UNIVERSITAT
ROVIRA I VIRGILI

Universitat Rovira i Virgili
Department of Computer Engineering and Maths

Mitigating Routing Attacks with Local Trust in DHTs

A thesis submitted for the degree of

Master in Computer Engineering and Security

Author:

Raúl Gracia Tinedo

Advised by:

Dr. Pedro García López

Dr. Marc Sánchez Artigas

Department of Computer
Engineering and Maths
Universitat Rovira i Virgili

2011 May

Abstract

Distributed Hash Tables (DHTs) have been used as a common building block in many distributed applications, including Peer-to-Peer (P2P), Cloud and Grid Computing. However, there are still important security vulnerabilities that hinder their adoption in today's large-scale computing platforms. For instance, routing vulnerabilities have been a subject of intensive research but existing solutions are mainly based on redundancy.

Regarding redundancy, there exist well-studied techniques that strengthen DHTs against routing attacks. The trouble is that the introduction of redundancy increases communication costs and might significantly reduce scalability. Therefore, it seems reasonable to focus on improving the *quality of routing paths* in terms of forwarder reliability.

In this thesis, we present *Sophia*, a novel and generic security technique which combines iterative routing with local trust to fortify routing in DHTs. *Sophia* strictly benefits from first-hand observations about the success/failure of a node's own lookups to improve forwarding paths. Moreover, unlike redundant routing, *Sophia* dynamically protects routing without introducing additional network overhead. To the best of our knowledge, this is the first work which exploits a local trust system to fortify routing in DHTs.

We compared the performance of *Sophia* with redundant routing techniques in the Kademlia DHT. Our simulation framework considers both stable and dynamic scenarios as well as several threat models. In our simulations, *Sophia* obtained significant improvements regarding routing resilience, self-adjustment and network traffic reduction compared with traditional routing redundancy techniques.

Agradecimientos (Acknowledgements)

La presente tesis es el resultado de una etapa de trabajo duro, pero también de gran crecimiento personal y profesional. Por supuesto, todo esto no hubiera sido posible sin la ayuda de las personas más cercanas a mí.

En primer lugar, me gustaría agradecer sinceramente a mis directores de tesis su ayuda y guía durante el desarrollo de esta tesis. Agradecer de corazón a Pedro García su confianza por haberme dado la posibilidad de entrar en el mundo de la investigación. Además, es para mí muy importante mencionar la libertad para pensar y proponer que Pedro me ha dado durante el desarrollo de esta tesis, ayudándome así a aprender y madurar en muchos aspectos. A Marc Sánchez, me gustaría dedicarle también unas palabras de sentido agradecimiento: su consejo, visión y capacidad han hecho de los momentos difíciles meras lecciones de aprendizaje, sin mayor trascendencia; un saber hacer al alcance de muy pocos.

En segundo lugar, me gustaría agradecer a todos los compañeros del grupo Arquitectures i Serveis Telemàtics (AST) los buenos momentos que hemos pasado durante más de dos años. No me es molestia enumerarlos a todos: Marc Espelt, gracias por hacer que las innumerables horas en el laboratorio sean divertidas y enriquecedoras. Lluís Pàmies, gracias por compartir tu capacidad y punto de vista con los demás, consigues que todos seamos mejores cada día que pasa. A Jordi Pujol, siempre dispuesto a ayudar con una sonrisa, algo que es sin duda un tesoro en estos tiempos. A Helio Tejedor, gracias por pensar de forma tan diferente al resto, solo tu humanidad supera lo buen profesional que eres. A Sandra Ferrer, siempre dispuesta a colaborar sin esperar nada a cambio, gracias por compartir el duro final de máster conmigo. Y por supuesto, gracias a otros compañeros como Rubén Mondéjar, Enrique Fernández y Carlos Anglés por los buenos momentos que hemos compartido.

Finalmente, me gustaría mostrar mi más alta gratitud a mi familia en general, y especialmente a los más cercanos que siempre han apoyado cualquier decisión que pudiera tomar. Muchas gracias también a mis amigos por estar ahí: Juan Héctor, Marc, Víctor, Ana Belén, Fátima, Bea,... y, afortunadamente, la lista podría continuar.

Gracias a todos por hacer esto posible,

Raúl Gracia.

Contents

List of Figures

List of Tables

1

Introduction

In the last decade, Peer-to-Peer (P2P) systems have aroused a remarkable interest from private corporations, users and researchers. Prior to delving into the universe of P2P technologies, we believe that giving a general definition of a P2P System is mandatory:

> *"P2P systems are a type of distributed applications that manage the resources –storage space, CPU cycles, files, network bandwidth– provided by the users at the edges of the Internet to their benefit."*

The earliest P2P systems, such as Napster and Gnutella [1], were devised to provide file-sharing services. These popular applications made the free exchange of resources possible to users for their own benefit, giving an alternative to the traditional client-server architecture. This fact caused a revolutionary paradigm shift in the field of distributed systems.

However, pioneering P2P systems exhibited important scalability problems as increasingly large amounts of users employed them as a common service. On the one hand, Napster-like systems (1_{st} generation), relied on a centralized server to index resources shared by users (also called *peers*). This architectural design suffered from the single point of failure problem.

On the other hand, unstructured P2P systems such as Gnutella (2_{nd} generation), removed the necessity of a centralized index server by delegating the responsibility of indexing shared contents to users. Although unstructured P2P networks were considered to be the first representative example of fully decentralized distributed systems, their search scheme was extremely inefficient. Typically, object location was achieved by performing a scoped flooding search. This fact represented a twofold shortcoming to unstructured overlays: First, since the *flooding's scope* was a predefined parameter, rare objects were difficult to find because queries often reached such a bound before locating an object replica (ineffective). Second, flooding-based search produced an exaggerated network overhead in each search (inefficient). Such a cost was prohibitive taking into account that P2P networks are built in the *application layer* of the network stack.

Despite important efforts, the research community realized that designing *efficient and effective* search algorithms on top of unstructured P2P networks was a complex task [2]. For this reason, researchers considered the possibility of constructing *structured P2P networks*, also known as *Distributed Hash Tables* (DHTs): the 3_{rd} generation of P2P systems. This new class of overlay network organizes nodes in a particular way, so that every node can always find a path to any other node. Additionally, this is achieved incurring a relatively low cost in terms of network overhead and resolution time. To this end, every node is labeled with a value, called *ID* or *key*, which identifies uniquely a node within the system.

In a DHT, keys are distributed among a potentially very large amount of nodes. However, due to the overlay structure, each node needs to contact only a small subset of other nodes. This reduced neighborhood is sufficient to deterministically route a query towards the responsible node for the requested key. Generally, whereas the neighborhood maintenance cost of each node is limited, queries in the network can be resolved within $O(logN)$ hops, where N is the number of peers in the network. This is the key factor behind the virtually limitless scalability of DHTs.

DHTs inherently provide self-organization, fault-tolerance, flexibility, and scalability as well as ease of data access and storage. DHTs rely on the distribution of workload and responsibility among the entire network, where all the participants hold identical roles. In this sense, DHTs represent a rich substrate to deploy large-scale decentralized applications and services. For instance, open DHTs deployed for file-sharing applications such as eMule's Kad [3] or Bittorent Mainline DHT [4] support millions of concurrent users at any moment [5], [6]. Furthermore, the actual impact of DHTs deserves a special mention if we realize that they are integrated in many other massive distributed systems [7], [8], [9].

Mainly, applications benefit from DHTs by using two fundamental operations [10]: $value \leftarrow get(key)$ and $put(key, value)$. These operations are devised to efficiently store and retrieve $< key, value >$ pairs from the overlay. However, these operations are exclusively based on the underlaying DHT routing service, that is, the *lookup service*.

Broadly speaking, the lookup service is a distributed algorithm, executed by a set of nodes, intended to find the responsible node for a certain key. For instance, in case an application executes $get(k)$ for obtaining the associated data object to key k, the DHT first needs to locate the responsible node n_k for key k. To do so, the requester node starts the lookup procedure, by executing $lookup(k)$, which will progressively find *closer nodes* to the requested key, until locating the node n_k. Note that, in the majority of cases, requester node's neighbors will not be the closest ones to its requested

keys. This means that, when a neighbor of the requester node receives a lookup request, it also should cooperate by forwarding the incoming request to any other closer neighbor to key k, if it does exist. Therefore, due to the intrinsic multi-hop nature of DHT routing, lookup requests are forwarded through a bounded number participants, until reaching the targeted destination.

1.1 Problem Statement

The local view of DHT nodes, while capital for achieving scalability, make DHTs vulnerable to the presence of malicious participants in an open environment such as the Internet. In this regard, security has become a primary concern to protect DHTs from malicious attackers, which can join the network and disrupt all operations by intentionally disobeying protocol specifications [11]. Moreover, if only a small set of nodes fails either to serve requests or to follow protocol specifications, the integrity of a very large-scale system may be compromised. One critical aspect is message forwarding [12]. For example, a malicious peer may simply drop all messages passing through it, even though these messages are cryptographically secure.

A wide variety of routing security techniques has been devised to protect routing in DHTs against dishonest participants [11], [13]. Nonetheless, the majority these techniques for DHTs rely on some form of *redundancy*: multiple routing paths and/or data replication. Although extensive research has been devoted to design effective redundancy approaches, the concept of redundancy itself yields additional network overhead. Such an overhead is non-negligible and represents a real scalability shortcoming for large-scale systems intended to provide reliable routing services. Consequently, it seems reasonable to focus on improving the *quality of routing paths* in terms of forwarder reliability.

Hence, our efforts in this thesis have been driven to involve the concept of *routing quality* in the design of a novel routing security technique against malicious nodes.

1.2 Goals

The solution devised in this thesis achieves the following goals:

- **Generic and Scalable Solution**. The network robustness should be offered in an abstract way. In other words, the network robustness should not depend on the attack or the overlay employed. Additionally, the solution proposed should be scalable and not incur extra communication overhead.

- **Pure Decentralized Solution**. By definition, a P2P System is a decentralized architecture. Therefore, a successful solution must be fully compliant with this requirement. Any centralized security approach —e.g. Central Certification Authority— becomes a scalability barrier, a performance bottleneck and an attack-prone system component.

- **Design Simplicity**. The solution adopted should avoid unnecessary complexity in order to be applicable in real and large-scale systems —millions of users and resources.

1.3 Contributions of this Thesis

The contributions of the present thesis are depicted as follows:

- **Analysis of the state of the art.** Before starting the design of our system, we make a concise study on the state of the art which is reported further in this manuscript.

- **Design of *Sophia*, a novel routing security technique for DHTs.** The main contribution of this thesis is *Sophia*: Securing Overlays with a Personal HIstory Approach. *Sophia* is a novel and generic security technique which combines iterative routing with local trust to fortify routing in DHTs. *Sophia* strictly benefits from first-hand observations (e.g. lookup success/failure) to qualitatively improve forwarding paths. Moreover, unlike redundant routing, *Sophia* dynamically protects routing without introducing additional network overhead. To the best of our knowledge, this is the first work which exploits a local trust system to reinforce routing in DHTs.

- **Validation of our approach in a rich variety of scenarios and threat models.** Our system has been validated in a wide spectrum of hostile scenarios. For instance, we defined and implemented several attack models (individual and coordinated) performed by dishonest participants. In addition, we analyzed the performance of our system under dynamic node participation (i.e. *churn*). Finally, we assess the impact of attackers with non-persistent identities; a critical aspect of any kind of trust system.

1.4 Thesis Structure

The remaining part of this thesis is structured as follows. In Chapter 2, we present the necessary background, concepts and definitions used through the remainder of the thesis. First, we review the main features of P2P systems, giving special mention to Kademlia DHT [14], the overlay we use to demonstrate the utility of our solution. In second place, we discuss the main attacks and defense techniques regarding routing in DHTs. Finally, we introduce the concepts of *trust and reputation* in P2P systems and the relevance of node identities in these settings.

In general terms, Chapter 3 is intended to expose the design of our solution, *Sophia*. We deeply detail the architectural nuances of our system. In addition, we validate *Sophia* comparing its performance with redundant routing, in the presence of attackers. In this line, Chapter 4 provides a broader view of the performance of *Sophia* against several attack models and scenarios. Moreover, this part of the thesis reveals the behavior of *Sophia* under dynamic node participation and attackers with non-persistent identities.

Finally, we draw some conclusions in Chapter 5.

1.5 Publications

The results of our work are reflected in the following publications:

- **Raúl Gracia Tinedo**, Pedro García López, Marc Sánchez Artigas. *"Sophia: Local Trust for Securing Routing in DHTs"*. In IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'11). May 23^{th}-26^{th}, 2011, New Port Beach, CA, USA. (***CORE Rank A***).

- **Raúl Gracia Tinedo**, Pedro García López, Marc Sánchez Artigas. *"When Identities Persist on P2P Lookup Services: A Vision"*. In Jornadas de Concurrencia y Sistemas Distribuidos (JCSD'10), June 9^{th}-11^{th}, 2010, Vall de Núria, Spain. (*Positional Paper*).

- **Raúl Gracia Tinedo**, Pedro García López, Marc Sánchez Artigas. *"On the Limits of Securing Routing with Local Trust in DHTs: When Churn and Misbehavior Play Together"*. (Submitted for publication to *Future Generation Computer Systems Journal* (1^{st} Q.)).

2

Background and Related Work

2.1 Peer-to-Peer Networks

We discern between unstructured and structured P2P systems. First, we will briefly focus on unstructured P2P systems. Subsequently, we introduce structured P2P systems, giving special emphasis to Kademlia DHT [14]; the overlay employed to materialize and validate our approach.

2.1.1 Unstructured Peer-to-Peer Networks

Unstructured P2P Networks were the first type of completely self-organizing and distributed systems to appear for bringing together user resources. In unstructured P2P overlays, such as Gnutella [1], peers use flooding or random walks to resolve queries. This design has been claimed as highly resilient and fault tolerant due to the inherently random network structure and the neighbor set distribution, namely *degree* distribution, of peers [15]. However, this approach for resolving queries does not scale satisfactorily, since its cost in messages presents a linear growth with the size of the network.

Although unstructured P2P systems present interesting network properties –e.g. small-world phenomenon, scale-free degree–, they have been discarded for building large-scale networks because their costs and search indeterminism. Therefore, we will not further consider unstructured P2P systems in this thesis.

2.1.2 Structured Peer-to-Peer Networks

Structured P2P Networks, commonly referred as *Distributed Hash Tables* (DHTs), emerged to tackle with the unaffordable shortcomings of their unstructured counterparts.

DHTs offer the same functionality of a traditional hash table but associating key-value mappings with participating nodes instead of hash buckets. Such functionality is basically supported by two fundamental operations: $value \leftarrow get(key)$ and $put(key, value)$. These operations are devised to store and retrieve $< key, value >$ pairs from the entire overlay, in an efficient manner. Both data objects and participants are

labeled with a unique identifier, called *ID* or *key*, randomly selected from the identifier space. DHTs resort to standard hash functions, such as MD5 [16] or SHA-1 [17] for producing unique identifiers. Hash and cryptographic functions produce m-bit hash values, where normally $m = 128$ or larger. Consequently, the DHT identifier space ranges from $I = [0, 2^m)$.

An important point related with the identifier space is the concept of *distance*. DHTs define a *distance metric* which is used to greedily route towards a key. There exist several distance metrics depending on the DHT design: for instance, CAN [18] uses the Euclidean distance to measure closeness among keys. Other systems, such as Pastry [19], Tapestry [20] and Kademlia [14] use common prefix length as distance metric, whereas Chord [21] employs the clockwise distance between identifiers on the circle $[0, 2^m)$ modulo 2^m.

Usually, the identifier space I is several orders of magnitude larger than the number of participants. For this reason, I is properly partitioned among existing participants and each participant is responsible for a distinct partition. In this regard, a fundamental concept related with DHTs is the *consistent hashing* [22]:

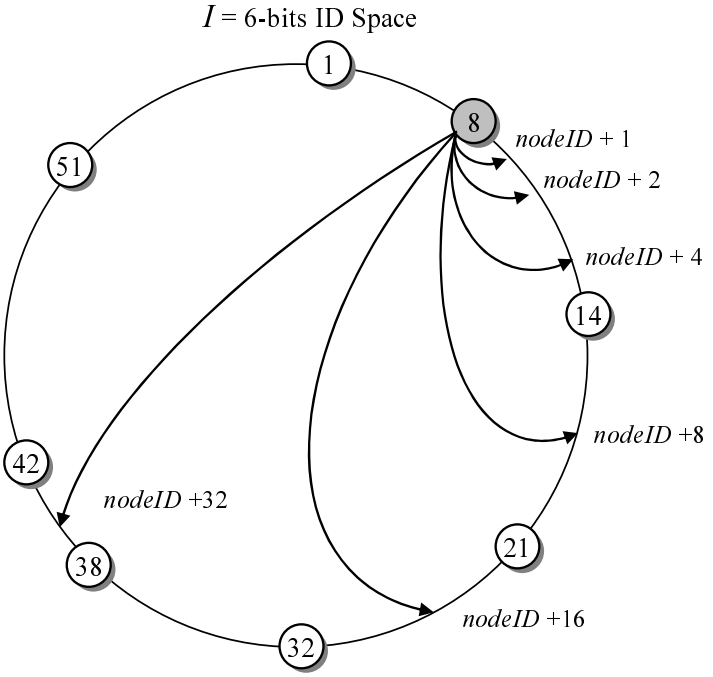
Definition 2.1 (Consistent Hashing) *For any set of N nodes and K keys, we have with high probability that:*

- *Each node is responsible for at most* $(1 + \epsilon)\frac{K}{N}$ *keys.*
- *When a new node* $(N + 1)$ *joins/leaves the network,* $O(\frac{K}{N})$ *keys need to be reallocated. This reallocation only affects to the joining/leaving node.*

As a consequence of Def. 2.1, DHTs provide *load-balancing*: nodes are responsible for a roughly equally-sized set of keys and, consequently, they will serve a balanced number of requests. In addition, it should be noted that the dynamic node participation has a *localized impact* on the network. In other words, if a node leaves or joins the network, just a small subset of keys is affected and only closest neighbors should deal with such an event, without the intervention of the rest of participants. These properties are capital for deploying DHTs in real, dynamic and heterogeneous environments.

To properly understand how a message can be *efficiently routed* from a node to any destination key, we must focus on the *links* that every node establishes with the rest of participants. An important characteristic of DHTs is that, in general, nodes present *logarithmic degree* with the total number of existing nodes. In other words, nodes in a DHT construct a *routing table* with $O(logN)$ positions, where N is the number of nodes in the network. Clearly, this limited membership is essential for the system's scalability and, of course, the correct construction of this routing table is not trivial.

Conceptually, each entry of a node's routing table represents a link to a certain area of the network, which is at a certain *distance* from that node, depending on the DHT distance metric specified. Typically, these links or entries refer to distance ranges which present an *exponential growth*. To better understand this, we suggest to carefully analyze Fig. 2.1.



N8 Routing Table

Distance	Neighbor
N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 + 16	N32
N8 + 32	N42

Figure 2.1: An example of a Chord overlay.

In Fig. 2.1, we can observe the organization of a Chord overlay. Concretely, the aim of this picture is to illustrate the routing table of a certain node, in this case the table of node $N8$. As aforementioned, it is appreciable that entries refer to exponentially further regions of the network. This fact yields that nodes maintain, in proportion, more links with closer nodes than with distant nodes. This is critical to make the lookup process *converge* (Def. 2.2) and, consequently, succeed. The main reason is that, when a lookup request is getting closer to the key, nodes in this final part of the lookup process hold precise routing information about neighbors surrounding the targeted key.

Definition 2.2 (Lookup Convergence) *Lookup convergence is ensured whenever lookups from two distinct peers in a DHT, heading to a common key k, reach the same responsible node for key k.*

As a consequence of such an elegant routing table design, the network diameter of a DHT is remarkably bounded. Furthermore, in the majority of DHTs, the diameter

of the network is limited to $O(logN)$ hops. That is, a message exchanged between the most distant pair of nodes, will visit with high probability at most a logarithmic number of intermediate forwarders where the population of the network is N.

In this line, it is worth mentioning the efficiency of DHT routing, which is reflected in the lookup service. Considering only local routing information, a DHT node *greedily routes* lookup requests. That is, in each host, lookup requests are forwarded to the *closest neighbor* available in the routing table, given a target key. Taking into account the dispersion property of hash functions, we can assume that lookup requests target uniformly distributed random keys over the identifier space. In this case, it is not hard to see that the *average path length* of routed messages is $\frac{1}{2}O(logN)$, where is N is the total amount of nodes in the network. As can be appreciated, for large values of N, the average number of intermediate forwarders in the routing process is quite limited.

The main features of DHTs can be summarized as:

- **Load-Balancing**: As a consequence of adopting consistent hashing, DHTs uniformly partition the overlay, materializing thus a proper distribution of load and responsibility among nodes. Additionally, fixed sizes of routing tables avoid the appearance of highly connected nodes, namely *hubs*, which represent a performance bottleneck.

- $O(logN)$ **Degree**: The limited membership of nodes in a DHT is an essential aspect regarding efficiency. Nodes only need to check the state of a handful of other participants in order to be properly connected to the network, whereas the network size might be several orders of magnitude larger.

- $O(logN)$ **Network Diameter and** $\frac{1}{2}O(logN)$ **Average Path Length**: By structuring the overlay, DHTs *efficiently and effectively* search objects within the network. The network design allows lookups to rapidly converge towards the desired key, guaranteeing that, given a key k, the responsible node n_k can be always determined.

- **Fault Tolerance**: Due to the overlay partitioning, if a node fails or leaves the system, only the closest neighbors should handle the failure by accepting the responsibility of the orphan identifier partition. This means that the rest of the network remains unaffected by such a node departure. For this reason DHTs are fault tolerant: First, there is no central point of failure (decentralization). And, second, node failures and dynamic participation have only a localized impact on the overlay.

- **Scalability**: Scalability means that a DHT should work efficiently for overlay networks of arbitrary sizes. As a consequence of previous properties, DHTs have salient scalability guarantees, even in heterogeneous and open environments.

Once reviewed the general features and properties of DHTs, we would like to introduce in more detail a particular one called Kademlia [14]. Due to its popularity in open environments [5], [23], we employed this overlay to test our approach in a wide variety of scenarios. Therefore, a deeper knowledge of Kademlia's design is strongly recommended.

2.1.3 Kademlia

Kademlia [14] is largely the most deployed *Distributed Hash Table* in the Internet today (e.g. Bittorrent, eMule Kad, OverNet). Kademlia is a structured overlay devised by Mazières et. al., which has several specific and desired features as a result of using the XOR metric to express closeness among identifiers. For instance, it should be noted that the XOR metric is a symmetric operation. More formally, let $d(x, y)$ be the XOR distance between identifiers x and y, thus it is not hard to see that $\forall x, y : d(x, y) = d(y, x)$. Albeit trivial, this property yields an important consequence: Kademlia nodes receive lookup queries from nodes which are also candidates to be inserted into their own routing tables. Hence, nodes benefit from incoming messages to efficiently build and maintain their routing tables. Another interesting property consists of the unidirectionality of the XOR operation. In other words, for any x there exists exactly one y which is at distance $d(x, y)$. This fact ensures that lookups targeted to the same key converge along the same path, irrespective of the source node. This property can be effectively exploited for caching purposes.

More technically, Kademlia is a prefix-matching DHT characterized by the use of the XOR metric. The XOR metric measures the distance between two nodes as the numeric value of the exclusive OR of their IDs. Each Kademlia node has a random m-bit and maintains a routing table of $\mathcal{O}(m)$ k-buckets. In every k-bucket, there are at most k entries, each leading to any node within XOR distance $[2^i, 2^{i+1})$ from itself, where k is a redundancy factor for tolerating $k-1$ routing failures. Each k-bucket entry is formed by a `<IP,UDP port,ID>` triple. Given a key K, the original Kademlia routing protocol iteratively queries α ($\alpha \leq k$) users with a `FIND_NODE` RPC for the closest k nodes to key K according to the XOR metric. In each step, the returned candidates from previous RPCs are merged into a sorted list from which the next α nodes are

chosen. The procedure is repeated until the node responsible for key K is found. In this regard, the lookup procedure is deeply detailed later on (see Section 2.2.2).

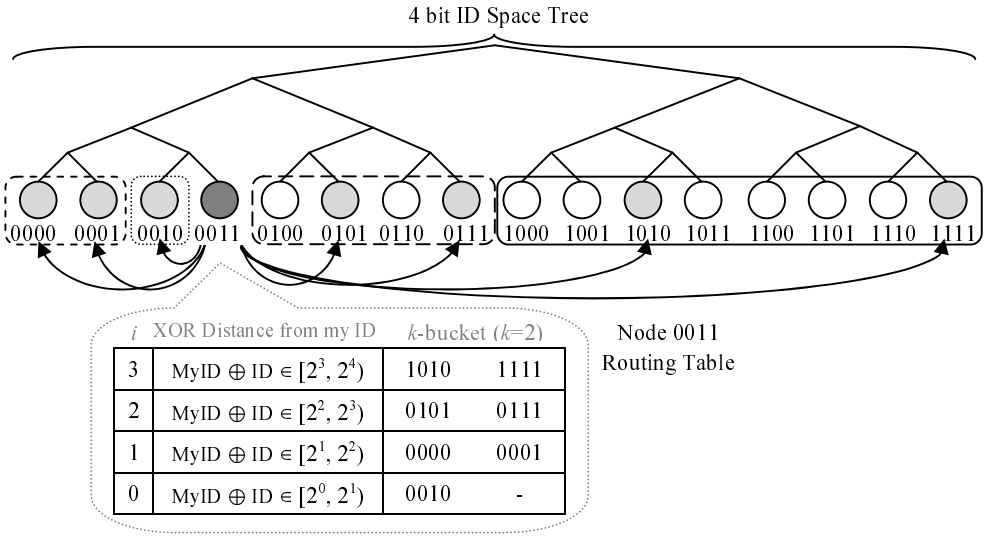


Figure 2.2: Example of a Kademlia node routing table within a full overlay tree.

Kademlia has been designed to be resilient to dynamic scenarios. To this end, the protocol defines two algorithms: First, a maintenance algorithm in order to keep fresh the routing table. This algorithm is composed by a k-bucket replacement policy (Least-Recently Used evicted) and a Ping/Pong availability test. Concretely, when a Kademlia node receives any message from another node, it executes this algorithm as follows: If the sending node already exists in the recipient's k-bucket, the recipient moves it to the tail of the list. In case the node is not already in the appropriate k-bucket and the k-bucket has fewer than k entries, the recipient simply inserts the sender at the tail of the list. However, if the bucket is full and the sender is a new contact, the recipient pings the k-bucket's first element, which is the least recently contacted node. If the least-recently contacted node fails to respond, it is evicted from the k-bucket and the new sender is inserted at the tail. On the contrary, if the least-recently contacted node responds, it is moved to the tail of the list and the new contacted node is discarded.

Second, each node refreshes any bucket to which it has not performed a node lookup for a certain period (stabilization). When authors in [14] say "refresh", they refer to perform a lookup to a randomly chosen ID belonging to the unused k-bucket range. For an exhaustive specification of the protocol see [14].

2.2 Routing Attacks and Countermeasures

2.2.1 Overview of Attacks Against DHTs

The objective of this thesis is to design a novel security technique to strengthen routing in DHTs. Nonetheless, in our view, it is capital to understand the wide variety of threats that a DHT is exposed to, specially in open environments. Despite this thesis is focused on *routing attacks*, we believe that a general background on attacks is recommended.

2.2.1.1 Sybil Attack

Regarding DHTs, the Sybil attack was first studied by Douceur [24] and it aroused a huge interest from researchers. Essentially, the Sybil attack exploits the fact that, in an open environment, there is no central management of system identities. On the contrary, each participant is responsible for creating its own identifier.

Normally, open peer-to-peer systems do not guarantee that each identifier represents only one physical entity. Thus, a malicious participant could create a large number of system identities. From the network point of view, these artificial identities are treated as regular nodes. In other words, these identities represent logical nodes, despite there is only one physical node behind them. Thus, these logical nodes will be inserted into other's routing tables, will receive lookup requests, etc., as any other node. Clearly, the main danger here lies in the fact that a single physical adversary is able to create a large fraction of artificial nodes, increasing the impact of an hypothetical attack. Furthermore, attacks may be much more effective if these logical nodes are orchestrated, or, in other words, if they perpetrate a *collusion attack*.

Actually, the Sybil attack itself does not disrupt the correct operation of the overlay. However, it is a mean to execute many other types of attacks such as polluting other's routing tables and censor the access to certain keys.

The main conclusion of Douceur's work is that, in an open P2P system, a central and trusted identity management is the only practical way of ensuring the one-to-one correspondence between identities and physical entities. Nonetheless, a plethora of research works have been focused to devise effective approaches (also distributed) to alleviate the effects of Sybil attacks [11], [25].

2.2.1.2 Eclipse Attacks

The Eclipse attack, also called routing table poisoning, is a malicious behavior where adversaries attempt to inject bogus nodes into other's routing tables. Since a node

maintains just a handful of contacts in its routing table, if a significant fraction of these contacts is corrupted, such node becomes isolated from the overlay. Consequently, incoming and/or outgoing requests related with the eclipsed node can be freely manipulated by the fraction of its malicious neighbors.

Sit and Morris [26] were pioneers on the study of this attack. Actually, they discussed about the important hazard that Eclipse attacks represent in DHTs without any kind of neighbor verification.

Eclipse attacks might be executed in several ways, depending on the overlay object of attack. For instance, Chord restricts the routing table construction by fixing in each entry the identifier that a node's neighbor should hold. Therefore, to successfully eclipse a Chord node would require to create a set of bogus nodes with IDs corresponding to the victim's routing table entries. In case of prefix matching overlays, such as Pastry [19] or Kademlia [14], the eclipse attack is considerably easier. That is, an attacker could create a set of common prefix nodes of only a few digits. This increases the number of spurious routing table updates that an attacker could supply to eclipse nodes in a region of the network.

There is no standard approach to perform Eclipse attacks. However, the most common attack model assumes a set of coordinated nodes intended to corrupt the routing table of as many honest nodes as possible. Thus, routing updates provided by a malicious participant to honest nodes will refer to other malicious participants. Nevertheless, the interest of adversaries could be, for instance, to eclipse just a small subset of peers to deny the access to a handful of specific keys.

Conceptually, defenses to Eclipse attacks [27], [11] can be considered successful if the fraction of malicious entries in the routing tables of honest nodes is similar to the fraction of malicious nodes in the system. However, normally, these defenses are complemented with *routing redundancy*. That is, in these settings, routing using a single path is in general not enough to ensure acceptable routing success guarantees.

2.2.1.3 Routing and Storage Attacks

As previously described, the Sybil attack is a way to maximize the influence of a physical attacker in the overlay by creating a set of artificial identities. By launching a Sybil attack, bogus nodes can be deliberately placed into other's routing tables, isolating the victims from the rest of the network –i.e. Eclipse attack.

However, these attacks by themselves do not disrupt the correct operation of the DHT; they are means to perform actual actions against the DHT service, that is, against

the correct management and location of $< key, value >$ pairs in the overlay. Actions against the DHT service can be classified in two categories:

- **Storage Attacks**: A malicious participant could join and route correctly, but it could deny the existence of data it was responsible for. In a similar fashion, it might claim to actually store data when asked, but then refuse to serve it to clients or serve incorrect data.
- **Routing Attacks**: An individual malicious node could forward lookups to an incorrect or non-existent node. Additionally, it simply may drop any lookup request passing though it. Since the malicious node would be participating in the routing update system in a normal way, it would appear to be alive, and would not ordinarily be removed from the routing tables of other nodes. Thus, retransmissions of failed lookups would also be sent to the malicious node.

There is a vast amount of research works devoted to mitigate the effects of this kind of attacks [26], [11]. The most relevant ones, from our viewpoint, are mentioned in our Related Work (Section 2.5). However, the majority of these works rely on the concept of redundancy as a mean to increase the probability of routing successfully. To better understand the concept of routing redundancy, the following section is aimed at depicting a few of paradigmatic approaches in this regard.

2.2.2 Routing Redundancy Security Techniques

In a structured overlay each node is responsible for serving data items and correctly routing messages. Malicious nodes may abuse these responsibilities and act to disrupt the correct operation of the system. Because of the multitude of potentially unforeseen vulnerabilities, it is important to introduce robustness into the system so it has the ability to work properly, even in the presence of unexpected attacks.

Generally, defenses against storage and routing attacks are based on two mechanisms: (1) redundant storage and (2) redundant routing. The first defense mechanism safeguards and makes data objects available by *appointing multiple nodes to store object replicas*. Therefore, regardless the presence of malicious or disconnected nodes, a proper data replication ensures that with high probability at least one replica is reachable. Consequently, an extra maintenance overhead (e.g., synchronization, additional messages) is introduced to correctly maintain each object replica. The design of efficient replication techniques against storage attacks has been object of intensive research [28], [29].

In a similar fashion, the second defense mechanism consists of strengthening routing by simultaneously *sending a lookup message in parallel through several distinct neighbors*, namely, *lookup paths*. Clearly, although a few forwarders fail to serve the lookup request, with high probability at least one lookup path will succeed in finding the responsible node for a certain key. There are several well-known routing redundancy techniques which propose elaborated designs for providing effective routing redundancy. To better understand this, we describe three paradigmatic redundancy approaches (see Fig. 2.3):

Independent Paths: Cyclone. Two paths are said to be *independent* if they share no common node other than the source and the destination. The argument of *independent paths* technique is that, if lookup paths do not pass through common intermediate forwarders, a single malicious participant cannot disrupt more that one path. This fact increases the probability of routing successfully when faulty nodes are present. Note that independent paths are built to reach a single replica. Therefore, this technique exclusively focuses on routing attacks, although it can be easily combined with some form of object replication to combat storage attacks.

As a practical example, Artigas et al. propose Cyclone, an equivalence-based routing scheme deployed over an existing structured peer-to-peer overlay [30]. This technique enhances the already existing non-independent multiple path approach presented by Castro et. al. [13]. Cyclone, based in Chord, partitions the identifier space $I = 2^m$ into several clusters. A Cyclone node identifier is divided in the cluster identifier, which corresponds to the p rightmost digits, and the intra-cluster node identifier, that is, which corresponds to the $m - p$ leftmost bits. Consequently, a single Chord overlay is virtually fragmented into $c = 2^p$ distinct clusters. Nodes fill up their routing tables with nodes of their own cluster. However, the successor list does not follow this restriction. Such differentiation between routing table and successor list permits inter-cluster lookups. That is, a node may request a key outside its own cluster. Thus, the cluster's closest node to the key will forward the lookup to an even closer node existing into its successor list, irrespective of whether the node belongs to the same cluster or not. In this technique, independent paths are achieved by routing through the c distinct clusters.

Wide Paths: Kademlia Redundant Routing The concept of *wide paths* has been proven as a really effective routing redundancy technique [31]. The main strength behind the concept of wide paths is that, in general, appointing α nodes to forward

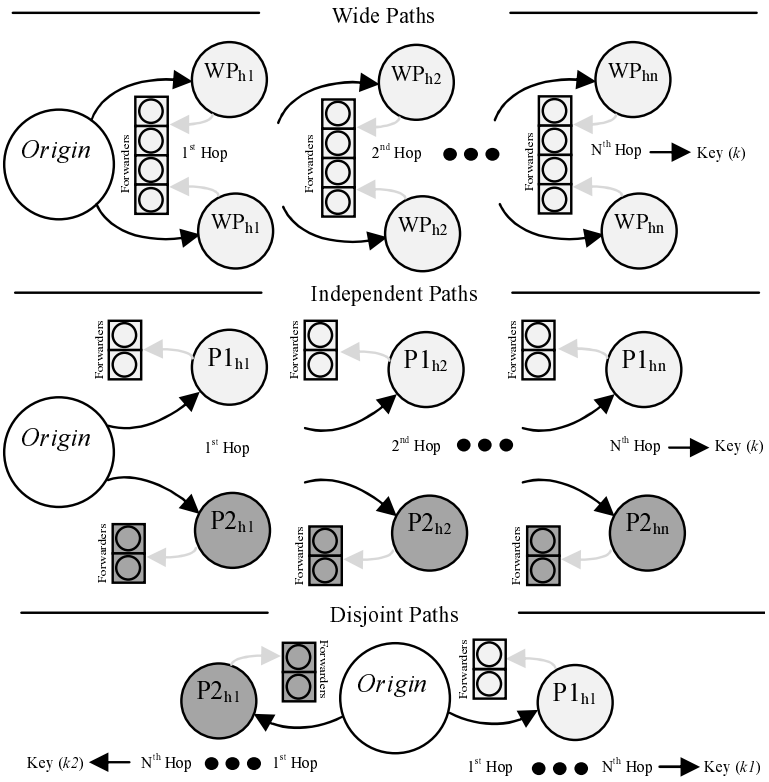


Figure 2.3: Standard routing redundancy techniques over an iterative routing scheme.

the same path is more effective than having each node forwarding a separate path. Intuitively, a wide path resists $\alpha - 1$ failures *at each step*. In [31], authors argue that this scheme is simple and it gives a much greater fault tolerance than the multiple path technique, which provides asymptotically bounded routing success guarantees as depicted in Section 3.2.2.

Kademlia redundant routing, as an example of the wide paths approach, works as follows: Suppose that a certain node A is interested in contacting the responsible node for key q. Thus, A starts a lookup sending α parallel paths to the closest candidates to the key q. Note that these candidates belong to the closest k-bucket to the destination key or, in other words, to the k-bucket which minimizes the XOR distance between a and q. In each lookup step, the set of k candidates returned from each previously sent parallel path ($k \cdot \alpha$ candidates in total) are merged into a list, ordered by XOR distance to the destination key. Thus, given this new list of candidates, the originator node sends again α parallel paths using the closest obtained candidates to the destination key q. This process is repeated until the key is found or there are no closer forwarders to the requested key.

Disjoint Paths: Equally-Spaced Replicas. The disjoint paths approach seeks to tackle with both routing and storage attacks. That is, disjoint paths only share the source node among them. This means that, once d parallel paths are sent, they presumably will be forwarded by distinct participants and will reach distinct object replicas. Although the routing costs are similar to previous techniques, this approach yields object replication costs since it provides a defense against storage attacks.

Equally-Spaced Replicas (ESR) technique [28] is a representative example of the *disjoint paths* approach. They state that this method is more reliable than the standard method of placing replicas at consecutive locations because the replicas are accessed using diverse routes instead of a single route. ESR exploits key replication to produce disjoint routes and, consequently, increase the probability of routing successfully. Concretely, this technique creates ω disjoint routes by equi-spacing $2^{\omega-1}$ replicas over the overlay identifier space. The main strength of this approach lies in the concept of disjoint paths: lookup paths simultaneously start from different routing table entries targeting different replica destinations. This fact ensures that any pair of lookup paths share no other common node than the originator node, increasing then routing resilience against the appearance of dishonest forwarders.

Definitely, these techniques significantly increase the routing success probabilities of a system. However, they also introduce an important network overhead. We strongly believe that considering the quality of forwarders in routing paths is an approach to reinforce routing avoiding additional network costs. However, for relating a node to its behavior we first need to persistently identify it.

2.3 Identity in Peer-to-Peer Networks

One of the most challenging and important problems for securing DHTs is the robust and secure assignment of node identifiers. This is important to avoid many attacks where nodes can place themselves in strategic positions of the overlay and damage the whole network. In fact, different works in the literature [13], [11], [32], [33] have agreed that the problem of secure assignment of node identifiers is key for achieving security in DHTs.

In [13], authors present a solution called certificate IDs where the assignment of identifiers is delegated to a trusted certification authority (CA). The CA is responsible of assigning IDs to principals and to sign ID certificates using public key cryptography. The CA assures controlled bootstrapping to the overlay and it avoids strategic positioning of malicious nodes and the well-known sybil attack.

This solution is widely used in many works in the literature [34], [35] but it has a main drawback: the dependence on the CA for bootstrapping. Although it is true that the communications between nodes are not mediated by the CA, the bootstrapping is completely dependent on this centralized external entity. Recent approaches tried to decentralize the certificate authority and bootstrapping process [36], but they propose complex solutions that are difficult to adopt in real settings.

The aim of this thesis is to provide simple solutions with minimal overhead that can be adopted by nowadays peer-to-peer overlays. For these reasons, we are mostly interested in solutions that do not require centralized authorities or complex protocols involving considerable overhead. An interesting solution is to enable distributed ID generation using self-certified identifiers. Following the PGP model, each node generates its own key-value pair (public and private keys). The node could use the public key or a hash of the public key as its own ID. Self-certified certificates permit each node to generate its own identifier, so the Sybil attack is still possible. But at least these certificates ensure non-spoofable identities that can be used as persistent identifiers. Furthermore, the generation of the public and private keys also limits the strategic positioning of malicious nodes due to the computation cost.

In [13] they present a distributed ID generation solution based on self-certified IDs and crypto puzzles. The crypto puzzles are used to moderate the rate at which attackers can acquire IDs. They also propose to bind IP addresses with IDs to avoid attacks that exploit network locality. Nevertheless, authors recognize that this solution cannot prevent an attacker from acquiring a large collection of IDs.

In [32], authors present a similar solution to [13] using self-certified IDs and crypto puzzles in the Kademlia overlay [14]. Authors provide solutions to three problems: they use certified IDs and crypto puzzles to avoid collusion of malicious peers, they extend the Kademlia routing table with a sibling list to fortify the protocol against storage attacks, and they propose a lookup algorithm with multiple disjoint paths to defend against routing attacks. This work is close to our proposal because it aims to create a practicable approach to secure routing in a widely used overlay (Kademlia). Nevertheless, although they use self-certified IDs, they do not really benefit from the persistence of the identifier. Instead of that, their extensions to Kademlia like disjoint paths could even work with ephemeral identifiers and avoid the burden of certificates and crypto puzzles. We believe that persistent identifiers can be used to improve the protocol and make the overall network more robust to routing or storage attacks.

Other works that use self-certified identifiers are [37] and [38]. In both cases, the goal is to maintain a persistent identity that can be used to communicate with that

user. In [37] the identity of every user (ID) is the public key, and ID $\rightarrow$ IP, port mappings are stored in the DHT. The main use of identities is to enable communication between users that have exchanged their keys. Thanks to this system they can detect when other users are on-line. In [38], the goal is also to use the DHT as a directory of identities secured using self-certification. The identity is calculated as a hash function of the concatenated values of IP, date and time and a large random number. They use the DHT to store a tuple containing the ID, the public key, a timestamp, and a signature of this information with the private key. Since the DHT is used to distributed public keys, its security is enforced with a quorum-based query scheme.

Our conclusion from previous works is that they use self-certified identifiers as pseudonyms or user identifiers that are stored in the DHT. They serve as persistent identifiers to communicate with specific users. But we believe that, before storing sensible data in the DHT, the security of the system should be guaranteed. In our case, self-certified IDs will be used as persistent identifiers to secure the DHT against malicious attacks. We want to keep track of historical transactions with persistently identified nodes to reinforce paths and key sets and make the overall network more robust to routing and storage attacks. Inspired by self-organizing evolutionary algorithms like [39], [40] we want to create networks that evolve thanks to their previous interactions, reinforcing connections with well-behaved nodes. We believe that if the network has learned enough from previous interactions, it will be more robust to a broad range of security attacks.

2.3.1 From Redundant Routing to Persistent Identifiers

DHTs has been engineered to operate with only a small number of links per node. While this is positive in terms of scalability, it leaves honest nodes vulnerable to attacks that can exploit such limited view of the system such as routing attacks. This property is what allows attacking peers to drop and misroute messages at will, since the detection of disruptive actions is certainly hard with no historical information available. Formally, we can formulate the problem as follows:

Definition 2.3 *Provide a* `lookup` *primitive which makes sure that when a peer injects a message into the network for a key k, that message reaches all legitimate replica roots with high probability (w.h.p.).*

To provide a secure `lookup` primitive, one needs to solve two subproblems:

- The censor problem, which refers to the impossibility of accessing each honest replica root independently of the others; and

- The forwarding problem, which refers to the difficulty of reaching the replica roots.

Concerning the censor problem, observe that DHTs such as Chord and Pastry suffer from this problem. As an illustrative example, consider Chord. For a given key k, it can be easily seen that reaching a replica root $r \neq \texttt{lookup}(k)$ can only be achieved through the predecessor of $\texttt{lookup}(k)$. As a result, the predecessor peer of each key (if compromised) can censor the access to all the other replicas. Formally, the consequence of this is that the probability for a lookup to succeed is upper bounded by $(1-f)^2$, since both $\texttt{lookup}(k)$ and its predecessor must be honest. Note that this bound decreases exponentially when f increases.

The other challenge is to ensure that each intermediate peer cannot drop and misroute a query message in transit to the set of replica roots. Let ξ_i^h denote the event that a `lookup` operation that requires h hops fails at the i^{th} step. Then, for conventional DHTs (Chord, Pastry, Tapestry, Kademlia, etc.), it is not hard to see that the probability of failure can be approximated by

$$\Pr\{\texttt{Failure}\} = 1 - \Pr\left\{\bigcap_{i=1}^{h} \bar{\xi}_i^h\right\} = 1 - (1-f)^h, \tag{2.1}$$

which is asymptotically 1 when $h \longrightarrow \infty$. The consequence of this result was the birth of a group of techniques that tried to improve this bound through the use of redundancy, in the sense of routing over multiple paths. The logic behind this view was to overcome with redundancy the complexity of identifying reliable forwarders, which in turn was tightly related to the persistence of identifiers.

For instance, works like Cyclone [30] and Equally-Spaced Replication [28] made use of independent paths, i.e., the sending of multiple copies of a message over independent routes, to raise the probability of query success. To give some idea of their performance in the wild, the authors of Cyclone showed that 85% of the requests were correctly delivered when attackers controlled 30% of a 1024 node Chord network and peers sent messages using 8 redundant paths. Formally, this can be expressed as follows:

Assume that peer sent messages over exactly d independent paths. Then, the probability of query failure can be diminished by an exponential decay factor:

$$\Pr\{\texttt{Failure}\} \leq \mathcal{B}(0, d, (1-f)^h) = (1-(1-f)^h)^d, \tag{2.2}$$

where $\mathcal{B}(0, d, (1-f)^h)$ denotes the probability of obtaining no successes out of d Bernoulli trials (there are d independent routes), where the probability of success is $(1-f)^h$, i.e., all the intermediate peers on the path are good.

Table 2.1: Optimal d to ensure that $\Pr\{\texttt{Failure}\} \leq \frac{1}{e}$ when $f = \frac{1}{5}$ and $b = 2$.

Network Size n	1,000	2,000	4,000	8,000	16,000	32,000
d	9.31	11.5	14.4	18.0	22.5	28.2

Despite their significant improvement over conventional `lookup` operations, the problem with these techniques is they are not asymptotically fault-tolerant: *they cannot prevent that* $\Pr\{\texttt{Failure}\}$ *becomes asymptotically high for a fixed d as n increases.*

To better understand this, consider that, in order to reach the farthest peer in the overlay, `lookup()` requires $c \log_b n$ hops. Then, it is easy to see that

$$\Pr\{\texttt{Failure}\} = \left(1 - (1-f)^{c \log_b n}\right)^d \approx exp\left(-dn^{c \ln(1-f)/\ln b}\right), \tag{2.3}$$

which implies that to make $\Pr\{\texttt{Failure}\}$ constant, d must be polynomial in n. In particular, if $exp(-dn^{c \ln(1-f)/\ln b}) \leq \varepsilon$[1], d must be at least $\ln(\frac{1}{\varepsilon})n^{-c \ln(1-f)/\ln b}$. Consequently, the use of independent paths might lead to traffic overload of the peers with low bandwidth in the effort of keeping $\Pr\{\texttt{Failure}\} \leq \varepsilon$.

As we show in Table 2.1, a relatively small d is sufficient to ensure $\Pr\{\texttt{Failure}\} \leq \frac{1}{e}$ when $f = \frac{1}{5}$ and $b = 2$ (Chord) for small networks. Unfortunately, for large n, the value of d should be greater than 30, thus imposing a high overhead on the network. In practice, this implies that for redundant routing to be effective, one shall dynamically adapt the number of independent paths to achieve the desired performance with the minimum overhead.

To reduce communication overhead, another option might be to improve the quality of delivery paths. In general terms, this idea can be formulated as follows: *Either iteratively or recursively, select as next hop the neighbor which presumably provides more chances to reach the destination.* Non-surprisingly, at the heart of this approach will lie the uncertainty of how to identify a reliable forwarder with less overhead than redundant routing. In a recent manuscript, it has been shown that this is possible with the aid of a simple trust mechanism [41]. In this paper, the authors propose an innovative routing protocol based on reputation to refine the decision about which neighbor select as next hop. Concretely, the authors use the standard Bayesian model, specifically a beta model, for reputation evolution. The main feature of their protocol is that progression through the identifier space is not completely greedy: *At each hop, the protocol might choose a node farther to the destination but a better forwarder.* With this loose

[1]Note that by setting $\varepsilon = n^{-k}$, this bound is a *high probability* bound, as $\Pr\{\texttt{Success}\} = 1 - \Pr\{\texttt{Failure}\}$.

requirement, their routing algorithm was able to outperform Equally-Spaced Replication [28] by over 200% with similar communication overhead, showing the promising possibilities of this approach.

We share this vision, and we want to examine what might happen if we apply persistent identifiers to ease the detection of reliable forwarders. However, to identify a reliable forwarder is a concern related to the *trust and reputation literature*. Therefore, the design of an appropriate rating mechanism to numerically evaluate the behavior of other nodes poses new challenges and requires specific knowledge. This is the reason why the last part of the background chapter is devoted to provide an overview of trust and reputations systems.

2.4 Trust and Reputation Systems

An increasing popularity of applications and services deployed in an open environment such as the Internet, where mutually untrusted parties are expected to cooperate, originated the development of trust and reputation systems. With the help of trust and reputation mechanisms, applications and users try to *identify* which entities are *cooperating* with the system and which ones are *acting selfishly*, or even *attacking the system*.

There exist abundant research articles and surveys which excellently describe the paradigm of trust and reputation [42], [43], [44]. Thus, our objective is not to extensively survey trust literature, but to provide: (1) An overview of the necessary concepts to properly comprehend the trust mechanism we defined in our solution, (2) a summarized view of the main components of a trust system and (3) a general classification of these mechanisms, depending on their source of information.

Concepts and Definitions

This part of the section aims to ease the proper understanding of the further discussion about trust systems by defining several capital concepts. Therefore, it seems reasonable to start by introducing the concept of *trust* [44]:

Definition 2.4 (Trust) *Trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends.*

Def. 2.4 does not impose any specific approach to calculate a trust value; in the literature we can find several approaches and calculation methodologies [45], [44]. However, it seems clear that the concept of trust only considers the own opinion of

a certain individual of another one. This yields that the exchange of opinions among participants is not involved at all.

Rather, the exchange of trust values among participants in a system is referred in the literature as *reputation* [44]:

Definition 2.5 (Reputation) *Reputation can be considered as a collective measure of trustworthiness based on the referrals or ratings from members in a community or system.*

Again, Def. 2.5 does not mention the way to disseminate and share these trust values among participants. For this reason, a myriad of reputation systems seek to propose efficient and secure ways to distribute trust values within a system [43].

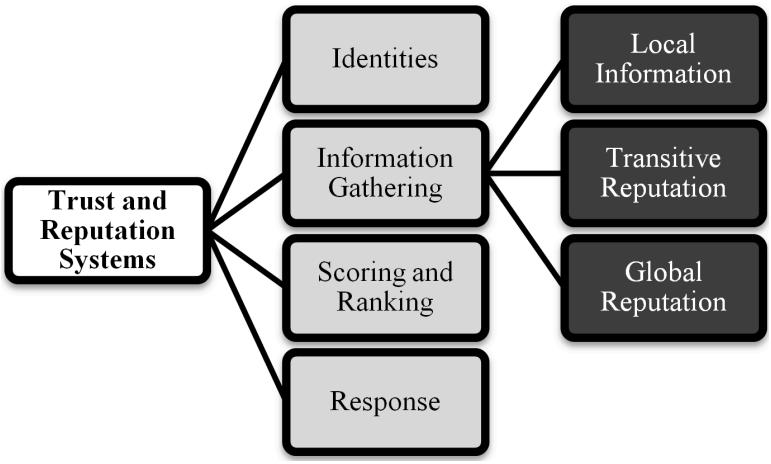


Figure 2.4: The middle column of the figure illustrates the components of trust and reputation systems. The rightmost column classifies trust/reputation systems depending on their source of information.

Trust ratings are built by executing a certain algorithm over a sample of past *transactions* [42]:

Definition 2.6 (Transaction) *Peer-to-peer systems are defined by interactions of autonomous agents or peers. These interactions may include swapping files, storing data, answering queries, or remote CPU usage. We refer to all interactions in general as transactions between two parties.*

We will mention further in this section the coupled relationship between *trust and identity*. For instance, when a node or a group of nodes calculate the behavior of a certain participant, such calculation is related with the identity of this participant in the system. Therefore, if identities are not persistent, any node would be able to change its identifier at any time, making trust calculations related with it invalid. This phenomenon is known as *whitewashing* [42]:

Definition 2.7 (Whitewashing) *The action of purposefully leave and rejoin the system with a new identity in an attempt to disassociate any bad reputation accumulated under the previous identity.*

It would be unnecessary to define concepts such as *cooperate* or *defect*, because they are self-explanatory. However, there is a particular concept related with the classification of a node's behavior that might unfamiliar to the reader [42]:

Definition 2.8 (Strangers) *Peers that appear to be new to the system. They have not transacted with other peers and therefore no trust information on them is available.*

Thus, trust and reputation systems should define policies to tackle with new participants. It is up to the designer to assume good or bad behavior of newcomers (optimistic/pessimistic).

Once reviewed the main concepts and definitions regarding trust and reputation systems, we proceed to define the main components which form any trust/reputation scheme. This will give to the reader a better perspective on the design of our trust-based routing security technique.

Architecture of Trust and Reputation Mechanisms

This point of the background introduces the general architecture of any trust or reputation system:

Identities. Regarding trust and reputation systems, identities are assumed to be long lived, that is, persistent [44]. This fact prevents that non-malicious participants change their identities each time they rejoin the system. Hence, whitewashing is a concern strictly related with malicious individuals.

There are other desirable features of identities, in addition to the persistence. For instance, *spoof-resistant* identities prevent malicious nodes to impersonate other participants within the system. One common solution is the use of a private/public pair of cryptographic keys to verify any communication with another party. In addition, *unforgeable* identities prevent that a single participant could create a large number of identities and, therefore, perpetrate a Sybil attack (see Section 2.2.1.1). Note that this property cannot be ensured with a public/private key pair, since a node is capable to generate an arbitrary amount of self-certificates. In this respect, a CA or login server could ensure the one-to-one correspondence between entities and identities. Proposed

distributed solutions make identities costly to produce, mitigating then whitewashing or generating multiple identifiers. However, distributed approaches do not completely eliminate this threat.

The latest property we mention is related with user's privacy. Anonymity in peer-to-peer trust/reputation systems is not ensured if the identifier and the network address of a participant are publicly related. To hide the relationship between a node's system identity and its network address there exist a plenty of approaches, such as Onion Routing [46].

A detailed discussion about identity in peer-to-peer systems is provided in Section 2.3. Anyway, assuming sufficiently strong identities, participants within a system armed with a trust/reputation scheme can gather information from others in order to consistently evaluate their behavior. The approach employed to gather such information is normally called the *source of information* of a trust/reputation system.

Source of Information. Normally, *quantity* and *quality* of trust information are diametrically opposed aspects. In other words, when a system distributes trust values among participants, as the more information is gathered less credible is each piece of information. The main reason behind this fact is that malicious participants may introduce false feedback in the system. For this reason, there are mainly three approaches we can find in the literature to tackle with this issue: First, *Global Reputation systems* are intended to merge the opinions of every user about each other. Second, in *Transitive Reputation systems* a node only considers opinions of its near-by neighbors or acquaintances, to aggregate the opinion of others in a controlled manner. Finally, *Local Trust systems* only process first-hand information, namely personal experience, to evaluate how others behave. Classifying trust/reputation systems depending on their source of information is treated in more detail in the next section.

However, collecting information about other's behavior is worthless without translating that information into a quantitative metric. This process is intended to rate neighbors depending on past transactions (scoring) in order to condition server selection future transactions (ranking).

Scoring and Ranking. Creating an appropriate method to calculate a node's behavior is a complex task. There are several design decisions involved in this component of a trust/reputation system and, some of them, are highly coupled with the nature of the system to be protected. Thus, some simple questions in this regard might be: *What*

we should take into account to calculate trust values about individuals? Are old transactions as important as recent ones? How we could correctly express the notion of trust?

An important aspect is selecting which kind of transactions to adopt as input of a trust algorithm. In an ideal case, *successful* and *failed* transactions should be taken into account for evaluating a participant [42]. However, this fact is not always achievable. For instance, when a node is off-line, it could falsely appear to be dishonest if there is no other mechanism to know the origin of such failure. Normally, detecting the bad behavior of a participant —if possible— gives much more information that tracking the amount of successful transactions exchanged with it.

Clearly, the behavior of an individual can be unstable along the time. Furthermore, if good behavior yields to be probably selected as server of other participants, an attacker could correctly behave for a period of time and then start to defect —i.e. *traitor*. In this regard, there exist the possibility of giving more importance to recent transactions than to old ones [47]. By doing this, the trust calculation rapidly reflects any sign of recent misbehavior in the trust value of a certain participant, even though it has been transacting properly in the past.

The trust which a node A deposits in another node B needs to be expressed in some form. This value can be expressed in a binary form (trusted/untrusted), in an integer scale (e.g. $[1-10]$), or using floating point values (e.g. $[0,1]$). As other aforementioned features of the calculation, this is up to the designer. However, in open architectures such as Peer-to-peer, systems, a binary representation seems not to be suitable, since all participants are untrusted entities.

Finally, any trust/reputation system should react depending on the behavior of other participants. The way how a system reacts to the behavior of others is known as the system's *response*.

Response. Gathering past transactions, sharing reputations values and using them to compute opinions about participants is needed to take a further action. However, *which kind of response could be guided by trust information?*

Actually, we mainly distinguish three possible responses. First, a node could simply decide the highest trusted neighbor to transact with, that is, *neighbor selection*. Thus, behavioral information gathered by a node is only used to the benefit of that node. Concretely, to maximize the probabilities of successfully transact. Second, this trust information can be used to promote good behavior among users. In other words, the system can build *incentives* to encourage cooperation. Incentives can mitigate malicious behavior by rewarding cooperation with an improved service (e.g. bandwidth,

storage capacity, quality), or even money. Finally, other systems propose a diametrically opposite solution: *punishments*. Thus, in case a node is identified as misbehaving or selfish, it is punished with lower quality services or even disconnected from the system.

Categorizing Trust and Reputation Systems

One of the main points of the security technique presented in this thesis is the approach designed to gather behavioral information about other participants. Concretely, the trust system incorporated in our technique takes into account only first-hand observations, that is, routing operations performed in each node, to reinforce routing in a local fashion. However, to better understand this design decision, we illustrate a categorization of trust/reputation systems based on the source of information:

Global Reputation Systems. Considering a global reputation system, the reputation score of a single user is built as a result of gathering the other's opinion about that user. This means that in a system of n participants, the reputation value of node n_1 can be built by, at most, the $n-1$ remaining users. It is not hard to see that, in a distributed infrastructure, sharing in a secure way such amount of information might be excessive due to the associated network and computational overhead. However, in other settings this reputation scheme has been successfully applied. As a clarifying example, here we briefly depict the Ebay's [48] reputation mechanism.

Ebay is an on-line auction service. Due to the monetary nature of transactions among users, Housers and Wooders [49] designed a robust reputation mechanism for Ebay.

In Ebay, users leave ratings about other users after the completion each transaction. A transaction may be selling or buying and article, for instance. The algorithm intended to calculate the reputation of a certain user expects as input integer values, where 1 corresponds to a successful transaction, a value of 0 means a neutral transaction and, finally, -1 represents a negative rating.

A problem arises when malicious users try to spread false negative opinions about other participants —i.e. *slandering*. To tackle with this contingency, Ebay lets users to defend against negative ratings by leaving comments about ratings. Therefore, other participants can evaluate if a certain rating about a user is fair or not. Aiming to precisely evaluate the quality of a transaction, there are other criteria which can be rated, such as delivery time or article correspondence to the advertisement.

In general terms, the reputation of a user is the difference of positive and negative ratings. Such calculation is public and visible by the rest of users and it is used as a confidence metric (*incentives*).

Transitive Reputation Systems. A Transitive Reputation system is based on aggregating trust information exchanged with close-by nodes to a user's personal experience. Generally, this approach reduces the time needed to correctly identify the behavior of a certain participant. However, it is also vulnerable to false rumors spread by malicious participants. Additionally, this approach incurs additional network and computational overhead. A really popular transitive reputation is EigenTrust [50], devised by Kamvar et. al. in 2005.

The EigenTrust algorithm was designed to deal with inauthentic content in peer-to-peer file-sharing systems. This algorithm generates reputation values by merging a node's direct observations of its neighbors with the opinions its neighbors about each other. Direct observations are provided manually by the user, after checking if downloaded content from a peer is authentic or not. A user ranks a file in a binary fashion, that is, $0 \rightarrow$ inauthentic and $1 \rightarrow$ authentic.

Opinions of nodes are automatically exchanged among themselves and is what gives to the system the reputation transitivity. Broadly speaking, the reputation metric is formulated as follows: first, each node i computes a normalized local trust value c_{ij} for node j based on its direct observations. Then, node i calculates the reputation metric for another participant, k, by asking its neighbors, j, for their opinions of identity k. These third party opinions are weighted depending on the i's trust of j: $t_{ik} = \sum_j c_{ij}c_{jk}$. By accepting the opinions of neighbors of neighbors, the system will continue broaden its view of trust across the system.

Local Trust Systems. A node armed with a Local Trust system is intended to gather and store behavioral information of its neighbors, using such information in future operations. This approach does not need to exchange any reputation information with other close-by nodes avoiding network overhead, algorithmic complexity and false feedback coming from malicious nodes. However, the time taken by a node to obtain a reliable trust value of a certain neighbor is slower considering only its own transactions than receiving additional feedback from others. To better understand this approach, we shortly describe a popular incentives mechanism for content distribution, called Scrivener [51].

The main objective of Scrivener is to promote cooperation among users in peer-to-peer file sharing applications, assuming that by nature they are selfish. Nodes gather behavioral information about neighbors, but this information is not shared at all. On the contrary, this information concerns only to a node pair maintaining a sharing relationship in the overlay.

The main concept behind Scrivener is clear: each node maintains a persistent history of transactions with its neighbors. For each neighbor, a node keeps a (1) credit/debit account and its (2) trust value or confidence metric. Credits reflect resource provisioning of a neighbor to the user whereas debit means the resource consumption from a user of that neighbor. The trust value reflects how often a request is successfully fulfilled by a given node. Employing this mechanism, nodes are encouraged to cooperate in order to receive resources when needed.

Scrivener also provides a small credit to newcomers, in order to allow them joining the system. However, Scrivener effectively implements mechanisms to avoid selfish nodes —i.e. free-riders— abusing this initial credit by frequently changing their identity. Finally, Scrivener proposes a mechanism to fairly exchange resources with non-neighbors, called transitive trade.

Once reviewed the necessary background involved in this thesis, we overview the literature on routing security in DHTs and emphasize the key differences with our work.

2.5 Related Work

In the previous sections, we introduced the main motivation of our research: the existing *security vulnerabilities* in the DHT lookup service. As a popular security countermeasure, we reviewed the concept of routing redundancy as well as some routing redundancy techniques. However, we mentioned that redundancy itself offers a poor cost/effectiveness trade-off: the larger the network is, the more parallel paths are needed to meet a certain routing reliability threshold. In this line, we argued that taking into account the quality of forwarders in terms of routing reliability could be an attractive approach to reinforce routing without introducing additional costs. To this end, we should undertake two different tasks which have been discussed in this chapter: (1) Persistently identifying nodes within a system. (2) Evaluating the routing reliability of nodes resorting to trust/reputation mechanisms. Therefore, selecting in each node the most reliable neighbors, the whole network will presumably become more resilient against malicious participants.

This section is intended to survey existing routing security techniques and discuss the major differences between these techniques and our approach intended to reinforce routing.

For secure forwarding in DHTs, there are few works in the literature that present, from the viewpoint of robustness, redundancy as just an alternative with its own advantages and shortcomings; rather, recent results have focused only on its benefits, paying no attention to the role that the quality of the intermediate routers play on delivery paths. For example, [13] proposed first to route normally, and then perform a failure test to decide whether or not routing had gone wrong. If the test failed, routing was retried but this time with a secure routing protocol. A similar idea was adopted in Cyclone [30], but guaranteeing d independent paths between every two nodes in the overlay. Harvesf and Blough [28] tried to create d disjoint paths by equi-spacing 2^{d-1} replicas on the Chord ring. Querying the 2^{d-1} replicas in parallel, they were able to increase routing robustness. Finally, Hildrum and Kubiatowicz [31] demonstrated that appointing d forwarders for each lookup step in an iterative fashion is a more effective approach to provide routing redundancy. However, the main problem of using pure routing redundancy is the poor asymptotic guarantees it can provide on the success rate without overloading any node (as the value of d increases).

The following works that attempt to improve the quality of delivery paths:

In [12], the authors overview the potential benefits of reputation on routing, hypothesizing an abstract reputation system with both false positives and false negatives. Although the results are promising, this work does not include any algorithm to sustain their arguments.

In [52], the authors propose SPROUT, a DHT routing protocol that increases the odds of message delivery by using social links. SPROUT defines a reputation model based upon social distance to avoid routing messages through dishonest peers. However, it relies on social links that may not be always available.

Another related algorithm is the Feedback Forward Protocol (FFP) [53]. In FFP, each peer captures evidence to predict the routing behavior of its neighbors. Based on this evidence, a node can find out whether a given neighbor did a good routing job, and select it as a next-hop neighbor if it generally forwards messages. The problem with this protocol is that malicious peers can spread false feedback, a vulnerability that can be exploited to carry out denial of service attacks against particular peers.

Trust and reputation mechanisms have been effectively introduced in several distributed systems, specially in P2P settings [47], [54], [40], [50]. Additionally, major advances in this regard are already exceptionally surveyed [44], [42], [43]. In general,

the purpose of trust and reputation mechanisms in P2P settings is to enforce cooperation and fairness in applications such as file-sharing and content distribution [55], [51]. However, this thesis differs from previous works concerning trust/reputation systems in various aspects.

First, the introduction of trusted relationships to reinforce DHT routing is a recent research topic and remains considerably unexplored. Second, the majority of trust/reputation systems designed for P2P applications require the exchange and dissemination of trust values among participants. Clearly, this fact increases the available trust information of each node and accelerates the creation of trusted relationships when transactions are limited. The trouble is that this process of disseminating trust information within the network introduces additional threats and shortcomings (e.g. network overhead, false rumors). However, in routing settings, each node can benefit from a potentially large number of transactions. In other words, unlike transactions in other applications (e.g. file-sharing), routing transactions are numerous and become an abundant source of trust information. We believe that this source of information is enough to create robust trust relationships while avoiding the exchange of trust values among nodes.

The closest related work we are aware of is our own research presented in [41]. In [41] authors propose, as a case of study, a reputation mechanism which considers only a node's routing operations to select future forwarders. Although this work is in line with the present thesis, there exist several and important differences between both. First, this protocol does not maintain a persistent database of historical transactions to make routing decisions; past transactions with a certain neighbor are kept until it becomes off-line or changes its identity. However, exhaustive research has been focused on providing mechanisms to ensure the persistence of identities in DHTs [44], [36], [56]. Therefore, we can assume the persistence of identities and thus exploit the benefits of keeping a local history of neighbors, specially in terms of trust convergence. Second, they do not consider the advantages of iterative routing to this kind of reputation mechanism. Finally, authors in [41] did not propose any architecture to develop this technique in a real environment.

Compared with prior work, our technique is driven towards improving the quality of routing paths by using only historical-direct feedback, that is, a node's past transactions with other participants. Consequently, our technique avoids the introduction of additional network overhead due to the exchange of trust values as well as additional routing paths. Besides, taking into account only a node's local transactions, our approach is immune to false rumors spread by malicious participants.

3

Sophia: Local Trust for Securing Routing in DHTs

3.1 Introduction

Distributed Hash Tables (DHTs) are becoming a key building block for many distributed applications due to their ability to scale in fast-growing environments with massive workloads. Apart from the clear impact of DHTs in Peer-to-Peer computing [57], [58], other paradigms such as Grid and Cloud Computing have adopted DHTs to achieve specific targets. In the case of Grid, a plethora of research projects have benefited from DHTs to decentralize services such as Resource Discovery [59] or Data Management [60], to name a few. In the realm of Cloud Computing, DHTs have not only been employed in trusted environments [7], but also in recently proposed open-cloud architectures such as Cloud Federations [61], [62].

However, security has become a primary concern to protect DHTs from malicious attackers, which can join the network and disrupt all operations by intentionally disobeying protocol specifications [11]. One critical aspect is message forwarding [12]. For example, a malicious peer may simply drop all messages passing through it, even though these messages are cryptographically secure.

In this sense, we discern two main approaches to overcome this issue [12]: (1) We can secure message forwarding by introducing redundancy. Thus, redundant paths and replicated data alleviate the impact of misbehaving nodes. (2) We can improve the quality of selected paths in the forwarding process.

Regarding redundancy, there exist well-studied techniques [11] that strengthen DHTs against routing attacks. The trouble is that the introduction of redundancy increases communication costs and might significantly reduce scalability. Therefore, it seems reasonable to focus on improving the *quality of routing paths*. To achieve this we do two things. First, we identify the honest forwarders within the network. For this purpose, we resort to standard techniques in the reputation and trust literature that aim to distinguish cooperating nodes from misbehaving ones. Second, we implement a neighbor selection policy to choose the routing entries according to the trust scores.

One problem with existing reputation systems is that, in general, they need to disseminate trust values among participants [43], [42]. In the case of improving routing, this dissemination process becomes a hard problem: Firstly, communication costs due to the sharing of reputation scores could be prohibitive compared with the routing cost itself. Secondly, we can find in the literature a myriad of specific attacks related with the dissemination of reputation values such as slandering, self-promoting, etc. [43], which can even make routing more vulnerable than before.

For these reasons, we rather propose to fortify routing in DHTs using only *first-hand observations*, i.e., direct observations about the success or failure of a node's own lookups. In this sense, *iterative routing* fits perfectly in this scheme since a node does not rely in third party message delivery; the requester personally checks every node along a lookup path.

In this thesis, we present *Sophia*: Securing Overlays with a Personal HIstory Approach. (1) *Sophia* is a novel and generic security technique which combines iterative routing with local trust to fortify routing in DHTs. *Sophia* strictly benefits from first-hand observations (e.g. lookup success/failure) to qualitatively improve forwarding paths. (2) Moreover, unlike redundant routing, *Sophia* dynamically protects routing without introducing additional network overhead. To the best of our knowledge, this is the first work which exploits a local trust system to reinforce routing in DHTs.

Finally, we use Kademlia DHT to validate our approach because it is largely the most deployed DHT nowadays. We compare the performance of *Sophia* with Kademlia redundant routing in various hostile scenarios. Compared with Kademlia redundant routing, *Sophia* is able to increase successful routing rate up to 35% for the same degree of redundancy. This means that *Sophia* can tolerate the same fraction of malicious nodes than Kademlia, with less parallel paths, yielding important traffic savings. Moreover, *Sophia* evolves towards better quality paths as it receives more feedback from transactions. This fact allows *Sophia* to dynamically recover from unexpected attacks.

3.2 Threat Model and Assumptions

3.2.1 Threat Model and Assumptions

In this chapter, we assume the following threat model. A Kademlia overlay is infiltrated over a certain period of time by malicious users joining the overlay. After some time, the network has $N(1-f)$ honest nodes and Nf malicious nodes, where f represents the fraction of attackers. Further, we assume that the attackers are uniformly

distributed along the m-bit identifier space. That is, we adopt the random fault model, where each peer is malicious with some probability f, irrespective of the other nodes. This means that the attackers cannot assign their identifiers, safeguarding the overlay against *Sybil Attacks* [24].

A wide spectrum of attacks preventing honest nodes from communicating together has been identified in the literature. In addition to Sybil attacks, the attackers can conduct *Eclipse attacks* [63] to corrupt the routing tables of honest peers, and *forwarding attacks* to block the resolution of keys. Although Eclipse attacks are not negligible, dealing with such attacks is out of the scope of this thesis.

We focus in this thesis on forwarding attacks, which are not less disruptive than Eclipse attacks, as their objective is to block the resolution of keys. Suppose an honest user u wants to find a key K. In a standard DHT, u asks another node u knows is closer to K, which in turn forwards to another even-closer node, and so on until key K is found. Attackers can disrupt this process by spreading false information about their closeness to a particular key, by intercepting routing queries or responding with "no such key" messages.

For this chapter, we consider two types of forwarding attacks:

Uncoordinated Attacks. Uncoordinated attacks are launched by the individual attackers independently. In this attack, an attacker drops each incoming request in an attempt to censor the access to as many key-value pairs as possible. For the other aspects the attacker acts as a regular Kademlia client. This means that attackers answer to `PING` RPC messages in order to be part of the network.

Collusion Attacks. In this attack, we assume that there is an adversary who coordinates all the malicious clients to punish honest users. In this case, the attackers create a fictitious route for each query that terminates at the closest attacker to the destination key in an attempt to deceive the source into thinking that the key-value pair does not exist. To generate the artificial routes, the adversary bootstraps all the malicious instances into a parallel Kademlia overlay network. In this way, every time a request touches a malicious client it gets trapped into the overlay of malicious instances. Note that when a Kademlia node receives any message, it inserts (or updates) contacted nodes into the appropriate routing table bucket. Exploiting this fact, each time a regular node contacts an attacker it introduces more adversaries into its own routing table. Finally, if the query is from a malicious submitter, the regular routing table is used in order to minimize the routing effort of the malicious coalition.

We conclude this part with the final assumption that users have non-spoofable semi-permanent identifiers to ensure the integrity of the transactions. This is why we

use the hash over a public key to generate identifiers as in S/Kademlia [32]. To prevent users from spoofing the identity of another client, honest users use a challenge-response protocol for validating identifiers similar to that in [64].

3.2.2 Existing Countermeasures: The Sibling Zone

Ensuring that data objects in the DHT have high availability levels when the nodes that are storing them are not themselves 100% available requires some form of data redundancy. Normally, DHTs resort to object replication to provide an acceptable degree of data availability.

Specifically, Kademlia replicates each key-value pair over the k closest nodes. For small values of k, this number of replicas might be too small to guarantee that a key-value pair can be found with high probability (w.h.p.) after adversarial deletions. To address this problem, the authors of S/Kademlia [32] added a list of sibling neighbors of length $\eta \cdot s$ per node to ensure that Kademlia routing protocol reaches at least s siblings of the destination key w.h.p. In Kademlia jargon, the siblings of a key are the nodes whose XOR distance to the key is minimal. As shown in [32], a value of $\eta \geq 5$ is enough to make sure that routing operations converge to at least s siblings w.h.p. and hence, allow Kademlia to store data in a safe way.

In our view, it is interesting to analyze in our simulations how Kademlia and *Sophia* perform when they are armed with this countermeasure. Hence, we will make use of a sibling list of size 2^{S_z}, where $S_z = \log_2 \left\lceil \frac{N}{\eta \cdot s} \right\rceil$ denotes the prefix length such that $\frac{1}{2}\frac{N}{\eta \cdot s} < S_z \leq \frac{N}{\eta \cdot s}$. This way, we ensure that there exists at least s nodes whose identifier shares the first $m - S_z$ bits of any given key.

3.3 Sophia: System Design

As argued in Section 3.1, we propose to fortify routing using only first-hand information about the success or failure of a routing operation. This is the main reason why iterative routing fits neatly into our solution. With iterative routing, the requester asks each intermediate node along the routing path about the next hop. Hence, the requester can accrue personal experience about the routing behavior of intermediate forwarders and rate them according to the result of the routing operation.

To mitigate the effect of bogus results in the forwarding process, *Sophia* uses redundancy without merging forwarding candidates: each parallel path is an isolated lookup to the same key. This design decision makes *Sophia* more resilient to spurious forwarders retrieved by malicious nodes. Additionally, keeping paths isolated from

each other is a clearer way of gathering historical information: the success/failure of a certain path affects only actual forwarders of that path (see Section 3.3.3).

In *Sophia*, each node synthesizes direct observations on the routing behavior of its neighbors in form of a trust ratings as a result of its routing operations. These ratings are stored and updated in a data structure called *Personal History* (PH). As more observations are available, the ratings about neighbors become progressively more reliable and eventually converge, enabling nodes to identify which forwarders are more likely to route successfully towards the destination.

3.3.1 Architecture

Sophia is a generic security technique and it is, therefore, applicable to any DHT. Since the selection of the next hop is individually done by each host, nothing prevents a regular participant from communicating with any computer armed with *Sophia* and *vice versa*. Also, the use of iterative routing gives *Sophia* the skill to improve the quality of routing paths with no extra overhead. It is important to note that iterative routing provides more feedback than recursive routing does; for each lookup, the requester inspects all the intermediate hops along the routing path. Consequently, the time necessary to identify reliable nodes is smaller with iterative routing than with recursive routing.

Fig. 3.1 presents *Sophia*'s architecture, which will be detailed in the following subsections. Moreover, that picture shows how a *Sophia* node performs a lookup. In the forwarding process, a malicious node drops the lookup, making the routing operation fail. Consequently, the requester applies a *trust policy* (Section 3.3.3) —e.g. punishing all intermediate nodes or only the last hop— to evaluate the routing behavior of intermediate hops. After rating each hop, their corresponding trust values are updated into the requester's *Personal History* (Section 3.3.2) using a *trust calculation algorithm* (Section 3.3.4). As a result, the requester accumulates more knowledge about the behavior of its neighbors and will presumably perform a better *neighbor selection* (Section 3.3.5) for the future lookups.

3.3.2 Personal History

The Personal History (PH) is the main data structure of our architecture. In it, a node stores the trust values for their neighbors. The PH presents the same structure of a standard routing table. For each entry of the PH, we have a list of candidate forwarders of size h. The value of h must guarantee, with high probability, that there

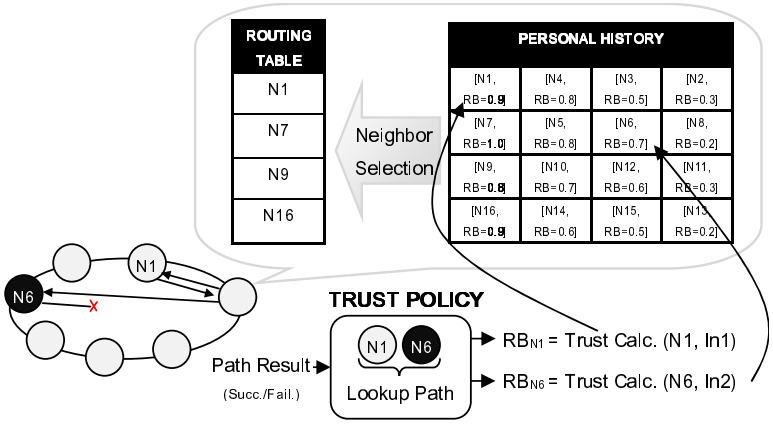


Figure 3.1: Architecture of *Sophia*.

exists at least one honest candidate for each entry. By a reasoning similar to that developed for the sibling zone (see Section 3.2.2), it can be shown that $h = \Omega(logN)$ is sufficient to guarantee this property. From now on, we will refer to each entry of the PH by the term h-bucket.

3.3.3 Local Trust Policies

The provisioning of a criterion about how to evaluate nodes involved in a transaction is given by the *trust policy*. In our system, a transaction corresponds to the process of constructing a routing path. Therefore, the result of a transaction can be formalized well using a binary value, a value of 1 for a successful transaction and 0 for a failure.

Basically, in *Sophia* a trust policy is an algorithm which decides the input value given to the trust calculation (see Section 3.3.4). Such decision is based on the result of a transaction and it is independently applied to all forwarder nodes responsible for that transaction.

In case of sending a lookup using α parallel paths, we consider α isolated transactions —as mentioned in Section 3.3. Consequently, the trust policy is applied in each of these α paths. Thus, irrespective of whether a lookup succeeds or not, the trust policy evaluates forwarder nodes depending upon whether their respective paths were successful or not.

For evaluating our system we developed two policies:

Pessimistic Policy. Pessimistic Policy gives a positive trust score of 1 to all nodes along the path if a transaction has been successful. On the contrary, Pessimistic Policy punishes all nodes along the path giving a trust score of 0 when a transaction fails. Note that this policy could erroneously punish cooperating nodes (false positives).

However, since a node evaluates its neighbors only with its own transactions, without any other external information, this policy is more conservative in misclassifying malicious nodes as good ones.

Oracle Trust. To compare the performance of local trust policies we implemented an Oracle. The Oracle is, theoretically, the best trust policy possible since it evaluates nodes depending on whether they are malicious or not (independently of the transaction result). Clearly, this kind of policy cannot be materialized in the real world. Nonetheless, from our point of view, it is an accurate way to compare any local trust policy with the theoretically best one.

3.3.4 Trust Calculation

A trust calculation algorithm computes trust values which represent the routing behavior (RB) of a node's neighborhood. In this section, we adopted some approaches from the literature to design a trust algorithm that properly deals with routing particularities.

For the sake of clarity, we will describe our algorithm following the decomposition and analysis framework proposed in [43]:

Source of Information. We defined as input for the trust system first-hand observations on the success or failure of lookups, a vision aligned with representative trust systems such as [51]. Based on this input, the information that the calculation algorithm finally receives are the binary values given by the trust policy after processing a transaction.

Information Type. We decided to take into account a node's good and bad behavior. This twofold consideration of a node's activity will provide a broader notion of its behavior and will be useful in order to response to network or behavioral changes (e.g. *traitors*) [43].

Temporal Aspects. It is important to weight the importance of recent and historical transactions [43]. In this respect, we propose to compute the *routing behavior* value in an aggressive short-term history fashion, in order to rapidly tackle with unexpected defectors [43], [47].

Trust Metric. To represent the degree of trust a node deposits in a neighbor, we use a real value $[0, 1]$ similar to other reputation systems such as [55].

Calculation. Trust calculation is performed locally in every node in the network to obtain its own *view of trust*. Finally, the routing behavior value (RB) of a node n_i is calculated as follows:

$$RB\,(n_i) = \lambda \cdot \frac{\sum_{j=t-d}^{t} RB(n_i,j)}{d} + (1-\lambda) \cdot \frac{\sum_{j=0}^{t-d-1} RB(n_i,j)}{t-d},$$

where $\lambda \in [0, 1]$ determines the weights given to the most recent d transactions from the total t.

3.3.5 Select-Best as a Neighbor Selection Policy

At this point, we have developed means to calculate and store the trust values of neighbors. However, we also need a criteria to update a node's routing table depending on its neighbors' trust scores, a role adopted by the neighbor selection policy. To measure the potential of our solution, we implemented the policy that maximizes the routing improvement. This policy updates a bucket b_i with the k highest trust-valued neighbors in the ith h-bucket of PH. From now on, we will refer to this policy as Select-Best.

3.4 Validation

3.4.1 System Model

Our simulation scenario consists of a fully populated Kademlia tree (i.e. 2^m nodes). Such scenario forces lookups to perform the maximum number of possible hops since every requested key has an owner. Additionally, a lookup will successfully terminate if any lookup path reaches the requested key *sibling zone* (S_z) (see Section 3.2.2).

In order to test routing, every node in the network injects periodically (each δ seconds) lookups to uniformly distributed random keys. Moreover, the amount of lookups sent by a node is divided in *transitory lookups* (T_l) and *stationary lookups* (S_l). We will store statistical data only for stationary lookups S_l; transitory lookups are intended to bootstrap the network and they are not considered in the evaluation at all.

Table 3.1: General simulation settings.

Parameter	Value
Number of Nodes (N)	8,192 nodes
ID Length (m)	13
Redundant Paths (α)	$[1 \rightarrow 3]$
k-bucket size (k)	3
Sibling Zone (S_z)	0 bits, 4 bits
Transitory Lookups (T_l)	50
Stationary Lookups (S_l)	$[100 \rightarrow 1000]$
Lookup Period (δ)	36 sec.
Fraction of Malicious Nodes (f)	$[0.0 \rightarrow 0.6]$
Attack Type	Individual, Collusion
History Bucket Size (h)	10
Short Term History Transactions (d)	3
Short Term History Weight (λ)	0.7

3.4.2 Experimental Results

In this section, we address the results obtained after implementing and evaluating our prototype. Concretely, we implemented *Sophia*, over the Kademlia protocol [14], in Erlang[1]. In this sense, Erlang provides a high degree of scalability and concurrency for developing large-scale distributed systems. Additionally, our simulator has been carefully designed and developed to ensure the correctness of the experimentation. Furthermore, the simulations shown in this article are the result of several months of work and exhaustive testing; this fact make us feel confident about the validity of our results and final conclusions. For each configuration (Table 4.1), each simulation was run 20 times and the results were averaged to obtain the plotted values.

We are mainly interested in assessing 4 aspects of our system, and comparing them with redundant routing in Kademlia:

- We evaluate how many lookups *Sophia* needs to provide a greater routing robustness compared with Kademlia redundant routing, that is, convergence.
- We analyze how the lookup path management of *Sophia* affects the average path length.
- We assess the self-adjustment of both approaches when a fraction of nodes unexpectedly start to misbehave.
- We detail how transactions are distributed between cooperating and adversarial nodes (load balancing).

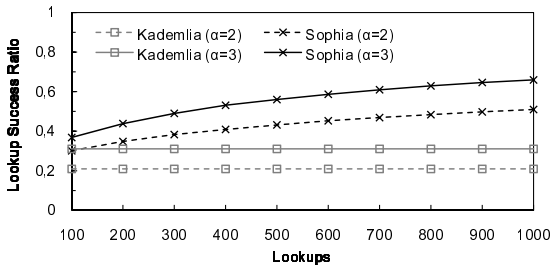
[1] http://www.erlang.org/

The main metric used in the evaluation is the Lookup Success Ratio (LSR). We define this metric as follows:

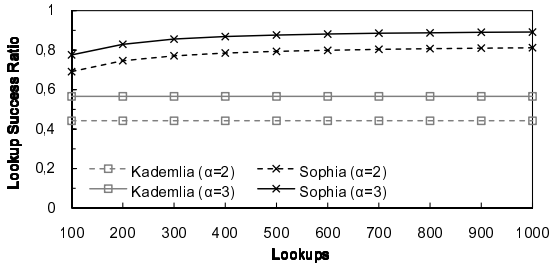
$$LSR = \frac{S_{lookups}}{T_{lookups}}, \tag{3.1}$$

where $S_{lookups}$ is the amount of successful lookups achieved by nodes from the total number of lookups sent $T_{lookups}$.

Trust Convergence. The performance of *Sophia* is related with the amount of transactions exchanged with neighbors. Figures 3.2 and 3.4 show the degree of routing robustness under attack (measured as LSR) provided by *Sophia* and Kademlia and how it evolves along the time. LSR is measured by intervals of 100 stationary lookups. The measurement starts from scratch in each interval. During the transitory period of a simulation, each node injects 50 queries (T_l) into the network. As can be observed in graphics 3.2 and 3.4, this reduced number of transitory lookups is enough for *Sophia* to obtain a greater LSR in the first interval of measurement compared with Kademlia, for the same degree of redundancy (α).



(a) $S_z = 0$, 20% Collusion Attack



(b) $S_z = 4$, 20% Collusion Attack

Figure 3.2: Lookup Success Ratio evolution of Kademlia and *Sophia* when 20% of nodes perform a Collusion Attack for different values of S_z and α.

Results in Fig. 3.2 belong to a hostile scenario where 20% of nodes collude to harm cooperating nodes. The difference between both graphics lies in the Sibling Zone (S_z) configuration; in the first graphic $S_z = 0$ whereas in the second one $S_z = 4$. The evolution of *Sophia* is significant in both cases; in the first interval of measurement, *Sophia* for $\alpha = 2$ provides the same or greater routing robustness than Kademlia for

$\alpha = 3$. The main cause is that transitory lookups are enough for *Sophia* to identify and evict a fraction of adversarial nodes from routing tables. Non-surprisingly, as more observations are available, the ratings about neighbors become progressively more reliable. This fact is clear in Fig. 3.2; for instance, when $S_z = 0$, *Sophia* outperforms Kademlia in approximatelly 30% for the same α.

Despite Kademlia slightly enhances the LSR when increasing α, it does not learn from neighbors, so the LSR remains lower than *Sophia*. Note that, to provide a certain threshold of routing robustness, *Sophia* needs less parallel paths than Kademlia. Thus, reducing α from 3 to 2 represents a network traffic saving of 33%.

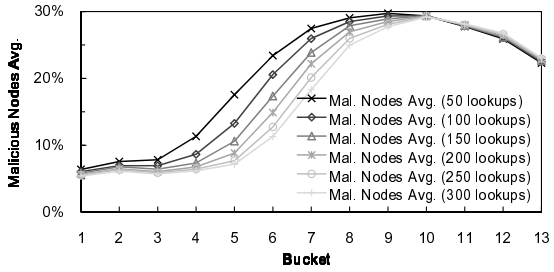
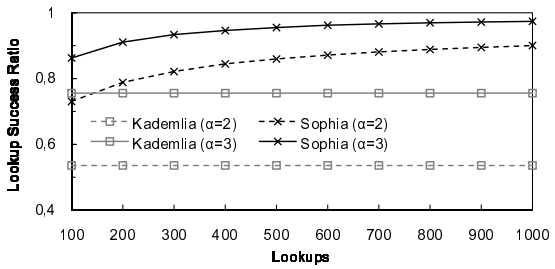


Figure 3.3: *Sophia*'s malicious nodes avg. in routing table buckets, depending on the amount of lookups sent per node (30% Individual Attackers, $\alpha = 2$).

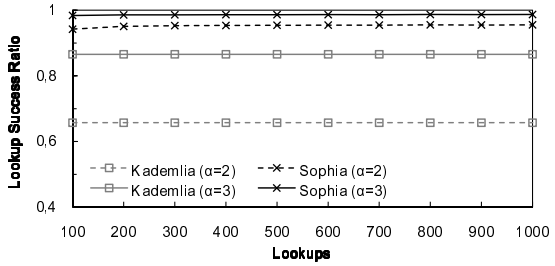
It is worth mentioning existing differences in LSR depending on S_z configurations (Fig. 3.2). Kademlia improves noticeably when introducing a larger S_z. The reason is that a larger S_z implies a shorter Average Path Length, and this fact increases lookup success probability. However, *Sophia* takes a greater advantage than Kademlia of a larger S_z. The main reason is that, in the forwarding process, nodes avoid using the lowest k-buckets of the closest nodes to the requested key. Since a node rarely uses its own lowest k-buckets to perform lookups, it cannot discern between cooperating and adversarial nodes in these k-buckets. Consequently, if an incoming lookup request asks for nodes placed in these lowest k-buckets (especially when $S_z = 0$), these nodes will be retrieved without being properly identified as good/bad forwarders. This thesis is corroborated by Fig. 3.3, where we can observe the evolution of a *Sophia* node routing table, measured as the average fraction of malicious nodes per k-bucket.

Fig. 3.4 corresponds to a hostile scenario where 30% of nodes perform an Individual Attack.

Results depicted in Fig. 3.4 follow a similar fashion than the previous experiment. However, results for this threat model are better for both systems since an Individual Attack is less aggressive than a collusion. Again, Kademlia obtains a constant and lower LSR along the experiment, compared with *Sophia*.



(a) $S_z = 0$, 30% Individual Attackers



(b) $S_z = 4$, 30% Individual Attackers

Figure 3.4: Lookup Success Ratio of Kademlia and *Sophia* when 30% of nodes perform an Individual Attack for different values of S_z and α.

Therefore, *Sophia* is able to significantly improve routing resilience under moderate traffic conditions.

Lastly, we want to compare the convergence speed and routing resilience of *Sophia* using Pessimistic Trust policy versus Oracle Trust (Fig. 3.5). As expected, the Oracle discerns better and faster between cooperating and adversarial nodes. The main difference between both policies resides in the number of lookups needed to converge: in the first interval (after T_l and the first 100 stationary lookups) the Oracle outperforms Pessimistic Policy in 13% for $\alpha = 2$ and 12% for $\alpha = 3$. The reason is that the Oracle incurs no false positives, being faster to identify misbehaving nodes. However, Pessimistic Policy evolves reducing the LSR disadvantage compared with Oracle considerably at the end of the experiment. As a conclusion, although the convergence of the Oracle is clearly faster, the performance of both policies is limited by the amount of lookups performed, especially to identify nodes placed in the lowest *k*-buckets of the routing table.

Average Path Length (APL). Table 3.2 compares APL obtained by *Sophia* and Kademlia with different values of S_z and redundancy configurations in the absence of adversarial nodes. When $S_z = 0$, lookups must strictly find the key owner. This fact increases APL in ≈ 1.2 hops compared with $S_z = 4$ and the same α. Such reduction of the APL, due to a larger S_z, improves significantly routing resilience in both systems as depicted in figures 3.4 and 3.2.

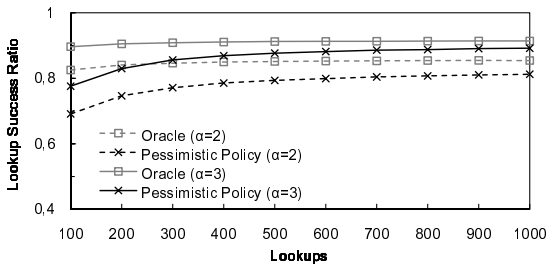


Figure 3.5: Lookup Success Ratio comparison of *Sophia* using Pessimistic Trust policy versus Oracle Trust, when 20% of nodes perform a Collusion Attack and $S_z = 4$.

Table 3.2: APL in absence of adversarial nodes.

		Sophia	Kademlia
$S_z = 0$	$\alpha = 1$	$\approx 4,4$	$\approx 4,3$
	$\alpha = 2$	$\approx 4,4$	$\approx 3,8$
	$\alpha = 3$	$\approx 4,4$	$\approx 3,5$
$S_z = 4$	$\alpha = 1$	$\approx 3,1$	$\approx 3,1$
	$\alpha = 2$	$\approx 3,1$	$\approx 2,7$
	$\alpha = 3$	$\approx 3,1$	$\approx 2,5$

As we can observe in Table 3.2, Kademlia reduces its APL depending on α. This fact is caused by the Kademlia lookup algorithm: in Kademlia a lookup is sent in parallel to α nodes. Then, contacted nodes in each path are merged and the α closest nodes to the key are selected to continue forwarding the lookup. In case of reaching the desired sibling zone or detecting that there is not any closer node to the key, all parallel paths simultaneously terminate, decreasing the APL —see Section 2.2.2. However, this approach is significantly vulnerable to bogus nodes retrieved by attackers [32]. On the contrary, *Sophia* uses redundancy without merging forwarding candidates: each parallel path is an isolated lookup to the requested key. This fact makes the APL insensitive to changes in α, because parallel paths terminate independently of each other.

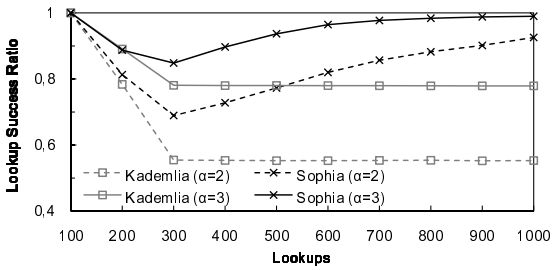
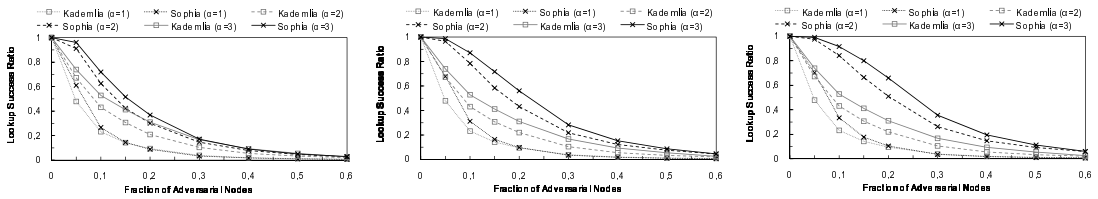


Figure 3.6: *Sophia* and Kademlia self-adjustment comparison activating 30% of individual attackers at lookup 150.

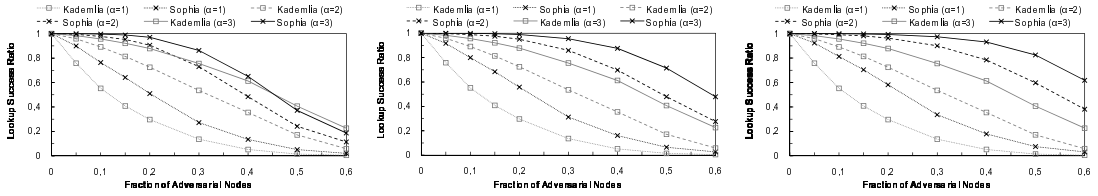
Self-Adjustment. In this part of the experimentation, we address how Kademlia and *Sophia* tackle with unexpected behavioral changes of nodes. Fig. 3.6 corresponds to an individual attack where suddenly (at lookup 150) 30% of nodes start dropping lookup requests. Additionally, both systems are configured with $S_z = 0$ and different degrees of redundancy.

As we can observe in Fig. 3.6, the attack starts at the second interval of measurement. At this point, we can see how both systems present a similar LSR downtrend. Nonetheless, whereas at the third interval Kademlia maintains the same degree of reliability loss, *Sophia* starts to slow down the impact of attackers in the network. From the third interval onwards, Kademlia maintains a static LSR of 55% for $\alpha = 2$ and 78% for $\alpha = 3$ because attackers coexist with cooperating nodes inside routing tables. However, *Sophia* initiates a notably improvement as it identifies malicious nodes, evicting them from routing tables. In conclusion, *Sophia* is capable to dynamically recover from unexpected attacks, restoring the lost routing reliability during the attack.



(a) 100 Lookups **(b)** 500 Lookups **(c)** 1000 Lookups

Figure 3.7: Lookup Success Ratio for different fractions of adversarial nodes performing a Collusion Attack.



(a) 100 Lookups **(b)** 500 Lookups **(c)** 1000 Lookups

Figure 3.8: Lookup Success Ratio for different fractions of adversarial nodes performing an Individual Attack.

Success Rate. In the following subset of experiments, we analyze routing resilience obtained by *Sophia* and Kademlia in terms of LSR for different fractions of malicious nodes. Moreover, we simulated both individual (Fig. 3.8) and collusion (Fig. 4.4) attacks. LSR in both systems is measured for different degrees of redundancy (α) and different amounts of lookups sent per node —concretely 100, 500 and 1000 lookups. Finally, we configured $S_z = 0$ for all simulations in this subset of experiments.

In Fig. 3.7a, we can see how after 100 lookups per node *Sophia* provides a higher but moderated LSR improvement compared with Kademlia. Such improvement is more notorious when the fraction of adversarial nodes (f) is small. For instance, when $f = 5\%$ *Sophia* advantages Kademlia in 14% for $\alpha = 1$ and over 23% when $\alpha = 2, 3$. However, when f exceeds 30%, the performance of both systems decreases dramatically. We should notice that, in line with previous experiments, the performance of Kademlia is static along the time. In contrast, *Sophia* evolves providing a better degree of robustness. This fact is especially clear in graphics 3.7b and 3.7c. We want to highlight results obtained after 1000 lookups sent per node: *Sophia* obtains an average LSR enhancement of approximately 30% for $\alpha = 2$ and 35% for $\alpha = 3$ when $5\% \leq f \leq 20\%$, compared with Kademlia.

In Fig. 3.8 we present results of introducing several fractions of individual attackers. *Sophia* clearly outperforms Kademlia for $\alpha = 1, 2$ in 3.8a —100 lookups sent per node. However, differences for $\alpha = 3$ are less significant, especially when f attains at its highest values. Thus, false positives introduced by *Sophia*'s Pessimistic Trust Policy delay to correctly discern between cooperating and adversarial nodes, when f attains extreme values. Nevertheless, in Figures 3.8b and 3.8c we can see how *Sophia* evolves, increasing significantly LSR.

In conclusion, *Sophia* demonstrates that the utilization of first-hand observations to reinforce routing is beneficial in a wide spectrum of hostile scenarios.

Load Balancing. In the last part of the evaluation, we want to analyze how Kademlia and *Sophia* distribute transactions between malicious and cooperating nodes.

Fig. 3.9 presents the Cumulative Distribution Function (CDF) of the number of received transactions per node after the first 100 stationary lookups (first interval). Moreover, simulation in Fig. 3.9 was executed with 30% of nodes performing an Individual Attack and $\alpha = 2$ for both systems. We can notice that the distribution of received transactions for Kademlia nodes is roughly equal, irrespective of whether nodes are malicious or not. This fact was expected since Kademlia does not evict malicious nodes from routing tables, unlike *Sophia*. In the case of *Sophia*, the fraction of cooperating nodes receives the majority of lookups, providing then a higher degree of routing robustness.

Fig. 3.10 depicts how *Sophia* distribution of received transactions evolves over time. In line with the previous experiment, Fig. 3.10 demonstrates how nodes increasingly avoid using attackers, as more lookups are performed.

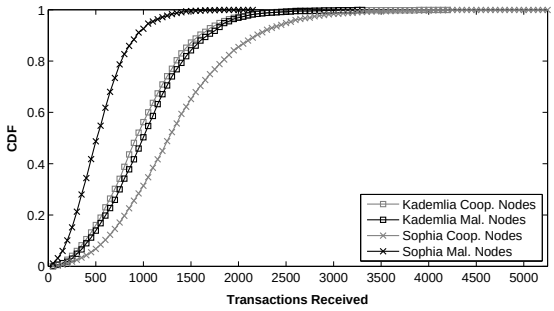


Figure 3.9: Load Balancing comparison between Kademlia and *Sophia* when 30% of nodes perform an Individual Attack and $\alpha = 2$, after 100 lookups sent per node.

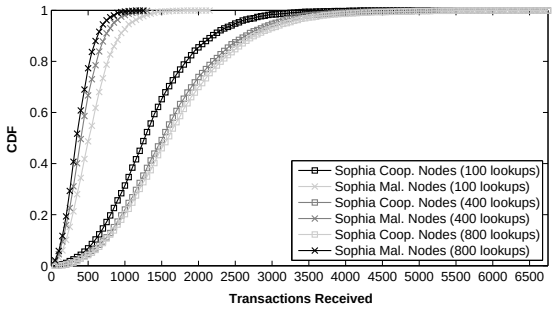


Figure 3.10: Load Balancing evolution of *Sophia* depending on the number of lookups sent per node (30% Individual Attackers, $\alpha = 2$).

3.5 Final Remarks

In this chapter, we described *Sophia*, a novel and generic security technique which combines iterative routing with local trust to fortify routing in DHTs.

We obtained significant improvements comparing the performance of *Sophia* with Kademlia redundant routing in the presence of attackers. Compared with Kademlia redundant routing, *Sophia* is able to increase successful routing rate from 10% to 35% for the same degree of redundancy. This means that *Sophia* can tolerate the same fraction of malicious nodes than Kademlia with less parallel paths, yielding important traffic savings. For instance, *Sophia* with 2 parallel paths provides a greater routing resilience than Kademlia with 3 parallel paths under moderate traffic conditions. This fact implies a network traffic reduction of 33%. Moreover, *Sophia* continuously evolves towards better quality paths as it receives more feedback from transactions. Hence, *Sophia* is capable to dynamically recover from unexpected attacks, restoring the lost routing reliability during the attack.

Therefore, *Sophia* is the first representative example which strictly uses first-hand observations to reinforce routing.

4

Sophia under Dynamic Scenarios

4.1 Introduction

In Chapter 3, we presented *Sophia*: a system intended to reinforce routing using a local trust system over an iterative routing scheme. However, there exist several factors, commonly found in largely deployed DHTs, which may compromise the feasibility of *Sophia* and have not been studied yet:

- *Joint Effects of Dynamic Participation and Misbehavior*: There is no exhaustive study on the performance of *Sophia* under dynamic participation of nodes (i.e. *churn*), an inherent property of large distributed systems. Furthermore, the effects on our security technique when churn and malicious nodes are simultaneously present have not been properly analyzed.

- *Non-Persistent Identities*: The lack of persistence of node identities (i.e. *white-washing*) is an important factor which may hinder the applicability of *Sophia* and it remains unexplored.

By exploring these questions, we assess whether *Sophia* is feasible to be deployed in *the wild* or not. In the affirmative case, largely deployed DHTs will be able to benefit from *Sophia* for providing more reliable lookup services. In our view, this research work represents a step closer to the realization of open and real-world distributed infrastructures requiring the lookup service as a primary element. Examples of these infrastructures include (but are not limited to) distributed wikis [65], distributed resource discovery schemes [59] and data management in grids [60], and even open cloud infrastructures [61], to name a few.

The present chapter covers the following issues: (1) A better understanding of the effects of hostile and dynamic scenarios on a DHT armed with *Sophia*. (2) An exhaustive performance evaluation of our system against routing redundancy techniques. (3) The design of novel policies devised to improve the performance of *Sophia* to reinforce routing under dynamism.

Our simulations show that *Sophia* is feasible under dynamic scenarios after implementing the corresponding adaptation proposed in this chapter. Furthermore, we asses that *Sophia* outperforms the routing reliability provided by pure redundant routing strategies (disjoint paths, wide paths) up to 5 times in hostile and dynamic scenarios. Finally, even when attackers carry out whitewashing, *Sophia* is still performing satisfactorily since it directs its efforts towards identifying the honest participants, which are persistently identified.

4.2 Preliminaries

4.2.1 Threat Model

In this Chapter, we follow the threat model proposed in Section 3.2.1, adding a couple of novelties. First, in addition to *individual and collusion attacks*, we introduce two new attacks related with dynamic scenarios:

Highly Available Attackers. We can find in the literature DHTs which assume that the longer a node has been on-line, the more likely it is to remain on-line in the near future [14]. This assumption serves as an argument to keep old nodes as entries in the routing table. Although this assumption has been proven to be accurate for identifying stable peers in dynamic networks, it carries out an associated vulnerability: if attackers are significantly more available than regular nodes, they will be in more routing entries as well, inflicting more damage into the network. In other words, taking into account only availabilities, attackers will be preferred as forwarders rather than other nodes just for their high availability. In order to confirm this hypothesis, we will configure scenarios where attackers are highly available while regular nodes follow a specific churn pattern.

Whitewashing Our threat model considers the known problem of *cheap pseudonyms* [66]. The *whitewashing attack* is made feasible by the availability of cheap pseudonyms, where an attacker repeatedly re-joins the network under new identities to avoid the bad reputation associated with its previous behavior. Regarding this attack, we define that an attacker carrying out whitewashing automatically renews its identity after performing ϑ lookups. Thus, a whitewasher might be susceptible of repeatedly being selected as a forwarder, since it periodically escapes from its bad reputation by an identity renewal. In addition, we assume that an identity renewal entails a change in

the physical connection data —i.e. IP address. This prevents the possibility of tracking an attacker by its network address.

In line with Section 3.2.1, users have non-spoofable semi-permanent identifiers to ensure the integrity of the transactions. However, *whitewashers* can switch their identities at any time instant. Assuming this behavior, we will assess the effects of whitewashing on *Sophia*'s performance.

4.2.2 Churn Models and Datasets

To reproduce a dynamic network, we first need data which represents the behavior of nodes leaving and joining the network, that is, an *availability trace*. Clearly, the availability pattern followed by nodes will considerably differ depending on the type of the availability trace adopted. We discern two possible sources of availability traces: (1) Real availability traces, which represent exhaustive measurements of real world systems and, (2) Synthetic availability traces, which are artificially created depending on parameters such as node *lifetime distribution* or node *mean on-line session length*.

Both types of availability traces are valuable: On the one hand, simulations configured with empirical measurements are a better approximation to the real performance that a system may reflect in the real world. On the other hand, the use of synthetic traces let researchers evaluate systems in specific, controlled and even worse situations than existing measurements.

Real Availability Traces. In order to test our system with real world data, we used measurements provided by various research works. Specifically, we considered the following datasets from the literature:

- *Skype Trace*: This trace was obtained by Guha et al. [67] as a result of monitoring 4000 randomly chosen Skype super-peers for one month, beginning at Sep. 12, 2005. The availability test was automatically performed every 30 minutes, providing a reasonably good lifetime resolution.

- *Kad Trace*: The second trace is from eMule's KAD overlay, obtained by Steiner et al. in [5], and describes the behavior of $400,000$ KAD peers, monitored during 6 months. Since our simulator cannot process such amount of concurrent nodes, we used a filtered dataset from this trace containing the $10,000$ peers with longer membership [68].

Synthetic Availability Traces. For synthetically generated traces, we adopted the *Heterogeneous Yao Model* [69]. The mean node on-line session was set to $E[On] = 1$ hour and the disconnection period was also configured to $E[Off] = 1$ hour. Besides, it has been observed that the distribution of user lifetimes in real P2P systems is often *heavy-tailed* (i.e., Pareto), where most users spend minutes per day browsing the network while a handful of other peers exhibit server-like behavior and keep their computers logged in for weeks at a time [70]. For this reason, this trace was produced as a *very heavy-tailed* lifetime distribution. Finally, synthetic traces were obtained using a novel churn generator presented in [71].

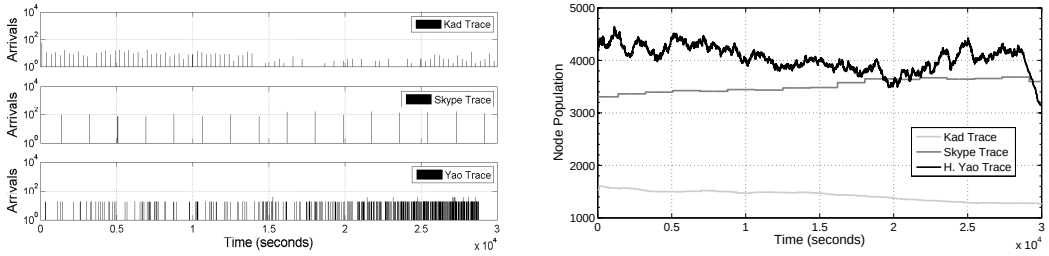


Figure 4.1: Node population and node arrivals of the considered traces during the periods defined for the experiments in the validation section.

4.3 Large-Scale Deployment Considerations

A particularly delicate matter of largely deployed distributed systems is the dynamic node participation, namely, *churn*, [72]. That is, nodes may ungracefully leave and join the system to participate an unpredictable amount of time. In our view, it is crucial to make *Sophia* churn-tolerant for achieving a satisfactory performance in dynamic scenarios. The current section illustrates the necessary changes in *Sophia*'s design to make it churn-tolerant.

4.3.1 Tracking Availability

In order to tolerate churn, we need to keep track of node availabilities in some way. For the sake of simplicity, we incorporated the availability of a neighbor into the Personal History (PH). That is, in addition to the trust information, now we store whether a node is on-line or not.

Another important issue is the procedure to update the availability state of a given node. Our objective is to achieve a clean and simple way to update such information without being intrusive to the DHT, keeping unaltered the pluggable architecture of

Sophia. To do so, we benefit from two existing elements in any DHT: (1) The availability test (Ping/Pong) and, (2) the information about the success or failure of a routing transaction. We implemented the availability tracking for *Sophia* in the Kademlia DHT as case study.

Availability Updated via Ping/Pong test. When a Kademlia node receives any message, it applies a simple maintenance algorithm to the corresponding k-bucket for the sender's ID (see Section 2.1.3 for a detailed description of the algorithm). Basically, this maintenance algorithm decides whether to replace or not the least-recently used node, depending on the result of the Ping/Pong test. In this line, we capture the result of that Ping/Pong test to update the availability state of the least-recently node into the Personal History (PH).

Availability Updated via Lookup Path Success/Failure. Another source of information about node availabilities is the result of a routing transaction. In this case, we expect two possible results from a routing transaction: First, if the construction of a lookup path has been successful, we assume that all nodes involved in such path are on-line. Second, if the path has failed, we analyze the entire path assuming that the last hop is either malicious or off-line. From a local point of view, we cannot differentiate between misbehavior or unavailability. Therefore, we update to false the availability state of the last hop and apply the appropriate trust policy to the entire path (see Fig. 4.2).

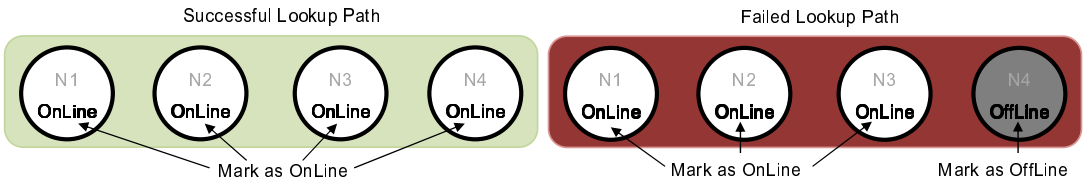


Figure 4.2: Algorithm for updating node availabilities into the PH, using as a source of information the result of a routing transaction.

4.3.2 Churn-Aware Neighbor Selection and Local Trust Policies

To explore the performance of *Sophia* under churn, we need new neighbor selection policies that take as input both trust information and node availabilities. Thus, we designed the following neighbor selection policies:

- *Churn-Aware Select-Best*: In this selection policy, we choose the k available nodes with highest trust scores from the h-bucket.

- *Churn-Aware Probabilistic Selection*: Cooperating and highly available nodes might be overloaded just because they successfully resolve request during long periods of time. To avoid such effect, we propose a neighbor selection policy which probabilistically selects the best on-line forwarders. Each h-bucket contains nodes sorted in decreasing order of their trust values. This policy considers candidates as they are ordered in the h-bucket. Thus, a node n from a certain h-bucket (h_i), with a trust value of $t(n)$, will be selected as a forwarder if $\{t(n) > x\}$ where x is a random variable $\in [0, 1]$. This test will be performed in h_i until collecting the necessary k elements to update the appropriate routing table k-bucket. Note that this policy is prone to select best forwarders, while giving a chance to other candidates.

Regarding the evaluation of nodes, there are not many chances to adapt a local trust policy (see Section 3.3.3) to evaluate nodes depending if they are on-line or not. The main reason is that malicious and off-line nodes are virtually the same from a local point of view. Additionally, there is no point on aggregating the amount of time a node has been on-line because it does not provide reliable information about its immediate future participation. For this reason, we advocate to simplify evaluation by making use of the local trust policies (Pessimistic and Oracle Policies) and the trust calculation algorithm presented in Chapter 3.

4.4 Simulation Framework

Our simulation scenario consists of a Kademlia tree formed by N nodes, identified by m-bit keys, whose lifetimes are described by the input traces. The format of input traces is the `.avt` (availability trace) file format which has been extensively adopted in the literature [5], [67]. Our simulator has been developed in Erlang[1], a programming language conceived for building large scale distributed systems. Besides, the simulator has been carefully implemented and intensively tested during the progress of this research. For each configuration in table 4.1, each simulation was run 10 times and the results were averaged to obtain the plotted values.

In our simulations, every on-line node in the network injects periodically (each δ seconds) lookups to keys chosen uniformly at random. With the exception of the whitewashing scenario, nodes form a fully populated Kademlia tree. In other words, the identifier space is completely covered by nodes even though a fraction of them might be disconnected due to dynamism.

[1] http://www.erlang.com

Table 4.1: General simulation settings.

Parameter	Value
Number of Nodes (N)	8,192 nodes
ID Length (m)	13 bits, 20 bits
k-bucket size (k)	3
Redundant Paths (α)	$[1 \rightarrow 3]$
Sibling Zone (S_z)	0 bits, 11 bits
Simulation Time	30,000 sec.
Transitory Lookups (T_l)	50
Lookup Period (δ)	30 sec.
Stabilization Execution Period (S_t)	5 min.
Ping/Pong Timeout (P_t)	3 sec.
Fraction of Malicious Nodes (f)	$[0.0 \rightarrow 0.6]$
Transactions Until Whitewash (ϑ)	50
h-Bucket Size (h)	15
Short Term History Transactions (d)	3
Short Term History Weight (λ)	0.7

Additionally, a lookup will successfully terminate if any lookup path reaches the requested key *sibling zone* (S_z) (see Section 3.2.2). That is, we consider a lookup as successful if any lookup path reaches a node whose ID shares $m - S_z$ bits in common with the requested key. This consideration is valid even if the ultimate responsible node for a certain key is either malicious or off-line. From our viewpoint, this is a correct criterion of *lookup success* since we consider that the routing process itself has been successful; although the responsible node does not serve the associated data object, which is a storage issue.

In case a node becomes unavailable for a certain period, it completely interrupts its activity until it returns to be available again. To prevent the network size from depleting to zero, when a node exhausts its lifetime, it immediately picks another lifetime randomly selected from the configured availability trace. In addition, the amount of lookups sent by a node is divided in transitory (T_l) and stationary (S_l) lookups. The graphics shown in this section represent the statistical data gathered for stationary lookups; transitory lookups are used to bootstrap the network and are not considered in the evaluation at all. Apart from periodic lookup messages, nodes send messages due to availability tests (Ping/Pong) and stabilization algorithms (see Section 2.1.3). These algorithms, designed to combat the effects of churn, are configured with their specific timeouts (P_t and S_t, respectively).

4.5 Validation

To shed some light on the feasibility of *Sophia* in dynamic and hostile scenarios, we perform a vast set of simulations. In this context, we evaluate three main properties:

First, we assess the Lookup Success Ratio (LSR) which measures the routing reliability provided by each routing technique. Second, we measure the average path length (APL) exhibited by the employed systems. Finally, we analyze how transactions are distributed among nodes, namely, load balancing.

Actually, evaluating the routing reliability provided by a system is, so far, the most relevant point in this section. For the sake of clarity, we decided to measure such property in two ways:

- Lookup Success Ratio (LSR): This metric measures the routing reliability exhibited by a system. This metric is calculated as follows:

$$LSR = \frac{S_{lookups}}{T_{lookups}} \tag{4.1}$$

  where $S_{lookups}$ is the amount of successful lookups achieved by nodes from the total number of lookups sent $T_{lookups}$.

- LSR Gain Ratio: This metric represents the relative LSR performance achieved by a system S_1 compared with another system S_2. Thus, it is calculated as follows:

$$LSR\ Gain\ Ratio = \frac{LSR_{S_1}}{LSR_{S_2}} \tag{4.2}$$

Finally, the proposed simulation framework and metrics have been designed to answer the following questions:

- ¿Does *Sophia* provide a greater routing reliability than redundancy techniques in dynamic and hostile scenarios? (Section 4.5.1)

- Regarding the adaptation of *Sophia* to dynamism: ¿Does it perform as expected in the presence/absence of attackers? (Section 4.5.1)

- ¿How do highly available attackers affect the routing performance? (Section 4.5.2)

- ¿What is the trade-off that *Sophia* poses between reliability and load-balancing? (Section 4.5.3)

- Finally, ¿What happens to *Sophia*'s performance if attackers are not persistently identified by the system? (Section 4.5.4)

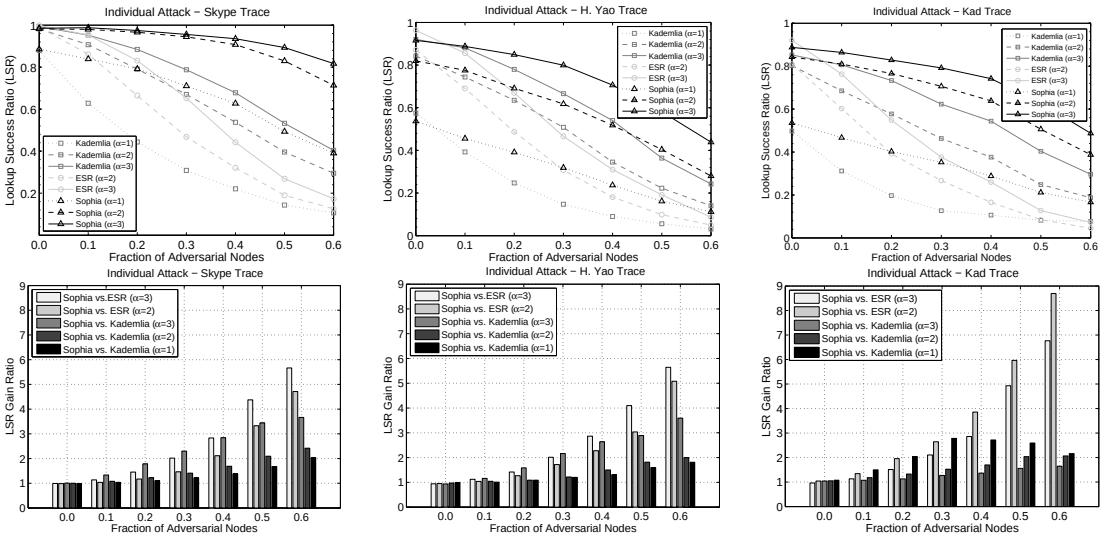


Figure 4.3: Routing performance for several fractions of individual attackers and churn scenarios.

4.5.1 Handling Churn and Adversarial Nodes

The battery of simulations presented in Fig. 4.3 and Fig. 4.4 depicts the routing reliability provided by the evaluated systems under individual and collusion attacks. These attacks are performed in all the proposed churn scenarios in Section 4.2.2. As can be observed, all the systems sufficiently tolerate churn in most cases due to the availability tests and the stabilization algorithm provided by the DHT. However, in cases of high churn rates, results obtained suggest that one path ($\alpha = 1$) is not enough for providing an acceptable routing reliability. Therefore, a certain degree of routing redundancy is needed for any system to tackle with the effects of high churn rates. Besides, irrespective of whether the churn rate is moderated (Skype Trace), high (H. Yao Trace), or even with a large fraction of permanent churn (Kad Trace), all the systems tested exhibit similar reliability numbers when attackers are not present.

However, when misbehaving nodes are infiltrated in the network, the behavior of systems changes dramatically. This fact is clear observing LSR values depicted in Fig. 4.3. For instance, regarding the Skype Trace, Kademlia for $\alpha = 2$ reduces its reliability approximately in 35% when $f = 0.3$. In this line, Equally-Spaced Replication (ESR) reports even worse performance values, losing over 50% of its routing reliability, compared with a scenario without attackers. The main reason why ESR provides lower LSR numbers compared with Kademlia is argued in [31]: although ESR paths do not share common nodes aiming to avoid faulty nodes as much as possible, paths are forwarded by one node at each step. This means that a single malicious node is able to

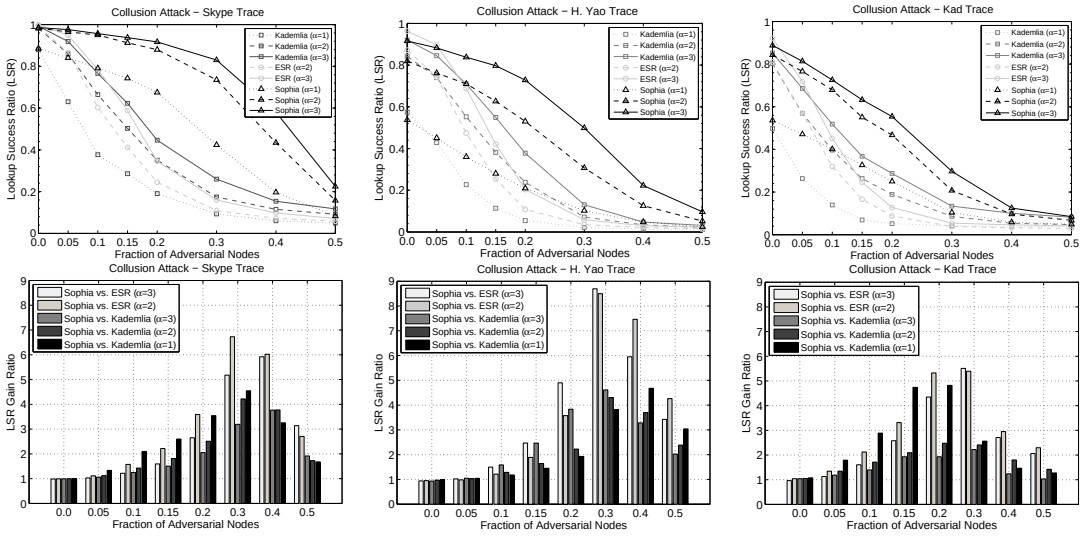


Figure 4.4: Routing performance under collusion attack and churn.

destroy an entire path. In [31], this approach of redundant routing is declared less resilient against malicious nodes than the wide paths technique, adopted by Kademlia (see Section 2.3). Our experiments clearly support this argument.

On the other hand, we can see how *Sophia* provides a higher level of routing reliability in the presence of adversarial nodes compared with Kademlia and ESR techniques. Furthermore, *Sophia*'s LSR numbers are several times higher than its counterparts in some cases, when the fraction of faulty nodes is pronounced. This fact yields an important remark: routing redundancy is much more effective if we take into account the quality of selected forwarders. This remark is illustrated in the LSR Gain Ratio graphics where, in general, the LSR Gain Ratio (Fig. 4.3) of *Sophia* versus Kademlia and ESR increases as f attains larger values.

In Fig. 4.4 we present the results obtained after introducing in the network several fractions of collusive nodes. It is clear that this threat model is significantly more aggressive than an individual attack, as we can infer from the LSR numbers provided by evaluated systems. In such scenario, maintaining an historical tracking of nodes is considerably beneficial: for example, in case of $f = 0.2$ and $\alpha = 2$, *Sophia* obtains LSR gains ranging from 2 to 5 times the performance of its counterparts, irrespective of the availability trace configured. Nevertheless, when f reaches its highest values the effects of the malicious coalition are dramatic in all cases. This is specially evident in the Kad Trace scenario, where in addition to the malicious coalition, routing failures are increased by a low average node availability (approximately 30%).

Fig. 4.5 provides a wide vision about the impact of malicious nodes on the successful lookup Average Path Length (APL). As can be appreciated in the graphic matrix,

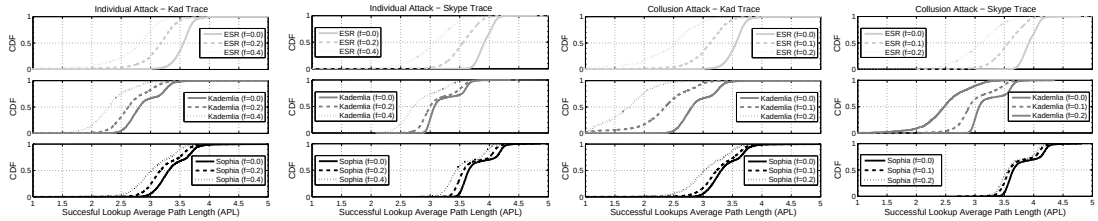


Figure 4.5: Impact on the Average Path Length of varying the fraction of individual and collusive nodes ($\alpha = 2$).

adversarial nodes produce a lower impact to *Sophia*'s lookup APL than to the studied redundancy techniques. The explanation for this effect is clear: while tested redundancy techniques suffer the consequences of an important fraction of attackers, *Sophia* avoids selecting attackers as forwarders after exchanging several routing operations. Thus, the fraction of attackers which affects *Sophia*'s lookup APL becomes smaller as their trust values reflect their behavior. These graphics are coherent with results obtained in figures 4.3 and 4.4.

We summarize this set of simulations with the following conclusions: (i) In the absence of attackers, the underlying redundancy scheme is not as important as the mechanisms intended to keep fresh neighbors into the routing table. Therefore, if a DHT keeps updated routing tables and employs a certain degree of routing redundancy, the lookup success can be guaranteed even in highly dynamic scenarios. (ii) *Sophia* has been successfully adapted to tolerate churn since, in the absence of attackers, it shows virtually the same LSR numbers than Kademlia and ESR. (iii) In the presence of attackers, routing redundancy itself has poor routing success guarantees in comparison with its cost. In other words, *Sophia* demonstrates that routing redundancy can be much more effective taking into account the quality of nodes involved in routing paths.

4.5.2 Highly Available Attackers

In this part of the validation we present the effects of varying the fraction of individual attackers that deliberately remains on-line along the whole simulation process. As can be inferred from Fig. 4.6, this threat model is remarkably aggressive, specially in scenarios with high churn rates. The main reason behind this phenomenon is that, once a regular node incorporates an attacker into its routing table, it is kept always there, because it satisfactorily replies to availability tests at any moment. This fact is clear in the Heterogeneous Yao Trace, where the high churn rates suffered by regular nodes give more opportunities to attackers of being incorporated into other's routing

tables. Consequently, these experiments confirm our thesis that availability should not be the only property to take into account in order to keep neighbors into the routing table.

In Fig. 4.6 it is appreciable the significant routing reliability gain that *Sophia* obtains, compared with the other techniques. That is, in the case of the Heterogeneous Yao Trace for $f = 0.3$ and $\alpha = 3$, LSR values provided by ESR and Kademlia are over 15% and 20%, respectively. However, *Sophia* is delivering messages with a LSR of 65%, under the same conditions. It should be noted that one important factor related with the performance of *Sophia* is the network traffic conditions. In other words, the more transactions a node performs, the more accurate trust ratings it is able to obtain. We simulated traces of 8 hours where nodes inject lookups each 30 seconds while on-line. This amount of traffic is much more than sufficient for enabling *Sophia* to distinguish between cooperating and adversarial nodes.

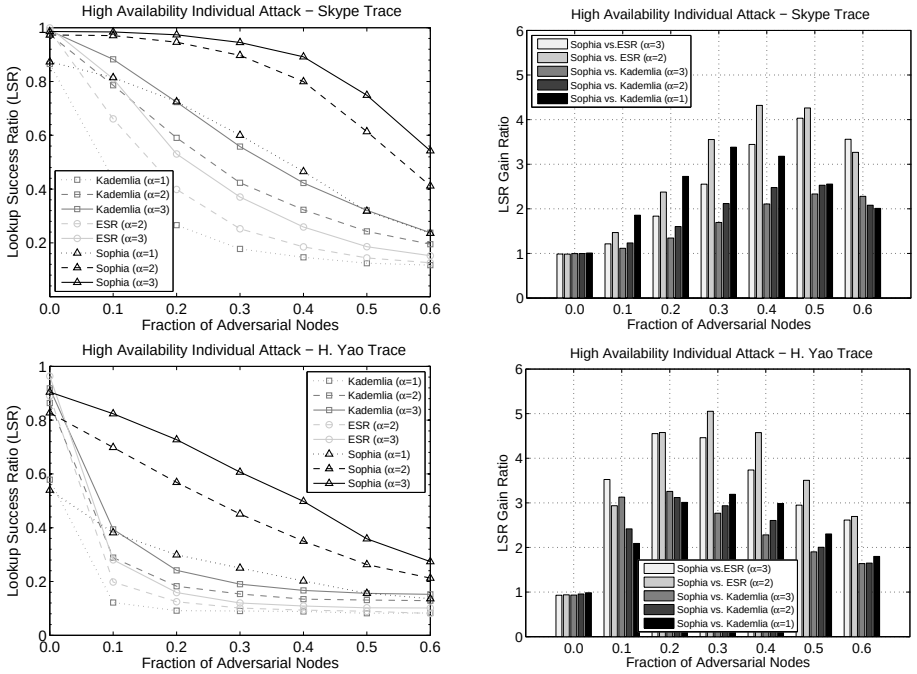


Figure 4.6: Simulations concerning to varying the fraction of highly available attackers under churn.

An important aspect which is specially noticeable in this experiment (also appreciable in 4.5.1) is the network traffic savings that *Sophia* can potentially provide. In the context of routing redundancy, communication costs are mainly determined by the number of parallel paths needed to meet a specific routing reliability threshold. However, *Sophia* dynamically reinforces routing irrespective of the number of parallel paths employed. In this regard, in Fig. 4.6 we can see how *Sophia* configured with $\alpha = 2$ has

a much greater LSR than Kademlia and ESR configured with $\alpha = 3$, in both scenarios. Considering this simple comparison, it is not hard to see that our system clearly outperforms the tested redundancy techniques while reducing the redundancy overhead in 1 path. Intuitively, this means over a 33% of network traffic reduction. Therefore, taking into account the *quality of paths* is an efficient manner of reducing the necessity of redundancy for providing secure routing. Consequently, the underlying DHT will be considerably more scalable.

Let us draw a couple of remarks from this set of experiments: (i) First, although availability is an important information about nodes, it should not be the only criterion used to build the routing table. In our view, this has been confirmed analyzing the damage caused by several fractions of highly available attackers. (ii) As an important insight, in addition to fortify routing, *Sophia* is also capable of significantly reducing routing redundancy to reach a specific LSR threshold. This fact yields considerable network traffic savings.

4.5.3 Neighbor Selection and Trust Policies

The main reason for the success of *Sophia* is to persistently rate the quality of a node's neighbors and use this information to increase the probability of routing successfully through a neighbor selection policy. However, a strictly greedy neighbor selection policy may overload stable and cooperating nodes. This fact can eventually saturate these nodes, affecting thus the overall network performance. We want to explore how does affect the *Sophia*'s routing reliability do not greedily select best forwarders. For this reason, we executed simulations configuring *Sophia* with a non-greedy neighbor selection policy (see Section 4.3.2): Select Probabilistic (SP). Additionally, we analyze the performance of the Pessimistic Trust Policy against the theoretically best local trust policy, the Oracle.

Fig. 4.7 shows the LSR and Load Balancing obtained by *Sophia* configured with several policies, adopting the Kad Trace as availability pattern. Let us discuss about the behavior of *Sophia* when it is configured with the Pessimistic Trust Policy. In this case, the SP neighbor selection obtains modest LSR gains compared with Kademlia, for the same number of parallel paths. However, its performance is significantly worse than the Select Best neighbor selection (SB). In our opinion, there are two main reasons for such a difference in LSR numbers when *Sophia* is configured with SB and SP policies. First, the effect of churn makes trust values of nodes to be, in general, lower compared with a non-dynamic scenario. In other words, trust values of cooperating nodes are lower under dynamism since they also are responsible for failed paths when they

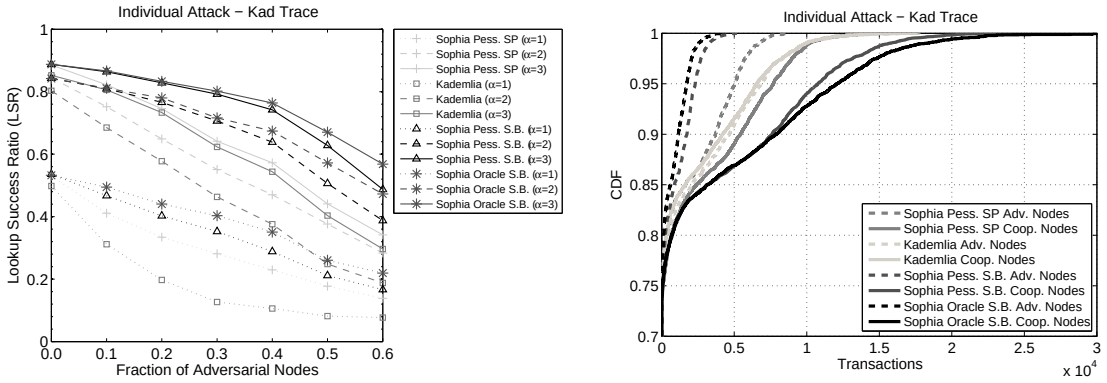


Figure 4.7: LSR and received transactions CDF of Kademlia and *Sophia* configured with Select Best (SB) and Select Probabilistic (SP) neighbor selection policies.

become off-line and the Pessimistic Trust Policy rates them negatively. Hence, when the SP policy tries to pick k nodes to fill up the appropriate k-bucket, the random trials for the first k candidates in the h-bucket tend to fail more often than desired. This means that adversarial nodes, placed at the back of the h-bucket, are sometimes selected to fill the routing table. Second, the h-bucket size plays an important role here. Configuring SB neighbor selection, a value of $h = 15$ historical candidates for routing bucket size of $k = 3$ suffices to find α on-line and cooperating nodes, in most cases. However, the value of h seems not to be large enough for the SP policy to share the load among cooperating nodes and maintain LSR values close to SB policy. Therefore, in case of deploying *Sophia* using a non-greedy neighbor selection, the h-bucket size must be larger than it would be in case of a greedy neighbor selection policy.

On the other hand, it is important to note that the distribution of transactions received by nodes is more fair in the case of SP than SB neighbor selection (Fig. 4.7). That is, the load assigned to cooperating and stable nodes is less abusive configuring *Sophia* with SP, avoiding these nodes to be eventually flooded by routing requests. As depicted in Fig. 4.7, the received transactions CDF corresponding to Kademlia presents the same shape, irrespective of whether nodes are malicious or not. Such result was expected since only *Sophia* is capable of discerning between both types of nodes.

As we expected, the Oracle shows the best LSR values and, consequently, the least fair load balancing in Fig. 4.7. However, performance differences between the Pessimistic policy and the Oracle are not really important. This means that the Pessimistic policy is an accurate approach to rate forwarders, even in dynamic scenarios.

The objective of this experiment is not to propose the optimal non-greedy neighbor selection policy, but rather to provide an insight about the relationship that *Sophia*

poses between routing reliability and load-balancing. In conclusion, as expected we are witnessing a trade-off between routing reliability and proper load balancing. This should be sorted out depending on the specific requirements of the system where *Sophia* would be deployed.

4.5.4 Whitewashing

In the final part of the experimentation, we want to address the impact of whitewashing on *Sophia*'s performance. Thus, this threat model assumes that attackers change their identities in the system after performing ϑ lookups. However, since *Sophia* assumes that nodes incorporate a mechanism to persist their identities, honest nodes will keep their identifiers along the simulation.

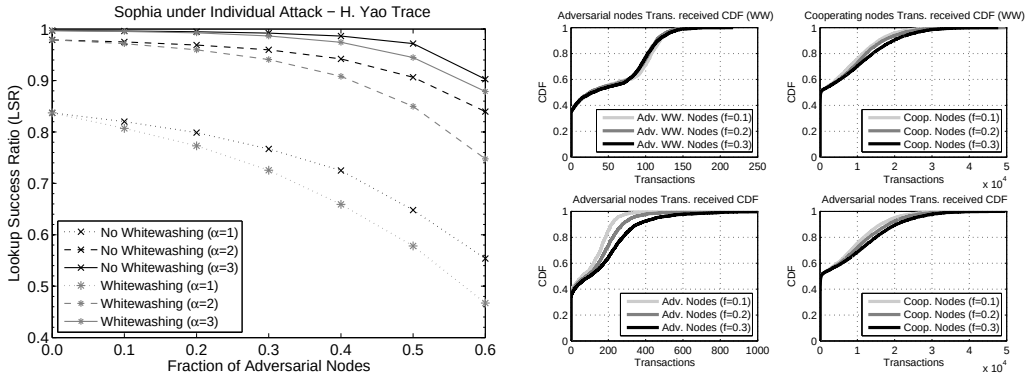


Figure 4.8: Performance of *Sophia* under individual attacks of withewashing/no withewashing nodes.

This experiment has a slightly different parametrization than previous ones. First, since the main objective of this test is to analyze the impact of whitewashers, we need to construct a non-complete overlay for leaving a large fraction of identifiers unused. Thus, attackers can choose several identities along the simulation.

Second, in this experiment we make use of a sibling zone of $S_z = 11$. This is needed for avoiding that stale identities produced by the whitewashing activity falsely increase lookup success. That is, in the particular case of Kademlia overlay, when nodes are spread over the identifier space, their lowest *k*-buckets tend to be empty. Thus, if attackers periodically change their identities they will slowly filling up a fraction of these lowest of other's routing tables. The collateral effect of this fact is that, virtually, routing tables of nodes will be more complete than in the absence of whitewashers, making a lookup reach a certain key with a higher probability. Introducing an appropriate sibling zone, lookups do not need to pass trough routing tables lowest *k*-buckets, providing then a truthful notion of routing success.

As we can see in Fig. 4.8, for small fractions of whitewashing attackers, the routing reliability provided by *Sophia* is just slightly affected. The main reason for such a difference in LSR numbers is that an identity renewal is, in the end, a kind of churn. This produces the existence of more stale elements into the PH compared with the scenario without whitewashing. This yields an additional source of lookup failures. However, if we observe the attackers' received transactions CDF we realize that they do not take advantage of whitewashing to disrupt more routing operations. Conversely, they receive fewer transactions. The main reason behind this phenomenon is that, when a whitewasher leaves and rejoins the system, other nodes are already identified as cooperating forwarders. Additionally, h-buckets of nodes are often full when attackers enter as new nodes and this fact prevents further forwarding attacks.

As a final conclusion, *Sophia* can successfully tolerate a small or moderate fraction of whitewashers, whenever the majority of participants are persistently identified.

4.6 Final Remarks

In this chapter, we explored the feasibility of *Sophia* under dynamic and hostile conditions. To this end, we designed mechanisms to make *Sophia* tolerant to churn. In our simulations, we compared *Sophia* with well-known routing redundancy techniques. The validation framework included several types of threat models and churn characterized by real and synthetic traces. Additionally, we introduced attackers with non-persistent identities (whitewashing), a critical aspect of trust systems in P2P settings.

Our simulations evince that *Sophia* clearly outperforms pure routing redundancy techniques in hostile and dynamic scenarios. For instance, in general, *Sophia* provides a routing reliability ranging from 1.5 up to 5 times larger than the considered redundancy techniques for the same number of parallel paths. In other words, *Sophia* requires fewer parallel paths to tolerate malicious nodes with the same reliability than pure routing redundancy techniques. This fact yields important network traffic savings, increasing the scalability of the underlying DHT substrate.

Finally, we verified that *Sophia* is capable to correctly deal with the effects of *whitewashers*. As expected, *Sophia* nodes spend their efforts on identifying cooperating forwarders. Consequently, attackers cannot benefit from whitewashing to disrupt more routing operations, since other nodes are already identified as cooperating forwarders. This means that *Sophia* can successfully tolerate a small or moderate fraction of whitewashers, whenever the majority of participants are persistently identified.

5

Conclusions and Future Work

Summary of this Thesis

This thesis has been devoted to mitigate the effects of malicious participation on the reliability of routing in DHTs.

In a spirit of contextualizing our work, we provided a solid background covering several topics. We thoroughly described the functioning and main properties of DHTs. We motivated our research by introducing the wide spectrum of attacks a DHT is exposed to. We depicted some redundancy-based countermeasures, pointing out their associated network costs. Then, we discussed the relevance of persistently identifying nodes within a system. In the last part of the background, we described the architecture of trust and reputation systems, complemented with a general taxonomy of them depending on their source of information.

The main contribution of this thesis is *Sophia*: a system aimed at reinforcing routing using a local trust system. In *Sophia*, each node synthesizes direct observations on the routing behavior of its neighbors in form of trust ratings as a result of its routing operations. These ratings are persistently stored and updated in a data structure called *Personal History*. As more observations are available, the ratings about neighbors become progressively more reliable and eventually converge, enabling nodes to identify which forwarders are more likely to route successfully towards the destination. Moreover, *Sophia* is a generic security technique and therefore it is applicable to any DHT. Since the selection of the next hop is individually done by each host, nothing prevents a regular participant from communicating with any computer armed with *Sophia* and *vice versa*. Additionally, the use of iterative routing gives to *Sophia* the property of improving the quality of routing paths with no extra overhead.

We compared our solution with other well-known redundancy techniques in both stable and dynamic scenarios, and under different attack models:

- *Stable Networks:* First, *Sophia* provides a greater routing reliability that redundancy techniques in the presence of attackers. For instance, compared with

Kademlia redundant routing, Sophia is able to increase the probability of routing successfully from 10% to 35% for the same degree of redundancy. Moreover, *Sophia* continuously evolves towards better quality paths as it receives more feedback from transactions. Also, *Sophia* is capable to dynamically recover from unexpected attacks, restoring the routing reliability lost during the attack.

- *Dynamic Networks:* Regarding routing reliability, *Sophia* clearly outperforms pure routing redundancy techniques against adversarial participants in dynamic scenarios. In most cases, given the same number of parallel paths, *Sophia* provides a routing reliability ranging from 1.5 - 5 times larger than the considered redundancy techniques. In addition, we verified that *Sophia* is capable to correctly mitigate the effects of whitewashers.

In both scenarios, *Sophia* requires fewer parallel paths to tolerate malicious nodes than pure routing redundancy techniques. This fact yields important network traffic savings, increasing the scalability of DHTs intended to provide a reliable lookup service.

Conclusions

Firstly, we argued the importance of persistently identifying nodes within a DHT. From our perspective, future DHT implementations, either totally or partially deployed in an open environment, should benefit from research efforts to produce long-lived and secure identities. With this substrate, new horizons arise to devise novel security techniques intended to mitigate attacks against DHTs.

In this line, we presented *Sophia*. *Sophia* is the first representative example of a local trust system devised to secure routing in DHTs. By identifying nodes within the system, *Sophia* is able to provide a notion of *routing quality* in terms of forwarder reliability.

After a thorough research and exhaustive simulations, we state that considering the quality of intermediate forwarders in lookup paths significantly increases lookup success probabilities. In other words, routing redundancy can be much more effective taking into account the quality of nodes involved in the forwarding process. Consequently, the routing redundancy required to offer reliable routing can be remarkably reduced. This fact significantly diminishes the network overhead introduced by using parallel routing paths.

In our opinion, this thesis represents a step further towards the achievement of reliable lookup services in open DHTs.

Future Research Lines

The present thesis covers several topics of the proposed research line. However, there are still further issues to investigate:

- *Incentives*: By the moment, *Sophia* does not define any reward/punishment response directed to cooperating/malicious participants. A natural extension of *Sophia* involves the introduction of routing incentives.
- *Characterization of routing attacks in real DHTs*: Exploring existing and real world routing vulnerabilities is mandatory for implementing effective countermeasures. For this reason, we are currently working on a *crawler* intended to analyze routing vulnerabilities of Bittorrent MainLine DHT. The objective of this crawler is to inspect the routing attacks we can find in real systems and gather relevant statistical data: lookup success ratios, mean latencies or fraction of malicious participants.
- *Sophia implementation for open DHTs*: The final stage of any thesis should be transferring gained research insights into a real implementation. To this end, we are considering to implement *Sophia* as a standalone library. Thus, real DHT implementations could able to benefit from our system and provide a more reliable lookup service.

Implementation Notes of this Thesis

In this thesis, we implemented a simulator to demonstrate the validity of our approach. We developed our simulator in Erlang[1], a programming language devised to develop large scale distributed systems.

Erlang

Erlang is a programming language used to build massively scalable soft real-time systems with requirements on high availability. Some of its uses can be found in telecommunications, banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for concurrency, distribution and fault tolerance.

Erlang is, in essence, a functional programming language, but there is also a large emphasis on concurrency and reliability. To be able to support hundreds of tasks running at the same time, Erlang uses the actor model, where each actor is a separate process in the virtual machine. Additionally, inter-process communication is provided in a asynchronous message passing fashion. That is, individual processes send messages among them to coordinate their execution, avoiding traditional synchronization mechanisms.

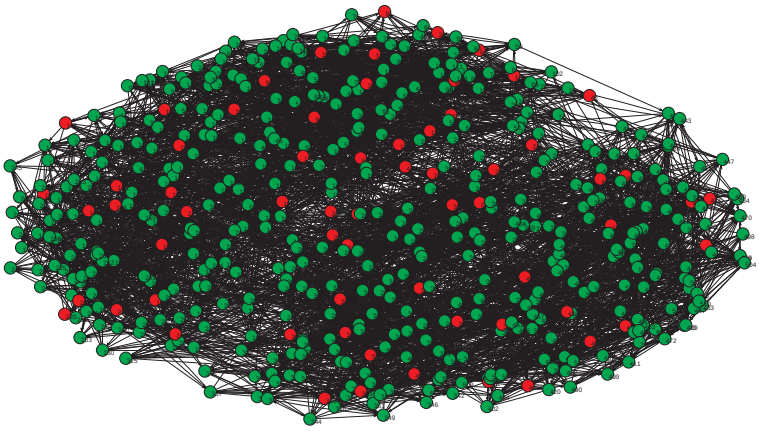


Figure 1: Kademlia overlay formed by 512 nodes, where a 10% of nodes are attackers (red ones). This network screen-shot has been exported from our simulator using the .net format (Pajek).

[1] www.erlang.org

Simulator Design

In our simulator, there is an initial process responsible for instantiating and bootstrapping all the network processes (i.e. nodes). Subsequently, *node processes* start to serve and request lookups. In addition, the way a node serves lookup requests is conditioned by its role within the network (attacker/cooperating). Every node process gathers useful data of its execution (e.g. number of lookups served/requested). After a defined period of time, node processes interrupt their execution for collecting statistics from the simulation. Note that node processes serve and request lookups in a concurrent manner. More technically, the simulator's parallelism is given by the Erlang platform (virtual machine and processors), instead of implementing an event-based approach (e.g. priority queue) to digest events from several processes.

In Erlang, processes are cheap to create. For this reason, is a good practice encapsulating specific tasks into processes. Our simulator employs processes for several tasks: identity service, statistical data gathering or lifetime service, to name a few.

Over $5,000$ lines of code are devoted to implement the Kademlia protocol, routing attacks, redundancy techniques, *Sophia*, statistical data management, availability traces parsing and processing, etc. It is worth mentioning that starting a simulator from scratch was considered after experiencing problems with our initial simulator Omnet++ [73]. To analyze the correct construction of the network and its properties, our simulator is capable to export a screen-shot of the network in a `.net` format (Fig. 1), employed by the Pajek tool [74].

Our simulator has been tested with more than $16,000$ concurrent nodes in a single computer, obtaining satisfactory performance results. Moreover, thanks to the Erlang name-space middleware, our simulator could be easily extended for working in a distributed manner. Thus, the scale of our simulations in a controlled environment, such a university laboratory with high-speed dedicated network, could reach thousands of concurrent nodes.

The code of our simulator can be accessed at `http://ast-deim.urv.cat/svn/olsr/trunk/kademlia/`.

References

[1] Gnutella. http://en.wikipedia.org/wiki/Gnutella, 2006.

[2] J. M. Kleinberg. Small-world phenomena and the dynamics of information, 2002.

[3] eMule KAD Project. http://www.emule-project.net/, 2002.

[4] Azureus Mainline DHT. http://azureus.sourceforge.net/, 2005.

[5] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A global view of kad. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 117–122, 2007.

[6] Chao Zhang, Prithula Dhungel, Di Wu, and Keith W. Ross. Unraveling the bittorrent ecosystem. *IEEE Transactions on Parallel and Distributed Systems*, 22:1164–1177, 2011.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[8] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. In *NSDI'04*, pages 18–18, 2004.

[9] Donald Kossmann, Tim Kraska, Simon Loesing, Stephan Merkli, Raman Mittal, and Flavio Pfaffhauser. Cloudy: a modular cloud storage system. *Proc. VLDB Endow.*, 3:1533–1536, 2010.

[10] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common api for structured peer-to-peer overlays. In *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 33–44. 2003.

[11] G. Urdaneta, G. Pierre, and M. van Steen. A survey of dht security techniques. *ACM Computing Surveys*, 2009, in press.

[12] Marc Sánchez-Artigas, Pedro García-López, and Antonio Skarmeta. Secure forwarding in dhts - is redundancy the key to robustness? In *Euro-Par 2008 – Parallel Processing*, volume 5168, pages 611–621. 2008.

[13] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[14] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS'02*, pages 53–65, 2002.

[15] Matei Ripeanu, Ian T. Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *CoRR*, cs.DC/0209028, 2002.

[16] R. Rivest. Rfc 1321: The md5 message-digest algorithm, 1992.

[17] D. Eastlake and P. Jones. Rfc 3174: Us secure hash algorithm 1 (sha1), 1992.

[18] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, 2001.

[19] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware'01*, pages 329–350, 2001.

[20] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Berkeley, CA, USA, 2001.

[21] I. Stoica et. al. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, pages 149–160, 2001.

[22] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, 1997.

[23] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a million user dht. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 129–134, 2007.

[24] J. R. Douceur. The sybil attack. In *IPTPS'02*, pages 251–260, 2002.

[25] Brian Neil, Levine Clay Shields, and N. Boris Margolin. A survey of solutions to the sybil attack, 2006.

[26] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 261–269. 2002.

[27] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstro. Defending against eclipse attacks on overlay networks. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, 2004.

[28] C. Harvesf and D. M. Blough. The effect of replica placement on routing robustness in distributed hash tables. In *Peer-to-Peer Computing (P2P'06)*, pages 57–66, 2006.

[29] Salma Ktari, Mathieu Zoubert, Artur Hecker, and Houda Labiod. Performance evaluation of replication strategies in dhts under churn. In *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, MUM '07, pages 90–97, 2007.

[30] M. Sánchez-Artigas, P. García-López, and A. Skarmeta. A novel methodology for constructing secure multipath overlays. *IEEE Internet Computing*, 9(6):50–57, 2005.

[31] Kirsten Hildrum and John Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Distributed Computing*, volume 2848 of *Lecture Notes in Computer Science*, pages 321–336. 2003.

[32] I. Baumgart and S. Mies. S/kademlia: A practicable approach towards secure key-based routing. In *ICPADS'07*, volume 2, pages 1–8, 2007.

[33] D. Cerri, A. Ghioni, S. Paraboschi, and S. Tiraboschi. Id mapping attacks in p2p networks. In *Global Telecommunications Conference, GLOBECOMï¿½05.*, 2005.

[34] P. Wang, I. Osipkov, N. Hopper, and Yongdae Kim. Myrmic: Secure and robust dht routing. In submission, 2007.

[35] A. Selcuk, E. Uzun, and M.R. Pariente. A reputation-based trust management system for p2p networks. In *Fourth International Workshop on Global and Peer-to-Peer Computing*, pages 20–21, 2004.

[36] F. Lesueur, L. Me, and V.V.T. Tong. An efficient distributed pki for structured p2p networks. In *Peer-to-Peer Computing (P2P'09)*, pages 1–10, 2009.

[37] Tomas Isdal, Micheal Piatek, Arvind Krishnamurthy, and Thomas Anderson. Privacy-preserving p2p data sharing with oneswarm. Technical report, Dept. of Computer Science and Engineering, Univ. of Washington, 2009.

[38] K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in peer-to-peer systems. *IEEE Transactions on*

Knowledge and Data Engineering, 16:858–869, 2004.

[39] David Hales. Applying evolutionary approaches for cooperation. *Cognitive Wireless Networks*, pages 63–74, 2007.

[40] D. Hales and S. Arteconi. Friends for free: Self-organizing artificial social networks for trust and cooperation. Technical report, Department of Computer Science (University of Bologna), 2005.

[41] M. Sánchez-Artigas and P. García-López. On routing in distributed hash tables: Is reputation a shelter from malicious behavior and churn? In *Peer-to-Peer Computing (P2P'09)*, pages 31–40, 2009.

[42] S. Marti and H. Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472–484, 2006.

[43] K. Hoffman, D. Zage, and C. Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Comput. Surv.*, 42(1):1–31, 2009.

[44] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[45] Audun Jøsang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th Bled Conference on Electronic Commerce*, 2002.

[46] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Commun. ACM*, 42:39–41, February 1999.

[47] Bin Yu, M.P. Singh, and K. Sycara. Developing trust in large-scale peer-to-peer systems. In *MASSUR'04*, pages 1–10, 2004.

[48] eBay. `http://www.ebay.com/`, 2001.

[49] Daniel Houser and John Wooders. Reputation in auctions: Theory, and evidence from ebay. *Journal of Economics and Management Strategy*, 15(2):353–369, 2006.

[50] Schlosser M. T. Kamvar, S. D. and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*, pages 640–651, 2003.

[51] A. Nandi, T. Ngan, A. Singh, P. Druschel, and D. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Middleware'05*, pages 270–291. 2005.

[52] S. Marti, P. Ganesan, and H. Garcia-Molina. Dht routing using social links. In *IPTPS'04*, pages 100–111, 2005.

[53] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer. Authentication-free fault-tolerant peer-to-peer service provisioning. In *DBISP2P'07*, 2007.

[54] Runfang Zhou and Kai Hwang. Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing. *IEEE Transactions on Parallel and Distributed Systems*, 18:460–473, 2007.

[55] S. Marti and H. Garcia-Molina. Limited reputation sharing in p2p systems. In *EC '04*, pages 91–101, 2004.

[56] L.M. Aiello, M. Milanesio, G. Ruffo, and R. Schifanella. Tempering kademlia with a robust identity based system. In *Peer-to-Peer Computing (P2P'08)*, pages 30–39, 2008.

[57] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36:335–371, 2004.

[58] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.

[59] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi. Peer-to-peer resource discovery in grids: Models and systems. *Future Generation Computer Systems*, 23(7):864–878, 2007.

[60] E. Pacitti, P. Valduriez, and M. Mattoso. Grid data management: Open problems and new issues. *Journal of Grid Computing*, 5:273–281, 2007.

[61] R. Buyya, R. Ranjan, and R. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and Architectures for Parallel Processing*, volume 6081, pages 13–31. 2010.

[62] Justin Cappos, Ivan Beschastnikh, Arvind Krishnamurthy, and Tom Anderson. Seattle: a platform for educational cloud computing. *SIGCSE Bull.*, 41:111–115, 2009.

[63] A. Singh, T. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *INFOCOM'06*, pages 1–12, 2006.

[64] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20(2):127–138, 2008.

[65] Gerald Oster, Pascal Molli, Sergiu Dumitriu, and Ruben Mondejar. Uniwiki: A collaborative p2p system for distributed wiki applications. *Enabling Technologies, IEEE International Workshops on*, 0:87–92, 2009.

[66] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. *Selected Areas in Communications, IEEE Journal on*, 24(5):1010–1019, 2006.

[67] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the skype peer-to-peer voip system. In *International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[68] L. Pàmies-Juarez, P. García-López, and M. Sánchez-Artigas. Availability and redundancy in harmony: Measuring retrieval times in p2p storage systems. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10, 2010.

[69] Zhongmei Yao, D. Leonard, Xiaoming Wang, and D. Loguinov. Modeling heterogeneous user churn and local resilience of unstructured p2p networks. In *Network Protocols (ICNP '06)*, pages 32–41, 2006.

[70] Derek Leonard, Vivek Rai, and Dmitri Loguinov. On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. *SIGMETRICS Perform. Eval. Rev.*, 33:26–37, June 2005.

[71] E. Fernández-Casado, M. Sánchez-Artigas, and P. García-López. Affluenza: Towards universal churn generation. In *Peer-to-Peer Computing (P2P'10)*, pages 1–2, 2010.

[72] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 10–10, 2004.

[73] A. Varga. The omnet++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001)*, 2001.

[74] Pajek Program for Large Network Analysis. `http://vlado.fmf.uni-lj.si/pub/networks/pajek/`, 2003.