. Select *SeqNum*, *SeqNum* = *NewSeqNum* + 1.
6. Create the message (*KeepaliveResponse*)

   {*NewNodeID*, "Keepalive Response", *CurrentNodeID*, *CurrentIPAddress*, *SeqNum*}

7. Sign *KeepaliveResponse* to create {*KeepaliveResponse*}*CurrentNodePr*.
8. Send {*KeepaliveResponse*}*CurrentNodePr* to *NewIPAddress*.

A.3.6. Any Node: Upon Receiving a Keepalive Response message

1. A {*KeepaliveResponse*}*NewNodePr* is received, it should have the form:

   {*CurrentNodeID*, "Keepalive Response",

   NewNodeID, *NewIPAddress*, *NewSeqNum*}*NewNodePr*

2. Find *i* so that *ForwardFingerTable[i]* has an *IdealNodeID* closest to *NewNodeID*. *ForwardFingerTable[i]* has the form:

   {*i*, *IdealNodeID*, *OldNodeID*, *OldIPAddress*, *OldNodePu*, *OldSeqNum*, *Time*}

3. If *ForwardFingerTable[i]* is null:
   1. Retrieve the certificate (*NewNodeCert*) using *NewNodeID*. Extract *NewNodePu* from *NewNodeCert*.
   2. Validate the {*KeepaliveResponse*}*NewNodePr* using *NewNodePu*. If the message is invalid, discard it and terminate.
   3. Complete the appropriate fields in *ForwardFingerTable[i]*. Set *Time* to the current Unix timestamp. Store *NewNodePu* in *ForwardFingerTable[i]*.
4. Else if *ForwardFingerTable[i]*is not null:
   1. If *NewNodeID* = *OldNodeID*:
      1. Verify that *NewSeqNum* > *OldSeqNum*, if not, discard the message.
      2. Validate the {*KeepaliveResponse*}*NewNodePr* using *OldNodePu* stored in *ForwardFingerTable[i]* . If the message is invalid, discard it and terminate.
      3. Otherwise, update *ForwardFingerTable[i]*. Update *OldIPAddress* to *NewIPAddress* if required. Set *Time* to the current Unix timestamp. Set *OldSeqNum* to *NewSeqNum*.
   2. Else if *NewNodeID* != *OldNodeID*:
      1. If *NewNodeID* is less close to *IdealNodeID* than *OldNodeID*:
         1. Issue a *Keepalive* message, as defined above, directly to *OldIPAddress*. If a response is received, discard the *KeepaliveResponse* from *NewNodeID*.
      2. If *NewNodeID* is closer to *IdealNodeID* than *OldNodeID* , or if a *KeepaliveResponse* message is not received for the above *Keepalive* message:
         1. Retrieve the certificate (*NewNodeCert*) using *NewNodeID*. Extract *NewNodePu* from *NewNodeCert*.
         2. Validate the {*KeepaliveResponse*}*NewNodePr* using *NewNodePu*. If the message is invalid, discard it and terminate.
         3. Replace the appropriate fields in *ForwardFingerTable[i]*. Set *Time* to the current Unix timestamp. Store *NewNodePu* in *ForwardFingerTable[i]*.

A.4. Publishing a Certificate

A.4.1. Purpose of Operation

Security is partially provided through the use of digital signatures. In order to verify a digital signature, a public-key is retrieved from the DHT and used to decrypt the signature. This secures the integrity of a signed message, verifies the authenticity of the sender, and provides non-repudiation. Public-keys are stored in certificates, which are published onto the DHT. They may then be freely retrieved by all nodes using the publisher's node ID. All nodes require a certificate in order to operate within the proposed model.

A.4.2. Context of Operation

Before performing any other operation, a new node joining the network uses one or more entry nodes to assist in the publication of its certificate. Every operation may require the retrieval of certificates in order to identify faulty nodes.

Preconditions: The IP address of at least one entry node is known.

Postconditions: The public-key certificate is published successfully, and is maintained by at least 67% of *r* replica nodes. Otherwise it is declined. Malicious nodes may be identified.

A.4.3. Global Variables

The following global variables are defined. These global variables are addressable by any set of processing on the node that defined them, until the operation terminates.

The Publishing node defines the variable *PublisherProofs[]*, it has the form:

PublisherProofs[ReplicaIDReplicaID] = {*ReplicaID*, {*Challenge*}*ReplicaPr*, *Response*}

Replica nodes define the variable *ReplicaProofs[]*, it has the form:

ReplicaProofs[OtherReplicaID] = {*OtherReplicaID*, {*Certificate*}*PublisherPr,*

{*Challenge*, *Response*}*PublisherPr*, *Nonce*}*OtherReplicaPr*

A.4.4. Publishing Node: Issuing the Certificate Publish Request

1. Select and create Public-key (*PublisherPu*), and corresponding Private-key (*PublisherPr*)
2. Calculate the node ID (*PublisherID*) by performing hash(*PublisherPu*).
3. Select a sequence number (*SeqNum*), typically zero.
4. Select a timestamp (*Time*) either now, or in the past.
5. Create certificate (*Certificate*) {*PublisherID*, *PublisherPu*, *SeqNum*, *Time*, Optional information}
6. Sign certificate to create {*Certificate*}*PublisherPr*
7. Apply the finger table replication scheme to create *r* messages of the form:

   {*ReplicaKey*, *RON*, "Certificate Publish" {*Certificate*}*PublisherPr*}*PublisherPr*

   Where *RON* is an integer value, ranging from 1 to *r* increasing monotonically by one, that uniquely identifies each replica message.
8. Send each of the above *r* messages to each replica node using the *ReplicaKey* in each message.

A.4.5. Replica Node: Upon Receiving a Certificate Publish message

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Certificate Publish" {*Certificate*}*PublisherPr*}*PublisherPr*

2. Extract {*Certificate*}*PublisherPr* from the message. The message should have the form:

   {*PublisherID*, *PublisherPu*, *SeqNum*, *Time*, Optional information}*PublisherPr*

3. Perform the following checks, discard *Certificate* and terminate if any fail:
   1. Verify that *PublisherID* is the cryptographic hash of *PublisherPu*.
   2. Verify that *Time* is in the past, and has not already expired.
   3. Validate *Certificate* by verifying the digital signature with *PublisherPu*.
4. Select a random large integer value *Nonce*.
5. Encrypt *Nonce* using *PublisherPu* to create *Challenge*.
6. Sign *Challenge* to create {*Challenge*}*ReplicaPr*.
7. Create message {*PublisherID*, "Challenge", *ReplicaID*, {*Challenge*}*ReplicaPr*}*ReplicaPr*.

   ReplicaID is the node ID of the current replica node creating the *Challenge*.

   ReplicaPr is the private-key of the current replica node creating the *Challenge*.
8. Send the above message directly to the publishing node.

A.4.6. Publishing Node: Upon Receiving a Challenge message

1. The message should have the form:

   {*PublisherID*, "Challenge", *ReplicaID*, {*Challenge*}*ReplicaPr*}*ReplicaPr*

2. Decrypt *Challenge* using *PublisherPr* to create *Response*.
3. *PublisherProofs[ReplicaID]* = {*ReplicaID*, {*Challenge*}*ReplicaPr*, *Response*}
4. Create the message

   {*ReplicaID*, "Response", {*Challenge*, *Response*}*PublisherPr*}*PublisherPr*

5. Send the above message directly to the replica node issuing the challenge.

A.4.7. Replica Node: Upon Receiving a Response message

1. The message should have the form:

   {*ReplicaID*, "Response", {*Challenge*, *Response*}*PublisherPr*}*PublisherPr*

2. Ensure the received *Challenge* matches the issued *Challenge*. If not, discard the message, resend the *Challenge* and terminate.
3. Validate {*Challenge*, *Response*}*PublisherPr* using *PublisherPu.* If the message is invalid, discard the message, resend the *Challenge* and terminate.
4. Compare the *Response* to the previously created *Nonce*. If the two are not identical, terminate.
5. Create message *(Proof*) to confirm that the publisher has responded correctly to the *Challenge*:

   {*ReplicaID*, {*Certificate*}*PublisherPr*, {*Challenge*, *Response*}*PublisherPr*, *Nonce*}*ReplicaPr*

6. Apply the finger table replication scheme to create *r*-1 messages of the form:

   {*ReplicaKey*, *RON*, "Proof", {*Proof*}*ReplicaPr*}*ReplicaPr*.

7. Send each of the above *r*-1 messages to each replica node using the *ReplicaKey* in each message.

A.4.8. Replica Node: Upon Receiving a Proof message

For the following processing, *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu.*

1. The message should have the form: {*ReplicaKey*, *RON*, "Proof",

   {*Proof*}*OtherReplicaPr*}*OtherReplicaPr*.

2. Extract the {*Proof*}*ReplicaPr* from the message. The message should have the form:

   {*OtherReplicaID*, {*Certificate*}*PublisherPr*,

   {*Challenge*, *Response*}*PublisherPr*, *Nonce*}*OtherReplicaPr*

3. If the {*Certificate*}*PublisherPr* contained in {*Proof*}*ReplicaPr* has not been previously received, perform the processing detailed in **Section A.4.5. Replica Node: Upon Receiving a Certificate Publish message** before continuing.

4. Validate {*Certificate*}*PublisherPr* and {*Challenge*, *Response*}*PublisherPr* using *PublisherPu*. If either digital signature is invalid, discard *Proof* and terminate.

5. If the {*Certificate*}*PublisherPr* differs from the {*Certificate*}*PublisherPr* previously received, the publisher is malicious. The protocol terminates with the certificate rejected.

6. Encrypt *Nonce* with *PublisherPu* and compare it to *Challenge*, if the two values are not identical:

   1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.

   2. Validate the received {*Proof*}*OtherReplicaPr* using *OtherReplicaPu*. If the digital signature is invalid, discard {*Proof*}*OtherReplicaPr*.

   3. If the signature is valid, the node identified by *OtherReplicaID* is malicious, and is issuing a *Proof*, even when the publisher is not responding to the correct *Challenge*.

7. If *ReplicaProofs[OtherReplicaID]* is null:

   1. *ReplicaProofs[OtherReplicaID]* = {*Proof*}*OtherReplicaPr*.

   2. Rebroadcast {*Proof*}*ReplicaPr* to all other replica nodes.

8. If *ReplicaProofs[OtherReplicaID]* is not null:
    1. Compare the two copies of {*Proof*}*OtherReplicaPr* . If both messages are identical, the newly received one is discarded.
    2. If the two copies of {*Proof*}*OtherReplicaPr* are not identical:
        1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
        2. Validate the stored and received {*Proof*}*OtherReplicaPr* using *OtherReplicaPu*. Discard each message found invalid, replace the stored value with the received one if necessary.
        3. If the digital signature is valid, the owner of *OtherReplicaID* is malicious. Discard *ReplicaProofs[OtherReplicaID]* and any messages from the owner of *OtherReplicaID*.
        4. Rebroadcast the received {*Proof*}*OtherReplicaPr* to all other replica nodes.

A.4.9. Replica Node: Upon Receiving a majority of Proof messages

This set of processing occurs when *ReplicaProofs[]* contains more than 67% of *r Proof* messages. These steps may repeat as *Proof* messages are received and discarded.

1. All *Proof*s in *ReplicaProofs[]* are analysed for a *Decision*. A *Decision* is a Boolean value on whether the *Response* value in the *Proof* message matches the corresponding *Nonce*.
2. If all messages in *Decision*s are in agreement, the protocol terminates. A unanimous *Decision* of "true" results in the certificate being published; "false" results in the certificate being declined.
3. If any *Decision*s are not in agreement, create the message

    {*PublisherID*, "Proof Request"}*ReplicaPr*

4. Send the above message directly to the publishing node.

A.4.10. Publishing Node: Upon Receiving a Proof Request message

1. Create the message {*ReplicaID*, "Proof Response", *PublisherProofs[]*}*PublisherPr*
2. Send the above message directly to the replica node that sent the proof request message.

A.4.11. Replica Node: Upon Receiving a Proof Response message

For the following processing, *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu.*

1. The message should have the form: {*ReplicaID*, "Proof Response", *PublisherProofs[]*}*PublisherPr*.
2. Validate the message using *PublisherPu*. If the digital signature is invalid, discard the message and resend the proof request message.
3. Extract *PublisherProofs[]* from the message. It should be an array of messages of the form:

   {*OtherReplicaID*, {*Challenge*}*OtherReplicaPr*, *Response*}
4. For each *OtherReplicaID* in both *ReplicaProofs[OtherReplicaID]* and *PublisherProofs[OtherReplicaID]*
   1. Compare the *Challenge* in each set. If the messages differ:
      1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
      2. Validate {*Challenge*}*OtherReplicaPr* and {*Proof*}*OtherReplicaPr* using *OtherReplicaPu*. If either digital signature is invalid, discard both *ReplicaProofs[OtherReplicaID]* and *PublisherProofs[OtherReplicaID]*.
      3. If the digital signature is valid, the owner of *OtherReplicaID* is malicious.
   2. Compare the *Response* messages in each set. If any differ, the publishing node is malicious and the protocol terminates.
5. Any remaining *Decision* messages that are not in agreement with the majority of responses are marked as suspicious.
6. Any remaining *Decision*s that are not in agreement with the *Decision* of the current replica node are marked as suspicious.
7. For each suspicious *Proof* as *SuspiciousProof*:
   1. Retrieve the certificate (*OtherReplicaCert*) of the replica node that sent *SuspiciousProof* using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
   2. Validate *SuspiciousProof* using *OtherReplicaPu*. If the digital signature is invalid, discard *SuspiciousProof*.
8. If 67% of *r Decision*s in *ReplicaProofs[]* remain, the protocol terminates with the majority *Decision* being trusted.
9. Else the protocol waits for another majority of proof messages.
10. If a majority of proof messages are never received, the protocol terminates with the certificate being rejected. The publishing node may make another attempt.

A.5. Publishing a Name Entry

A.5.1. Purpose of Operation

The primary purpose of the proposed model is the establishment of a secure and decentralised name service. The following processing details the secure publishing of name-address mappings. The process makes use of the modified optimistic atomic broadcast protocol.

A.5.2. Context of Operation

At any time, a node that has already published its certificate may initiate a name publish request. After the publishing has completed, any other node on the network may freely query the published entry.

Preconditions: The initiating node owns a published certificate.

The initiating node's finger table is complete.

Postconditions: The name entry is published successfully and maintained on at least 67% of *r* replica nodes for that entry. Otherwise it is declined and not published. Malicious nodes may be identified.

A.5.3. Global Variables

The following global variables are defined. These global variables are addressable by any set of processing on the node that defined them, until the operation terminates.

Replica nodes define the variable *ReplicaConfirmations[]*, it has the form:

ReplicaConfirmations[OtherReplicaID] = {*Confirmation*}*OtherReplicaPr*.

A.5.4. Publishing Node: Initiating the Name Publish Request

1. Select the name (*Name*) and address (*Address*) to be paired.
2. Perform hash(*Name*) to create *Key*.
3. Select a sequence number (*SeqNum*), typically zero.
4. Create the message (*NameEntry*) of the form:

   {*Key*, *Name*, *Address*, *SeqNum, PublisherID*}

   Depending on implementation details, *PublisherPu* may be included.

   Refer to **Section 3.5.3. Consistency Selection Criteria**.
5. Sign *NameEntry* to create {*NameEntry*}*PublisherPr*.
6. Apply the finger table replication scheme to create *r* messages of the form:

   {*ReplicaKey*, *RON*, "Name Publish" {*NameEntry*}*PublisherPr*}*PublisherPr*.
7. Send each of the above *r* messages to each replica using the *ReplicaKey* in each message.

A.5.5. Replica Node: Upon receiving a Name Publish message

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Name Publish" {*NameEntry*}*PublisherPr*}*PublisherPr*.

2. Extract the {*NameEntry*}*PublisherPr* from the message. The message should have the form:

   {*Key*, *Name*, *Address*, *SeqNum, PublisherID*}*PublisherPr*.

3. Search local storage for an entry with *Key* and *Name.* If found, or if another request with the same *Name* is in process, discard the *NameEntry* and terminate.

4. Retrieve the certificate (*PublisherCert*) using *PublisherID*. Extract *PublisherPu* from *PublisherCert*.

5. Validate the *NameEntry* by verifying the digital signature using *PublisherPu*. If *NameEntry* is invalid, discard it and terminate.

6. If *PublisherPu* is included in *NameEntry*, verify that the retrieved *PublisherPu* and the included *PublisherPu* are identical, if they are not, discard *NameEntry* and terminate.

7. Create the message (*Confirmation*) {*ReplicaID* , {*NameEntry*}*PublisherPr*}

8. Sign *Confirmation* to create {*Confirmation*}*ReplicaPr*

9. Apply the finger table replication scheme to create *r*-1 messages of the form:

   {*ReplicaKey*, *RON*, "Name Confirmation" {*Confirmation*}*ReplicaPr*}*ReplicaPr*

10. Send each of the above *r*-1 messages to each replica using the *ReplicaKey* in each message.

A.5.6. Replica Node: Upon receiving a Name Confirmation message

For the following processing, *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr*.

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Name Confirmation" {*Confirmation*} *OtherReplicaPr*}*OtherReplicaPr*

2. Extract {*Confirmation*}*ReplicaPr* from the message. The message should have the form:

   {*OtherReplicaID*, {*NameEntry*}*PublisherPr*}*OtherReplicaPr*

3. If the {*NameEntry*}*PublisherPr* has not been previously received, perform the processing detailed in **Section A.5.5. Replica Node: Upon Receiving a Name Publish message** before continuing.

4. Validate the received {*NameEntry*}*PublisherPr* using the previously retrieved *PublisherPu*. If {*NameEntry*}*PublisherPr* is invalid, discard it.

5. If the {*NameEntry* }*PublisherPr* differs from the {*NameEntry*}*PublisherPr* previously received, the publisher is malicious. The protocol terminates with the certificate rejected.

6. If *ReplicaConfirmations[OtherReplicaID]* is null:

   1. *ReplicaConfirmations[OtherReplicaID]* = {*Confirmation*}*OtherReplicaPr*

   2. Rebroadcast {*Confirmation*}*OtherReplicaPr* to all other replica nodes.

7. If *ReplicaConfirmations[OtherReplicaID]* is not null:
    1. Compare the two copies of {*Confirmation*}*OtherReplicaPr* . If both are identical, the newly received one is discarded.
    2. If *ReplicaConfirmations[OtherReplicaID]* and {*Confirmation*}*OtherReplicaPr* differ:
        1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
        2. Validate the stored and received {*Confirmation*}*OtherReplicaPr* using *OtherReplicaPu*. Discard each message found invalid, replace *ReplicaConfirmations[OtherReplicaID]* with {*Confirmation*}*OtherReplicaPr* if necessary.
        3. If the digital signature is valid, the owner of *OtherReplicaID* is malicious. Discard *ReplicaConfirmations[OtherReplicaID]* and any messages from the owner of *OtherReplicaID*.
        4. Rebroadcast the received {*Confirmation*}*OtherReplicaPr* to all other replica nodes.

A.5.7. Replica Node: Upon receiving a majority of Confirmation messages

This set of processing occurs when *ReplicaConfirmations[]* contains more than 67% of *r Confirmation* messages. These steps may repeat as additional *Confirmation* messages are received. For the following processing, *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu.*

1. If all {*Confirmation*}*OtherReplicaPr* messages contain the identical {*NameEntry*}*PublisherPr*, the name is published and the protocol terminates.
2. If any {*NameEntry*}*PublisherPr* messages are not identical, *Confirmation*s containing *NameEntry* messages differing from the majority are marked as suspicious.
3. *Confirmation* messages containing {*NameEntry*}*PublisherPr* messages differing with the {*NameEntry*}*PublisherPr* confirmed by the current replica node are marked as suspicious.
4. For each suspicious *ReplicaConfirmations[OtherReplicaID]* as *SuspiciousConfirmation*:
    1. Validate {*NameEntry*}*PublisherPr* using the previously retrieved *PublisherPu*, discard the message if the digital signature is found invalid.
    2. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID* contained in *SuspiciousConfirmation*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
    3. Validate *SuspiciousConfirmation* using *OtherReplicaCert*. If the digital signature is invalid, discard *SuspiciousConfirmation*.
5. All faulty messages are now eliminated. If more than 67% of *r Confirmation* messages remain, a majority decision may be made. If the majority of replica nodes have issued *Confirmation* messages containing the same {*NameEntry*}*PublisherPr*, that {*NameEntry*}*PublisherPr* is published.
6. Else, the protocol terminates with {*NameEntry*}*PublisherPr* being rejected. The publishing node may make another attempt, unless it was previously found malicious.

A.6. Transferring a Name Entry

A.6.1. Purpose of Operation

Throughout the life of the system, name entries owned by one publisher may need to be transferred to another. In this case, the receiving entity becomes the "publisher". Publishers of name entries are identified through the ownership of a private-key used to sign the name entry. This operation serves to move ownership of a name entry from one private-key to another. This may be required as key become compromised, or as certificates expire over time and node churn is required.

A.6.2. Context of Operation

After a name entry has been published, ownership of the entry may need to be moved. This is voluntarily done if ownership the name is moved to another entity, and is required before certificates expire due to enforced node churn.

This operation is functionally identical to the publishing of name entries, with the addition of processing to ensure the transferring party owned the name entry to begin with.

Preconditions: The initiating node owns a published certificate.

The initiating node's finger table is complete.

The transferred name is published, and signed by the private key corresponding to the public key contained in the published certificate.

The new publisher node owns a published certificate.

Postconditions: The name entry is transferred successfully and maintained on at least 67% of *r* replica nodes for that entry, under the new certificate. Otherwise it is declined and not transferred. Malicious nodes may be identified.

A.6.3. Global Variables

The following global variables are defined. These global variables are addressable by any set of processing on the node that defined them, until the operation terminates.

Replica nodes define the variable *ReplicaConfirmations[]*, it has the form:

ReplicaConfirmations[OtherReplicaID] = {*Confirmation*}*OtherReplicaPr*.

A.6.4. Transferring Node: Initiating the Name Transfer Request

1. Select the currently published *NameEntry* to be transferred to the new private-key. *NameEntry* messages have the form:

   {*Key*, *Name*, *Address*, *SeqNum, OldPublisherID*}*OldPublisherPr*

   OldPublisherID is the node ID (*PublisherID*) of the node transferring ownership.

   OldPublisherPr is the private-key (*PublisherPr)* of the node transferring ownership

2. Allow the node receiving ownership to select its node ID (*NewPublisherID*) and private-key (*NewPublisherPr*) of, with the corresponding public-key in an already published certificate.
3. Select a new sequence number (*NewSeqNum*) so that *NewSeqNum* > *SeqNum*.
4. Create message(*NewNameEntry*)

   {*Key*, *Name*, *Address*, *NewSeqNum, NewPublisherID*}

   Depending on implementation details, *NewPublisherPu* may be included.

   Refer to **Section 3.5.3. Consistency Selection Criteria**.

5. Have the receiving party sign *NewNameEntry* to create {*NewNameEntry*}*NewPublisherPr*.
6. Sign {*NewNameEntry*}*NewPublisherPr* to create {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr*.
7. Apply the finger table replication scheme to create *r* messages of the form:

   {*ReplicaKey*, *RON*, "Name Transfer"

   {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr*}*OldPublisherPr*.

8. Send each of the above *r* messages to each replica using the *ReplicaKey* in each message.

A.6.5. Replica Node: Upon receiving a Name Transfer message

A "Name Transfer" message may be received directly from the transferrer, or indirectly from other replica nodes as they broadcast their confirmation messages. This process is repeated each time a {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* message is received, regardless of origin.

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Name Transfer"

   {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr*}*OldPublisherPr*

2. Extract the {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* from the message. The message should have the form:

   {{*Key*, *Name*, *Address*, *NewSeqNum, NewPublisherID*}*NewPublisherPr*}*OldPublisherPr*

3. Search local storage for an entry with *Key* and *Name.* If the entry is not found, simply skip the next steps and wait for Transfer Confirmation messages. Otherwise the message should have the form:

   {*Key*, *Name*, *Address*, *SeqNum, OldPublisherID*}*OldPublisherPr*

4. Retrieve the certificate (*OldPublisherCert*) using *OldPublisherID*. Extract *OldPublisherPu* from *OldPublisherCert*.

5. Validate {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* by verifying the digital signature using *OldPublisherPu*. If the message is invalid, discard it and terminate.

6. Retrieve the certificate (*NewPublisherCert*) using *NewPublisherID*. Extract *NewPublisherPu* from *NewPublisherCert*.

7. Validate {*NewNameEntry*}*NewPublisherPr* by verifying the digital signature using *NewPublisherPu*. If the message is invalid, discard it and terminate.

8. If included in *NewNameEntry*, verify that the retrieved *NewPublisherPu* and the *NewPublisherPu* included in the name entry are identical, if they are not, discard *NewNameEntry* and terminate.

9. Create the message (*Confirmation*) {*ReplicaID*, {*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr*}

10. Sign *Confirmation* to create {*Confirmation*}*ReplicaPr*

11. Apply the finger table replication scheme to create *r*-1 messages of the form:

    {*ReplicaKey*, *RON*, "Transfer Confirmation" {*Confirmation*}*ReplicaPr*}*ReplicaPr*

12. Send each of the above *r*-1 messages to each replica using the *ReplicaKey* in each message.

A.6.6. Replica Node: Upon receiving a Transfer Confirmation message

For the following processing, *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu.*

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Transfer Confirmation" {*Confirmation*}*ReplicaPr*}*ReplicaPr*

2. Extract {*Confirmation*}*ReplicaPr* from the message. The message should have the form:

   {*OtherReplicaID*, {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr*}*OtherReplicaPr*.

3. If {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* has not been previously received, perform the processing detailed in **Section A.6.5. Replica Node: Upon Receiving a Name Transfer message** before continuing.

4. Validate the received {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* using the previously retrieved *OldPublisherPu*. If it is invalid, discard it.

5. If the {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* differs from the one previously received, the owner of *OldPublisherID* is malicious. The protocol terminates with the transfer rejected.

6. Validate the received {*NewNameEntry*}*NewPublisherPr* using the previously retrieved *NewPublisherPu*. If it is invalid, discard it.

7. If the {*NewNameEntry*}*NewPublisherPr* differs from the one previously received, the owner of *NewPublisherID* is malicious. The protocol terminates with the transfer rejected.

8. If *ReplicaConfirmations[OtherReplicaID]* is null:

   1. *ReplicaConfirmations[OtherReplicaID]* = {*Confirmation*}*OtherReplicaPr*.

   2. Rebroadcast {*Confirmation*}*OtherReplicaPr* to all other replica nodes.

9. If *ReplicaConfirmations[OtherReplicaID]* is not null:

   1. Compare the two copies of {*Confirmation*}*OtherReplicaPr* . If both are identical, the newly received one is discarded.

   2. If *ReplicaConfirmations[OtherReplicaID]* and {*Confirmation*}*OtherReplicaPr* differ:

      1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.

      2. Validate the stored and received {*Confirmation*}*OtherReplicaPr* using *OtherReplicaPu*. Discard each message found invalid, *ReplicaConfirmations[OtherReplicaID]* with {*Confirmation*}*OtherReplicaPr* if necessary.

      3. If the digital signature is valid, the owner of *OtherReplicaID* is malicious. Discard *ReplicaConfirmations[OtherReplicaID]* and any messages from the owner of *OtherReplicaID*.

      4. Rebroadcast the received { *Confirmation*}*OtherReplicaPr* to all other replica nodes.

A.6.7. Replica Node: Upon receiving a majority of Confirmation messages

This set of processing occurs when *ReplicaConfirmations[]* contains more than 67% of *r Confirmation* messages. These steps may repeat as additional *Confirmation* messages are received. *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu*.

1. If all {*Confirmation*}*OtherReplicaPr* messages contain identical {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* messages, {*NewNameEntry*}*NewPublisherPr* replaces {*OldNameEntry*}*OldPublisherPr*.
2. If any {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* messages are not identical, *Confirmation*s differing from the majority are marked as suspicious.
3. *Confirmation* messages containing {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* messages differing with the {{*NewNameEntry*}*NewPublisherPr*}*OldPublisherPr* confirmed by the current replica node are marked as suspicious.
4. For each suspicious *ReplicaConfirmations[OtherReplicaID]* as *SuspiciousConfirmation* :
    1. The certificate (*OtherReplicaCert*) of the node that sent *SuspiciousConfirmation* is retrieved using the *OtherReplicaID* contained in *SuspiciousConfirmation*. *OtherReplicaPu* is retrieved from *OtherReplicaCert*.
    2. Validate *SuspiciousConfirmation* using *OtherReplicaCert*. If the digital signature is invalid, discard *SuspiciousConfirmation*.
5. All provably faulty messages are now eliminated. If more than 67% of *r Confirmation* messages remain, a majority decision may be made. If the majority of replica nodes have issued *Confirmation* messages containing the same {*NewNameEntry*}*NewPublisherPr*, it replaces {*OldNameEntry*}*OldPublisherPr*.
6. Else, the protocol terminates with the transfer being rejected. The transferring node may make another attempt.

A.7. Updating Entries

A.7.1. Purpose of Operation

Physical and logical location of named objects may change over time; typically this involves a change of address. As such, the owner of the name entry may need to update the address of the named object. Similarly, the optional identifying information contained in a certificate may need to change. Publishers prove ownership of a particular entry by signing a message with the private key that corresponds to the certificate with the *PublisherID* contained in the original entry.

A.7.2. Context of Operation

After an entry is published, it may be modified to a limited extent by the publisher. Name entries may have their address updated, certificates may have only their optional identifying information updated.

Preconditions: The initiating node owns a published certificate.

The initiating node's finger table is complete.

The updated entry is published, and signed by the private key corresponding to the public key contained in the published certificate.

Postconditions: The entry is updated successfully and the update will eventually epidemically propagate to at least 67% of *r* replica nodes. Otherwise it is declined and not updated. The publisher may be identified as malicious.

A.7.3. Publishing Node: Initiating the Update request

1. Select any previously created *Entry*, as defined in: **Section A. 4. Publishing a Certificate** or **Section A.5. Publishing a Name Entry**, *Entry* will have the form of either:

   {*Key*, *Name*, *Address*, *SeqNum, PublisherID*}*PublisherPr*

   {*PublisherID*, *PublisherPu*, *SeqNum*, *Time,* Optional information}*PublisherPr*

2. Select either:
   - A new address (*NewAddress*) if *Entry* is a name entry.
   - New optional information if *Entry* is a certificate.
3. Select a new sequence number (*NewSeqNum*) so that *NewSeqNum* > *SeqNum*.
4. Create message (*NewEntry*) of the form of either

   {*Key*, *Name*, *NewAddress*, *NewSeqNum, PublisherID*}

   {*PublisherID*, *PublisherPu*, *NewSeqNum*, *Time,* Optional information}
5. Sign *NewEntry* to create {*NewEntry*}*PublisherPr*.
6. Apply the finger table replication scheme to create *r* messages of the form:

   {*ReplicaKey*, *RON*, "Name Update" {*NewEntry*}*PublisherPr*}*PublisherPr*.
7. Send each of the above *r* messages to each replica node using the *ReplicaKey* in each message.

A.7.4. Replica Node: Upon Receiving an Update message

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Name Update" {*NewEntry*}*PublisherPr*}*PublisherPr*

2. Extract the {*NewEntry*}*PublisherPr* from the message. It should have the form of either:

   {*Key*, *Name*, *NewAddress*, *NewSeqNum, PublisherID*}*PublisherPr*

   {*PublisherID*, *PublisherPu*, *NewSeqNum*, *Time*, Optional information}*PublisherPr*

3. Retrieve {*OldEntry*}*PublisherPr* from local storage using *Key* or *PublisherID*. If not found, perform the processing detailed in **Section A.9. Retrieving an Entry** and terminate. { *OldEntry* }*PublisherPr* will have the form of either:

   {*Key*, *Name*, *OldAddress*, *OldSeqNum, PublisherID*}*PublisherPr*

   {*PublisherID*, *PublisherPu*, *OldSeqNum*, Optional information}*PublisherPr*

4. Compare *OldEntry* and *NewEntry*, if they are identical, terminate.
5. Verify that ONLY the sequence number, and either address or optional information has changed. If any other field differs between *OldEntry* and *NewEntry*, discard *NewEntry* and terminate.
6. Verify *NewSeqNum* > *OldSeqNum*. If not, discard {*NewEntry*}*PublisherPr* and terminate.
7. If *NewEntry* is a certificate:
   1. Verify that *OldEntry* and *NewEntry* have not revoked by checking that *Time* has not expired, and no previous revocation messages are stored with *OldEntry*.
   2. If revocation messages are included in the update message, perform the processing detailed in **Section A.10. Revoking a Certificate**. Store the revocation messages if *NewEntry* is revoked.
8. If *PublisherPu* is not included in *NewEntry*, the certificate (*PublisherCert*) is retrieved using *PublisherID*. *PublisherPu* is retrieved from *PublisherCert*.
9. Validate {*NewEntry*}*PublisherPr* by verifying the digital signature using *PublisherPu*. If {*NewEntry*}*PublisherPr* is invalid, discard it and terminate.
10. If {*NewEntry*}*PublisherPr* is valid and *NewSeqNum* > *OldSeqNum*, store {*NewEntry*}*PublisherPr* in local storage, overwriting {*Entry*}*PublisherPr*.

A.8. Replica and Update Maintenance

A.8.1. Purpose of Operation

Similarly to how node disconnects affect the accuracy of other nodes' finger table, so too do they affect the reliability of the system. When a node disconnects, it takes all entries it was responsible for with it, leaving the system with one fewer replica of those entries. The system is naturally resilient to less than *r* node disconnects, however maintenance is required to ensure *r* replicas remain in the system consistently.

This operation has the second purpose of propagating update messages in an epidemic fashion. Updates are not performed atomically, thus it is feasible that there are replicas that have not received the updated entry. By combining replica maintenance with update propagation, the eventual integrity of entries is secured, with no additional messages sent.

A.8.2. Context of Operation

After a node becomes responsible for entries, it begins sending maintenance messages for those entries. The replica maintenance messages are sent every *m* seconds, with *m* being defined at system initialisation and remaining constant throughout the life of the system. A pseudo-randomly selected entry is selected, and shared with another replica node of that entry, also selected pseudo-randomly. This selection may be changed during system implementation. By skewing the pseudo-random selection of entries towards entries that have recently been updated, the updated entries are spread in an epidemic fashion.

Upon receiving a maintenance message, a node checks its local database of entries. If the entry does not exist in this database, it is retrieved. If the entry does exist and is identical to the received entry, no action is taken. If the entry does exist but differs from the updated message, several security checks are performed, before updating or denying the entry.

Preconditions: The initiating node owns a published certificate.

The initiating node's finger table is complete.

The maintained entry is published.

Postconditions: The entry is maintained on at least 67% of *r* replica nodes.

A.8.3. Every Replica Node: Every *m* seconds

1. Pseudo-randomly select an entry ({*Entry*}*PublisherPr*) that exists in local storage.

   This selection should show a bias towards any recently updated entries.

2. Pseudo-randomly select a *RON* and apply the finger table replication scheme to create a message of the form: {*OtherReplicaKey*, *RON*, "Maintenance", {*Entry*}*PublisherPr*}*ReplicaPr*

   This selection should show bias to other replicas that this replica has not already sent the above message to, or from whom an updated maintenance message has not been received.

3. Send {*Entry*}*PublisherPr* to the replica using *OtherReplicaKey* in the message. If *Entry* is a certificate, and any revocation messages are present, they are also sent.

A.8.4. Replica Node: Upon receiving a Maintenance message

1. Perform the processing detailed in **A.7.4. Replica Node: Upon Receiving an Update message**, considering the maintenance message as an update message.

A.9. Retrieving an Entry

A.9.1. Purpose of Operation

The previous operations have detailed the creation and maintenance of a decentralised and secure infrastructure for the storage of name entries and certificates. The following operation details the secure retrieval of these entries.

A.9.2. Context of Operation

This operation both relies on and supports the previous operations. Other operations provide the infrastructure from which entries are securely retrieved; however all other operations may require the retrieval of entries, typically certificates.

Requesting an entry is likely the most heavily used operation, executed at every name or certificate lookup, including those that form part of the other operations. To promote efficiency, the operation terminates early for the initiating node. A Byzantine agreement may then be initiated, and will continue between the replicas to re-establish consistency.

Preconditions: The initiating node owns a published certificate.

The initiating node's finger table is complete.

The requested entry is published.

Postconditions: The requested entry is returned. If the preconditions are not met, the entry is not returned. Nodes may be identified as malicious.

A.9.3. Global Variables

The following global variables are defined. These global variables are addressable by any set of processing on the node that defined them, until the operation terminates.

The initiating node defines the variable *ReceivedResponses[]*, it has the form:

InitiatorResponses[ReplicaID] = {{*Entry*}*PublisherPr*}*ReplicaPr*

The replica nodes define the variable *ReceivedResponses[]*, it has the form:

ReplicaResponses[ReplicaID] = {{*Entry*}*PublisherPr*}*ReplicaPr*

A.9.4. Initiating Node: Initiating an Entry Request

1. Select the key (*Key*) of the entry to be requested.
   - If the entry is a name, perform hash(*Name*) to create *Key*.
   - If the entry is a certificate, the node ID of the owner of the certificate is *Key*.
2. Create the message (*EntryRequest*) {*Key*, "Entry Request"}
3. Sign *EntryRequest* to create {*EntryRequest*}*InitiatorPr*

   Where *InitiatorPr* is the private key of the initiating node.
4. Apply the finger table replication scheme to create *r* messages of the form:

   {*ReplicaKey*, *RON*, "Entry Request" {*EntryRequest*}*InitiatorPr*}*InitiatorPr*.

5. Send each of the above *r* messages to each replica using the *ReplicaKey* in each message.

A.9.5. Replica Node: Upon receiving an Entry Request message

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Entry Request" {*EntryRequest*}*InitiatorPr*}*InitiatorPr*.

2. Extract the {*EntryRequest*}*InitiatorPr* from the message. It should have the form:

   {*Key*, "Entry Request"}*InitiatorPr*

3. Rebroadcast {*EntryRequest*}*InitiatorPr* to all other replica nodes.
4. Retrieve {*Entry*}*PublisherPr* from local storage using *Key.* If not found, retrieve the entry from the DHT, performing the processing detailed in **Section A.9. Retrieving an Entry** before continuing.
5. Sign {*Entry*}*PublisherPr* to create {{*Entry*}*PublisherPr*}*ReplicaPr*. If *Entry* is a certificate, and any revocation messages are present, they are included in *Entry*.
6. Create message {*InitiatorID*, "Entry Response", {{*Entry*}*PublisherPr*}*ReplicaPr*}*ReplicaPr*
7. Send the above message directly to the initiating node.

A.9.6. Initiating Node: Upon receiving an Entry Response message

1. The message should have the form:

   {*InitiatorID*, "Entry Response", {{*Entry*}*PublisherPr*}*ReplicaPr*}*ReplicaPr*

2. If *Entry* is a certificate:
   1. {*Entry*}*PublisherPr* should have the form:

      {*PublisherID*, *PublisherPu*, *SeqNum*, *Time,* Optional information}*PublisherPr*

   2. Perform the following checks, discard *Entry* if any fail:
      1. Verify that *PublisherID* is the cryptographic hash of *PublisherPu*.
      2. Validate *Entry* by verifying the digital signature with *PublisherPu*.
3. If *Entry* is a name entry:
   1. {*Entry*}*PublisherPr* should have the form:

      {*Key*, *Name*, *Address*, *SeqNum, PublisherID*}*PublisherPr*

   2. If {*Entry*}*PublisherPr* messages are received from multiple nodes, all claiming ownership of the same *ReplicaKey*, perform the processing detailed in **Section A.2.5. Message Forwarding: Upon Receiving Messages from Two Nodes Claiming the Same Key**.
4. *InitiatorResponses[ReplicaID]* = {{*Entry*}*PublisherPr*}*ReplicaPr*

A.9.7. Initiating Node: Upon receiving a majority of Entry Response messages

This set of processing occurs when *InitiatorResponses[]* contains more than 67% of *r* {{*Entry*}*PublisherPr*}*ReplicaPr* messages. These steps may repeat as additional messages are received.

1. If all {{*Entry*}*PublisherPr*}*ReplicaPr* messages in *InitiatorResponses[]* are identical, the *Entry* is deemed correct.
2. If some {{*Entry*}*PublisherPr*}*ReplicaPr* messages in *InitiatorResponses[]* differ:
    1. A majority decision is performed on the *PublisherID* present in each *Entry*.
    2. If *Entry* is a name entry:
        1. Retrieve the certificate (*PublisherCert*) for the node identified by the *PublisherID* in the majority of *Entry* messages. Extract *PublisherPu* from *PublisherCert*.
        2. Verify each {*Entry*}*PublisherPr* in *InitiatorResponses[]* using *PublisherPu*. If the digital signature is invalid, discard that {*Entry*}*PublisherPr*.
    3. If multiple *Entry* messages contain the same *PublisherID* and the same *SeqNum*, but otherwise differ, the node identified by *PublisherID* is malicious.
    4. Accept the *Entry*, agreeing with the majority, correctly signed, containing the highest *SeqNum*.
3. If *Entry* is a certificate, verify it has not been revoked by performing the following:
    1. Verify that *Time* is in the past, and has not already expired.
    2. If any revocation messages are included in the result set, perform the processing detailed in **Section A.10. Revocation of Certificates**.

Optionally:

4. Sign *InitiatorResponses[]* to create {*InitiatorResponses[]*}*InitiatorPr*
5. Apply the finger table replication scheme to create *r* messages of the form:

    {*ReplicaKey*, *RON*, “Agreement”, *InitiatorID*, {*InitiatorResponses[]*}*InitiatorPr*}*InitiatorPr*

6. Send each of the above *r* messages to each replica using the *ReplicaKey* in each message.

The remainder of the protocol is used to update the *Entry* messages stored at each replica node, and to identify possibly faulty or malicious replica nodes.

A.9.8. Replica Node: Upon Receiving an Agreement message

An "Agreement" message may be received directly from the initiator, or indirectly from other replica nodes as they rebroadcast the original message. This message will be repeated each time a name agreement message is received. *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu*.

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Agreement", *InitiatorID,* {*InitiatorResponses[]*}*InitiatorPr*}*InitiatorPr*

2. Extract {*InitiatorResponses[]*}*InitiatorPr* from the received message. *InitiatorResponses[]* should be an array of messages of the form: {{*Entry*}*PublisherPr*}*OtherReplicaPr*
3. Retrieve the certificate (*InitiatorCert*) of the initiating node using *InitiatorID*.
4. Verify the {*InitiatorResponses[]*}*InitiatorPr* using *InitiatorPr* contained in *InitiatorCert*. If the digital signature is invalid, discard the message and terminate.
5. For each *InitiatorResponses[OtherReplicaID]* as {{*Entry*}*PublisherPr*}*OtherReplicaPr*:
   1. If *ReplicaResponses[OtherReplicaID]* is null:
      1. *ReplicaResponses[OtherReplicaID]* = {{*Entry*}*PublisherPr*}*OtherReplicaPr*}
      2. Rebroadcast {*InitiatorResponses[]*}*InitiatorPr* to all other replica nodes
   2. If *ReplicaResponses[OtherReplicaID]* is not null:
      1. Compare *ReplicaResponses[OtherReplicaID]* and *InitiatorResponses[OtherReplicaID]*. If the two messages are identical, *InitiatorResponses[OtherReplicaID]* is discarded.
      2. If these two copies of {{*Entry*}*PublisherPr*}*ReplicaPr* differ:
         1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
         2. Verify each copy of {{*Entry*}*PublisherPr*}*ReplicaPr*. If a digital signature is invalid, discard the invalid message. Replace *ReplicaResponses[OtherReplicaID]* with *InitiatorResponses[OtherReplicaID]* if required.
         3. If both {{*Entry*}*PublisherPr*}*ReplicaPr* differ and are valid, the replica node identified by *ReplicaID* is malicious. Exclude the node from the remainder of the operation.
         4. Rebroadcast {*InitiatorResponses[]*}*InitiatorPr* to all other replica nodes
6. If this is the first time an agreement message has been received:
   1. Create the message (*ReplicaPair*)

      {*ReplicaID*, {*EntryRequest*}*InitiatorPr*, {{*Entry*}*PublisherPr*}*ReplicaPr*}

   2. Sign *ReplicaPair* to create {*ReplicaPair*}*ReplicaPr*
   3. Apply the finger table replication scheme to create *r*-1 messages of the form:

      {*OtherReplicaKey*, *RON*, "Replica Pair", {*ReplicaPair*}*ReplicaPr*}*ReplicaPr*

   4. Send the above *r*-1 messages to each replica using the *OtherReplicaKey* in the message.

A.9.9. Replica Node: Upon Receiving a Replica Pair message

OtherReplicaID refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu*.

1. The message should have the form:

   {*OtherReplicaKey*, *RON*, "Replica Pair", {*OtherReplicaPair*}*OtherReplicaPr*}*ReplicaPr*

2. Extract {*OtherReplicaPair*}*OtherReplicaPr* from the received message. It should be of the form:

   {*OtherReplicaID*, {*EntryRequest*}*InitiatorPr*, {{*Entry*}*PublisherPr*}*OtherReplicaPr*}

3. Validate {*EntryRequest*}*InitiatorPr* using *InitiatorPu* in the previously retrieved *InitiatorCert*. If the digital signature is invalid, discard *OtherReplicaPair* and terminate.
4. If the {*EntryRequest*}*InitiatorPr* differs from the {*EntryRequest*}*InitiatorPr* previously received, the initiator is malicious.
5. If the {{*Entry*}*PublisherPr* Is a certificate, perform the following checks. If either fail, the node identified by *OtherReplicaID* is faulty:
   1. Verify that *PublisherID* is the cryptographic hash of *PublisherPu*.
   2. Validate *Entry* by verifying the digital signature with *PublisherPu*.
6. If *ReplicaResponses[OtherReplicaID]* is null:
   1. *ReplicaResponses[OtherReplicaID]* ={{*Entry*}*PublisherPr*}*OtherReplicaPr*
   2. Rebroadcast {*OtherReplicaPair*}*OtherReplicaPr* to all other replica nodes using the finger table replication scheme.
7. If *ReplicaResponses[OtherReplicaID]* is not null:
   1. Compare *ReplicaResponses[OtherReplicaID]* and the new {{*Entry*}*PublisherPr*}*OtherReplicaPr*. If the two messages are identical, the new {{*Entry*}*PublisherPr*}*OtherReplicaPr* is discarded.
   2. If these two copies of {{*Entry*}*PublisherPr*}*ReplicaPr* differ:
      1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
      2. Verify each copy of {{*Entry*}*PublisherPr*}*ReplicaPr*. If a digital signature is invalid, discard the invalid message. Replace *ReplicaResponses[OtherReplicaID]* with *InitiatorResponses[OtherReplicaID]* if required.
      3. If both {{*Entry*}*PublisherPr*}*ReplicaPr* differ and are valid, the replica node identified by *ReplicaID* is malicious. Exclude the node from the remainder of the operation.
      4. Rebroadcast {*InitiatorResponses[]*}*InitiatorPr* to all other replica nodes

A.9.10. Replica Node: Upon receiving a majority of Replica Pair messages

This set of processing occurs when *ReplicaResponses[]* contains more than 67% of *r* {{*Entry*}*PublisherPr*}*OtherReplicaPr* messages. These steps may repeat as additional messages are received. *OtherReplicaID* refers to the *ReplicaID* in the previous sections, the Node ID of the replica from whom the message is received. Similarly for *OtherReplicaPr* and *OtherReplicaPu*.

1. If any {{*Entry*}*PublisherPr*}*OtherReplicaPr* messages in *ReplicaResponses[]* are not identical, the {{*Entry*}*PublisherPr*}*OtherReplicaPr* that differ from the majority are marked as suspicious.
2. {{*Entry*}*PublisherPr*}*OtherReplicaPr* messages that differ from the locally stored {*Entry*}*PublisherPr* are marked as suspicious.
3. For each suspicious *ReplicaResponses[OtherReplicaID]*:
    1. Retrieve the certificate (*OtherReplicaCert*) using *OtherReplicaID*. Extract *OtherReplicaPu* from *OtherReplicaCert*.
    2. Verify {{*Entry*}*PublisherPr*}*OtherReplicaPr* using *OtherReplicaPu*. If the digital signature is invalid, discard *ReplicaResponses[OtherReplicaID]*.
3. If 67% of *r ReplicaResponses[]* remain:
    1. A majority decision is performed on the *PublisherID* present in each *Entry*.
    2. If *Entry* is a name entry:
        1. Retrieve the certificate (*PublisherCert*) for the node identified by the *PublisherID* in the majority of *Entry* messages. Extract *PublisherPu* from *PublisherCert*.
        2. Verify each remaining {*Entry*}*PublisherPr* in *ReplicaResponses[]* using *PublisherPu*. If the digital signature is invalid, discard that {*Entry*}*PublisherPr*.
    3. If multiple *Entry* messages contain the same *PublisherID* and the same *SeqNum*, but otherwise differ, the node identified by *PublisherID* is malicious.
    4. Accept the *Entry*, agreeing with the majority, correctly signed, containing the highest *SeqNum*.
    5. If the accepted *Entry* is a certificate, verify it has not been revoked by performing the following:
        1. Verify that *Time* is in the past, and has not already expired.
        2. If any revocation messages are included in the result set, perform the processing detailed in **Section A.10. Revocation of Certificates**.
    6. If necessary, update the local {*Entry*}*PublisherPr* to reflect the accepted *Entry*.
4. If less than 67% of *r Responses[]* messages agree, the request may be repeated.

A.10. Revoking a Certificate

A. 10.1. Purpose of Operation

Throughout the lifecycle of the system, certificates may become compromised, and malicious nodes will be identified. When this occurs, the compromised or malicious certificate needs to be revoked. After a certificate is revoked, the revocation message is considered a part of the certificate and is returned on every request. This also means that the revocation is shared via the epidemic consistency algorithm.

A. 10.2. Context of Operation

Certificates may be revoked by the publisher of the certificate at any point after publishing. This is typically done in the case of the private-key corresponding to the public-key in the certificate becoming compromised. The revocation is irreversible, and may be done even after the certificate is compromised.

Additionally, certificates may be revoked by entities other than the publisher. During all operations making use of Byzantine agreements in this thesis, it is possible that nodes be identified as malicious when issuing multiple conflicting decisions. These correctly signed, but conflicting, decisions may also serve as a revocation message.

Finally, certificates may become revoked if their time-to-live has expired.

Preconditions: The initiating node owns a published certificate.

The initiating node's finger table is complete.

OR

Conflicting messages, correctly signed by the same private-key, and belonging to the same operation, have been received.

Postconditions: The certificate is revoked. This revocation will eventually propagate to at least 67% of *r* replica nodes for the certificate, through the update algorithms detailed in **Section A.7. Updating Entries** and **Section A.8. Replica and Update Maintenance**.

A. 10.3. Publishing Node: Initiating a Certificate Revocation

1. Select the certificate (*RevokedCertificate*) to be revoked. It should have the form:

   {*PublisherID*, *PublisherPu*, *NewSeqNum*, *Time,* Optional information}*PublisherPr*

2. Create the message (*SelfRevocation*) {*PublisherID*, *PublisherPu*, "Self Revocation"}
3. Sign *SelfRevocation* to create {*SelfRevocation*}*PublisherPr*.
4. Apply the finger table replication scheme to create *r* messages of the form:

   {*ReplicaKey*, *RON*, "Self Revocation" {*SelfRevocation*}*PublisherPr*}*PublisherPr*.

5. Send each of the above *r* messages to each replica using the *ReplicaKey* in each message.

A. 10.4. Any Node: Upon Detecting Malicious Activity

This operation is initiated after a Byzantine agreement, where a node has been found malicious.

1. At this point, two or more messages (*Message[]*) have been obtained, both from the same node, but each differing from the others. These messages have the form of one of the following:
    - (*NameEntry*) {*Key*, *Name*, *Address*, *SeqNum, PublisherID*}*PublisherPr*
    - {*ReplicaID* , {*NameEntry*}*PublisherPr*}*ReplicaPr*
    - (*Certificate*) {*PublisherID*, *PublisherPu*, *NewSeqNum*, *Time,* Optional information}*PublisherPr*
    - {*ReplicaID*, {*Certificate*}*PublisherPr*,{*Challenge*, *Response*}*PublisherPr*, *Nonce*}*ReplicaPr*

      And sometimes {*ReplicaID*, {*Challenge*}*ReplicaPr*, *Response*}*PublisherPr*
    - {{*NameEntry*}*NewPublisherPr*}*OldPublisherPr*
    - {*ReplicaID* , {{*NameEntry*}*NewPublisherPr*}*OldPublisherPr*}*ReplicaPr*
2. Create the message (*OtherRevocation*) {*MaliciousID*, "Other Revocation", *Message[]*}

   Where *MaliciousID* is the node ID of the malicious node.
3. Apply the finger table replication scheme to create *r* messages of the form:

   {*ReplicaKey*, *RON*, "Other Revocation" *OtherRevocation*}*CurrentPr*
4. Send each of the above *r* messages to each replica using the *ReplicaKey* in each message.

A. 10.5.Replica Node: Upon Receiving a Self Revocation message

1. The message should have the form:

   {*ReplicaKey*, *RON*, "Self Revocation" {*SelfRevocation*}*PublisherPr*}*PublisherPr*
2. Extract the {*SelfRevocation*}*PublisherPr* from the message. It should have the form:

   {*PublisherID*, *PublisherPu*, "Self Revocation"}
3. Retrieve {*Certificate*}*PublisherPr*, either from local storage using *PublisherID* or from a previous retrieval operation using *PublisherID*. {*Certificate*}*PublisherPr* will have the form:

   {*PublisherID*, *PublisherPu*, *SeqNum*, Optional information}*PublisherPr*
4. Verify that both *PublisherPu* messages are identical, else discard *SelfRevocation* and terminate.
5. Validate {*SelfRevocation*}*PublisherPr* by verifying the digital signature using *PublisherPu*. If {*SelfRevocation*}*PublisherPr* is invalid, discard it and terminate.
6. If the message is correctly signed, {*Certificate*}*PublisherPr* is revoked.

A. 10.6. Retrieving Node or Replica Node: Upon Receiving an Other Revocation message

1. The message should have the form:

   {*MaliciousID*, "Other Revocation", *Message[]*}

2. Extract *Message[]* from the message. It should consist of an array of messages of the form detailed in **Section A.10.4. Any Node: Upon Detecting Malicious Activity**.

3. Retrieve {*Certificate*}*MaliciousPr*, either from local storage using *MaliciousID* or from a previous retrieval operation using *MaliciousID*. {*Certificate*}*MaliciousPr* will have the form:

   {*MaliciousID*, *MaliciousPu*, *SeqNum*, Optional information}*MaliciousPr*

4. If *Message[]* contains multiple *NameEntry* or *Certificate* messages:
   1. Verify that both *Entry* messages are for the same *Key* or *PublisherID*.
   2. If both *Entry* messages do not contain identical *SeqNum*, discard *Message[]* and terminate.
5. Otherwise, if *Message[]* contains messages other than name entries or certificates:
   1. Verify that all messages in *Message[]* are from the same operation. Ensuring that all messages contain identical *NameEntry* or *Certificate* messages. If not, discard *Message[]* and terminate.
6. Verify that messages in *Message[]* are otherwise different in some way and contain different decisions. If they are identical, no conflict is found, discard *Message[]* and terminate.
7. Verify the authenticity and integrity of every message in *Message[]* that is signed by the node owning *MaliciousID* using *PublisherPu*. If any message is invalid, discard *Message[]* and terminate.
8. If the messages in *Message[]* are valid, from the same node, and contain conflicting decisions for the same operation, {*Certificate*}*MaliciousPr* is revoked.