Sophia: A local trust system to secure key-based routing in non-deterministic DHTs

Raúl Gracia-Tinedo *, Pedro García-López, Marc Sánchez-Artigas
Universitat Rovira i Virgili, Department of Computer Engineering and Mathematics, Av. Països Catalans 26, 43007 Tarragona, Spain

A R T I C L E   I N F O
Article history:
Received 29 July 2011
Received in revised form
21 May 2012
Accepted 28 July 2012
Available online xxxx

Keywords:
Large-scale distributed systems
Distributed hash tables
Security
Trust and reputation systems

A B S T R A C T
Today, many distributed applications are typically deployed at a large scale, including Grid, web search engines and content distribution networks, and it is expected for their scale to grow more in terms of number of machines, locations and administrative domains. This poses many scalability issues related to the scale of the environment they run in. To explicitly address these issues, many distributed systems and everyday services use peer-to-peer (P2P) overlays to allow other parts of the system to benefit from the fault-tolerance and scalability of P2P technology. In particular, Distributed Hash Tables (DHTs), which implement a simple put-and-get interface to a dictionary-like data structure, have been extensively used to overcome the current limitations associated with the centralized and hierarchical components of distributed systems, including data management, resource discovery, job scheduling etc.

However, DHTs exhibit a number of security problems in large-scale systems, where a large number of users are unknown to administrators (e.g., desktop grids). This makes the detection of malicious behavior an extremely complex task. As a result, attackers can disrupt the system in very dangerous ways, leading ultimately to the failure of the routing service, which is catastrophic for any DHT. To address this issue, we introduce Sophia, a new security technique which combines iterative routing with local trust to implement a secure lookup service with almost zero overhead. The key aspect to incur zero overhead is the use of local trust. In Sophia, each user identifies which routing entries are cooperative based on the success and failure of his own lookups, so no trust information is shared. Our simulation results demonstrate that Sophia does better than existing state-of-the-art solutions for secure routing in DHTs, both in stable and high dynamic environments, and even for collusive threat models.

© 2012 Elsevier Inc. All rights reserved.

1. Introduction

Despite the maturity of distributed systems, large-scale distributed applications such as open Grids, Clouds for completely distributed applications and on-line storage systems have yet to become a commodity technology in modern computing environments due to their scale and volatility of resources (i.e., arrival, departure and failure). To face these issues, researchers who build these infrastructures have resorted to Peer-to-Peer (P2P) techniques to increase the scalability and availability of their platforms, among which Distributed Hash Tables (DHTs) have been extensively adopted to develop many practical systems. To wit, DHTs have been used to implement some key components of Grid platforms such as resource discovery [55,24,8,5] and data management [33,42]. Further, European projects such as Grid4All [18] and XtremeOS [60] also use DHTs for several purposes. For example, XtremeOS targets large-scale dynamic Grid infrastructures and its goal is to facilitate the use and management of Grid resources. In particular, XtremeOS provides an implementation of two DHTs, Overlay Weaver and Scalaris, to build a scalable resource discovery service. In general, a large body of literature has been devoted to exploit the search performance of DHTs [46,52,5,43].

Although not all services should be implemented in a decentralized fashion (e.g., one can make a strong case against the full decentralization of a key management system such as Kerberos), DHTs have become a building block to scale wide-area services. DHTs provide a decentralized key-value (k, v) store that supports two basic operations: put(k, v) and v = get(k), which allow one to store and retrieve (k, v) pairs from a common key space.

* Corresponding author.
E-mail addresses: raul.gracia@urv.cat (R. Gracia-Tinedo), pedro.garcia@urv.cat (P. García-López), marc.sanchez@urv.cat (M. Sánchez-Artigas).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.07.009
DHTs consist of many autonomous peers that self-organize as an overlay network on top of large autonomous networks. Each peer is responsible for a certain portion (zone) of the key space and forwards any operation to an adjacent peer whose zone is closer to the key, until the operation reaches the peer that is responsible for the key. Representatives examples of DHTs are CAN [37], Chord [51], Pastry [40] and Kademlia [31]. They differ primarily in the structure of the key space, routing path selection, and entries in the routing table, but have in common that all the get and put operations are realized on top of a basic key-based routing layer. According to their routing geometries [20], DHTs can be classified into deterministic and non-deterministic. In a deterministic DHT, the routing tables are unique for the same set of nodes. Examples of this class of DHTs are CAN [37] and Chord [51]. However, in a non-deterministic DHT, there exists some flexibility in the selection of routing entries, so that a given node can present multiple valid routing tables for the same set of participants. Examples of non-deterministic DHTs are Kademlia [31], Pastry [40] and Tapestry [63].

The aim of key-based routing (KBR) is to route a message tagged with key $k$ to the node responsible for that key. This functionality is the same as that of the lookup($k$) interface first proposed in the Chord paper and adopted in [10] as the KBR interface (see Fig. 1). In the case of a get($k$) operation, the requester executes the procedure lookup($k$) to progressively find closer nodes to key $k$. As shown in the figure, in most cases the requester will not be the closest node to the key. Consequently, a neighbor of the requester will be asked to forward the lookup request to a node closer to key $k$ in the key space. Such a procedure will be repeated several times (typically, a logarithmic number of times) until reaching the node responsible for the zone in which key $k$ lies.

While excellent for achieving scalability, any DHT that operates with multi-hop routing will be subject to abuse by malicious users intentionally harming the correct operation of the DHT. For example, a malicious user may simply drop or modify all messages passing through it to disrupt the lookup service. Although it can be argued that in traditional cluster systems and small Grids local administrators usually know their users, large-scale systems expose administrators to a large number of unknown users with a great variety of usage patterns. This makes the detection of malicious behavior an extremely complex task and spurs the development of a secure lookup service for large-scale systems.

Several secure routing techniques have been devised to protect DHTs against dishonest users [56]. However, most of these techniques rely on the increase of routing redundancy to improve the probability of routing successfully to the key. The more redundancy there is, the higher the communication cost will be. For very large systems, such communication overhead is non-negligible and may represent a real scalability barrier to implementing a secure lookup service.

1.1. Our vision and contributions

Our key idea is to take advantage of trust as a criterion in neighbor selection and avoid, as much as possible, the use of redundancy. To this aim, we develop a local trust system where each user infers on its own how reliable a neighbor is. Concretely, each node uses the knowledge about the success or failure of its own lookups to determine the odds of a neighbor cooperating in routing lookups. Since there is no information sharing among users, our algorithm scales at zero cost and quickly adapts to the users that continuously come and go. To make informed decisions, our solution assumes the use of iterative routing. In this scheme, the requester does not rely on any intermediate hop to deliver the message to the next hop. On the contrary, every user along the routing path is asked for the next hop and it is the requester himself who forwards the message to the next user. The requester can then check whether the next hop in a route is correct. Although many DHTs make use of recursive routing, all of them can work over an iterative scheme to benefit from our technique—as in other research works [57,27,22,4].

The concrete realization of this vision 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 non-deterministic DHTs. Sophia strictly benefits from direct observations to qualitatively improve routing paths. Unlike redundant routing, Sophia strengthens routing against malicious users at no extra cost. Compared with routing redundancy techniques, it achieves a routing reliability from 1.5 up to 5 times higher. Given a fixed fraction of malicious users, this means that Sophia can tolerate the same proportion of attackers with less parallel paths. In addition, it can successfully tolerate small to moderate fractions of whitewashers, i.e., users who erase their past history by entering as a new node, so that they can continue attacking the network with no fear of punishment.

In summary, the original contributions of this work are:

- Design of Sophia, a novel routing security technique for non-deterministic DHTs combining iterative routing with local trust. Our technique benefits from the flexibility of those DHTs to autonomously select the most reliable forwarders to form the routing table. In addition to high scalability, Sophia is effective even in very dynamic environments, which makes it “ideal” for desktop grids and P2P Grids.
2. Related work

The synergy between Grid Computing and P2P technologies is nowadays a remarkable research topic [33]. A plethora of research works in data management, resource discovery, job scheduling etc., have adopted DHTs as an alternative to centralized or hierarchial service management approaches [53,42,55,24]. Grid middleware such as Organic Grid [7] and Nara Brokering [34] rely on P2P techniques to transparently provide self-organization and scalability. Albeit with important differences [15], recent research efforts pursue successfully introducing Cloud technologies in the classic Grid paradigm [39], where DHTs leverage efficient service decentralization [30].

However, the DHTs present numerous security vulnerabilities that hinder their adoption in large-scale applications including Grids and P2P networks [56]. Specifically, routing vulnerabilities have been a subject of intensive research [6,45,44,21,27,16,41], but yet there is no a universal solution to this problem.

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.

Castro et al. [6] 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 [44], but guaranteeing d independent paths between every two nodes in the overlay. Finally, Harvesf and Blough [21] tried to create d disjoint paths by equi-spacing 2^d-1 replicas on the Chord ring (Equally-Spaced Replicas, ESR). Querying the 2^d-1 replicas in parallel, they were able to increase routing robustness. The main problem of using independent paths is the poor asymptotic guarantees it can provide on the success rate without overloading any node (as the value of d increases).

Also, we can find the following works that attempt to improve the quality of delivery paths:

In [45], the authors overviewed the potential benefits and shortcomings of applying trust to routing, hypothesizing an abstract reputation system with both false positives and false negatives. Although the results were promising, this work did not include any concrete algorithm to sustain their arguments. In [27], the authors proposed SPROUT, a routing protocol that increased the probability of routing successfully by leveraging social links. SPROUT defined a reputation model based upon social distance to avoid routing messages through dishonest peers. However, it relied on social links that may not be always available.

Another related algorithm is the Feedback Forward Protocol (FFP) [16]. In this protocol, peers accumulate evidence to predict the routing behavior of their neighbors. Based on this evidence, each node can find out whether a given neighbor did a good routing job in the past, and select it as a next hop. 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.

The closest related work we are aware of is our own research in [41]. As a case study, we proposed in this work a local trust system to improve routing. Although this work is in line with the present manuscript, there exist several and important differences between both. First, the algorithm in [41] does not maintain a persistent history of past transactions to make routing decisions. That is, past transactions are recorded until the corresponding neighbor goes off-line or changes his identity. Second, the authors did not consider the advantages that iterative routing can bring to the convergence of any local trust system.

Compared with prior work, the aim of Sophia is to improve routing paths through the use of historical information. This information is not shared, remaining local to each user. As a result, Sophia does not add any extra overhead to the routing protocol due to the exchange of trust information, scaling at no cost. Also local trust immunizes Sophia against false rumor spreading, a problem of global trust systems.

To summarize this section, we facilitate a comparative frame of Sophia with several state-of-the-art routing security techniques in Table 1. These techniques are compared by the following criteria: (i) Whether a security technique involves some information or knowledge about nodes in its execution process. (ii) Type of lookup parallelism specified. (iii) Routing scheme supported by a certain technique (iterative/recursive). (iv) Identities employed by nodes. (v) Costs introduced by every approach. (vi) Applicability of a technique to multiple DHTs.

3. Preliminaries

In this section, we provide the necessary background and definitions used throughout the rest of the present manuscript.

---

Table 1: Comparative frame of Sophia with other well-known routing security techniques.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Informed decision</th>
<th>Routing redundancy</th>
<th>Routing scheme</th>
<th>Node identities</th>
<th>Costs</th>
<th>Genericity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFP [16]</td>
<td>Yes (transitive trust)</td>
<td>−</td>
<td>R</td>
<td>Social network I</td>
<td>N, C, S</td>
<td>G</td>
</tr>
<tr>
<td>SPROUT [27]</td>
<td>Yes (social information)</td>
<td>−</td>
<td>I</td>
<td>CA certificates</td>
<td>C (crypto), S</td>
<td>GF (deterministic DHTs)</td>
</tr>
<tr>
<td>Cyclone [44]</td>
<td>No</td>
<td>Independent paths</td>
<td>R, I</td>
<td>−</td>
<td>N, D</td>
<td>G</td>
</tr>
<tr>
<td>Myrmic [57]</td>
<td>No</td>
<td>−</td>
<td>CA certificates</td>
<td>C</td>
<td>N, D</td>
<td>G</td>
</tr>
<tr>
<td>ESR [21]</td>
<td>No</td>
<td>Disjoint paths</td>
<td>R, I</td>
<td>−</td>
<td>N, D</td>
<td>G</td>
</tr>
<tr>
<td>Sophia [17]</td>
<td>Yes (local trust)</td>
<td>Isolated paths</td>
<td>I</td>
<td>Self-certificates</td>
<td>C, S</td>
<td>GF (non-deterministic DHTs)</td>
</tr>
</tbody>
</table>

Routing scheme: (R) Recursive, (I) Iterative.

Costs: (N) Network overhead, (C) Computational overhead, (D) Data replication overhead, (S) Storage overhead.

Genericity: (G) Generic, (L) Limited to a single DHT, (GF) Generic for a family of DHTs.

---

1. Two paths are said to be independent if they share no common node other than the source and the destination.
Kademlia [31] is the most widely deployed Distributed Hash Table in the Internet today (e.g., Bittorrent Mainline DHT, eMule’s Kad, OverNet). Kademlia is a structured overlay devised by Maymounkov and Mazières, 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 identifier and maintains a routing table of \( m \) \( k \)-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 \( \alpha \) (\( \alpha \leq k \)) 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 \( \alpha \) nodes are chosen. The procedure is repeated until the node responsible for key \( K \) is found. In this regard, the lookup procedure is discussed in more detail later in Section 5 (see Fig. 2).

To be tolerant to failures and departures, each Kademlia peer learns about new nodes either by asking nodes he already knows while searching or by receiving messages from nodes. New nodes are inserted into its routing table if the corresponding \( k \)-bucket is not full, or the first contact in the \( k \)-bucket does not respond to a PING RPC message. In that case, the contact is evicted from the \( k \)-bucket and the new node is inserted at the tail. Otherwise, the new contact is discarded and the contact at the head of the \( k \)-bucket is moved to the tail.

3.1. Kademlia DHT

A node tests the liveness of its contacts opportunistically while searching, or (if necessary) periodically with PING RPC messages to check if they are still alive. The testing period for a contact is typically 2 h.

We use Kademlia to validate our findings: Why? A wide variety of DHTs have been proposed in the past decade [51,40,63]. However, very few DHTs have been actually implemented and deployed at large scale. One of the most remarkable cases is the Kademlia DHT [31]. Due to the robustness and simplicity of Kademlia’s design, nowadays many implementations based on this DHT provide distributed key/value management in applications such as file sharing (eMule’s Kad) and content distribution (Bittorrent’s Mainline DHT and Azureus DHT). The research community explored these Kademlia-based DHTs in depth since they represent a unique opportunity of analyzing real characteristics of massive systems [48,50,12]. Furthermore, many studies regarding security attacks and countermeasures have been deployed over these DHT implementations [58,49]. To the best of our knowledge, no other DHT has been studied, in both theoretical and real settings, as much as Kademlia.

Our contributions in this manuscript are not restricted to Kademlia DHT—see Section 4.8. However, we believe that Kademlia is a representative substrate to improve key-based routing in DHTs.

3.2. Threat model and assumptions

In this work, 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 \( N_f \) malicious nodes, where \( f \) represents the fraction of attackers. Further, we assume that the attackers are uniformly distributed along the \( m \)-bit key 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 [11]. We believe that this assumption is reasonable in many environments. For instance, in Grid platforms, the administrators of the different domains can bind digital certificates to user identities. In addition, many research works focused on securing DHTs against Sybil Attacks [26,9,56]. Thus, we assume the adoption of a Sybil-resistant mechanism in order to concentrate our efforts on protecting routing.

The threat model considered in this work is the following:

- **Message corruption and drop** [6]: Attackers can disrupt the lookup service by spreading false information about their...
closeness to a particular key, by intercepting routing requests or falsely responding with “no such key” messages.

- **Routing table poisoning:** Another important issue refers to routing table poisoning, intended to launch Eclipse Attacks [47]. Basically, attackers perpetrated an Eclipse attack try to corrupt the routing tables of honest nodes by filling them with other malicious nodes. This threat model is considered and explained next as a collusion attack. However, Eclipse attackers also benefit from a loosely controlled, or even from an open membership policy, to augment the impact of the attack. As we previously assumed, in our threat model an attacker cannot arbitrarily place bogus nodes in the overlay’s identifier space to perform this kind of attack.

For this work, malicious participants exploit the previous attacks as follows:

**Uncoordinated attacks.** Uncoordinated attacks are launched by attackers independently of each other. 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 node. This means that attackers respond 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 malicious nodes to punish honest users. In this case, the attackers create a fictitious route for each lookup that terminates at the closest attacker to the key in an attempt to deceive the requester into thinking that the targeted key-value pair does not exist. To create the artificial routes, the adversary bootsraps all the malicious instances into a parallel Kademlia overlay. In this way, every time a request touches a malicious user it gets trapped into the malicious overlay. For malicious users, the regular routing table is used in order to minimize the routing effort of the attacking coalition. A side effect of this attack is that the navigation of any user through the malicious overlay increases the odds by more than $f$ to introduce a malicious user into a $k$-bucket as Kademlia nodes learn about new nodes while searching.

**Highly available attackers.** Kademlia biases routing tables towards long-lived contacts by placing a node in a $k$-bucket only if the bucket is not full or an existing contact is off-line, which can be exploited by malicious users to disrupt routing. If attackers are more available than regular nodes, they will be inserted into more $k$-buckets than the average and preferred as forwarders for their high availability, inflicting more damage on the network.

**Whitewashing.** Our threat model considers the problem of cheap pseudonyms [13]. When identities are easy to obtain, malicious users can erase the historical records of their past bad actions by re-joining the overlay under a new identity. To model this attack, we assume that whitewashers automatically renew their node IDs after performing $\theta > 0$ lookups, so that they can be periodically chosen as next hops after each identity renewal. We assume also that an identity renewal entails a change in the physical connection information—i.e. IP address. This prevents the possibility of tracking an attacker by its network address.

With the exception of whitewashers, we assume that users have non-spoofable semi-permanent IDs to preserve the integrity of the transactions. For transaction we mean the process of finding a reliable routing path to the key. In particular, we use the hash over a public key to generate user IDs, since with public keys it is also possible to sign the messages exchanged by nodes [4]. Public Key Infrastructure (PKI) certificates are the core components of important de facto standards such as the Grid Security Infrastructure (GSI) [59], which facilitates the integration of Sophia with grid middleware such as Globus Toolkit [54]. To prevent users from spoofing the identity of other users, we use a challenge-response protocol similar to that in [36].

### 3.3. State-of-the-art countermeasures

In the particular case of Kademlia, key-based routing is vulnerable in two ways. The first is that in each step, the set of candidates returned from a previous RPC are merged into a list, ordered by increasing XOR distance to the destination key. This merging process facilitates that a single malicious user can make a lookup fail by returning the hypothetical or closest candidates to the key. In this sense, Kademlia behaves as a conventional DHT, in which there is exactly one routing path from the source to the destination. Even assuming a uniform distribution of the malicious users along the identifier space, this procedure leads to an asymptotically small probability of routing successfully.

More formally, let $\xi_k$ denote the event that a routing operation that requires $h$ hops fails at the $i$th step. It is not hard to see that the probability of failure can be approximated by $Pr(\text{Failure}) = 1 - \Pr \left( \bigcap_{i=0}^{h-1} \xi_i \right) = 1 - (1 - f)^h$, which is asymptotically $1$ when $h \to \infty$. This result gave birth to redundant routing, which tried to improve this bound through the use of redundancy, in the sense of routing over multiple independent paths. Given the high complexity of identifying honest users, the logic behind this technique was to trade complexity for cost. Works like Cyclone [44,21] made use of redundant routing to increase the probability of query success. Using $d$ independent paths, they could reduce the probability of query failure by an exponential decay factor:

$$
Pr(\text{Failure}) \leq (1 - (1 - f)^h)^d,
$$

which is approximately $\exp(-dN \log(1-f)/lnh)$. This means that to maintain $Pr(\text{Failure})$ constant, $d$ must be polynomial in $N$. That is, if $\exp(-dN \log(1-f)/lnh) \leq \epsilon$, $d$ must be at least $\ln\left(\frac{N}{\epsilon}\right) \frac{\log\frac{1}{\log^d(N)}}{\log\frac{1}{\log\frac{1}{\log^d(N)}}}$. Consequently, redundant routing might lead to overloading the weak peers in the effort of maintaining $Pr(\text{Failure}) \leq \epsilon$. This is the reason why we introduce historical information and avoid merging forwarding candidates to enhance the routing process.

The second vulnerability is that 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 w.h.p. after adversarial deletions. To address this problem, the authors of S/Kademlia [4] added a sibling list of length $\eta$ per node to ensure that the Kademlia routing protocol reaches at least $\eta$ 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 [4], a value of $\eta \geq 5$ is enough to make sure that routing operations converge to at least $\eta$ siblings w.h.p. and, hence, allow Kademlia to store data in a safe way.

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

---

Note that by setting $\varepsilon = N^{-\delta}$, this bound is a high probability bound, as $Pr(\text{Success}) = 1 - Pr(\text{Failure})$. 

---

4. Sophia: system design

Now, we present Sophia: Securing Overlays with a Personal History Approach. This section introduces and describes the main features of our routing security technique, its general functioning and the decisions involved in its design.

4.1. Overview and design decisions

From a general perspective, Sophia is a routing security technique which employs trust information gathered at each node to improve the routing security of the entire system. This local information relates a node's neighbors to the quality of routing services it received from them in the past. To wit, each node in the system evaluates whether a certain neighbor correctly responds to routing requests or not. By accumulating such information, nodes are capable of making a more effective neighbor selection and, thus, increase the probability of routing successfully.

Albeit conceptually clear, there are several design decisions involved in the functioning of Sophia. Next, we describe and discuss the main decisions that characterize our technique:

- **Local trust.** Many trust and reputation systems introduce the exchange of trust information among nodes to accelerate the convergence of trust values [23]. However, quantity and quality of trust information are diametrically opposed aspects [29]. In other words, when trust values are distributed in a system, the more information is gathered the less credible is each piece of information. The main reason behind this fact is that malicious nodes may introduce false feedback in the system. Moreover, the distributed management of trust information among nodes introduces a considerable complexity and network overhead.

- **Iterative routing.** Sophia works over an iterative routing scheme. That is, with iterative routing, the requestor asks each intermediate node along the routing path about the next hop. Hence, the requestor can accrue personal experience about the routing behavior of intermediate forwarders and rate them according to the result of the routing operation.

One can argue that iterative routing imposes a higher overhead than recursive routing in the lookup process. Although this fact is true, there are reasons in favor of this design decision: (i) Iterative routing enables a node to autonomously evaluate the behavior of its neighbors. With recursive routing, the requester node cannot discern locally which node along a lookup has been responsible for a routing failure. (ii) Inspecting every forwarder along a lookup path accelerates the acquisition of trust information and the convergence of trust values.

- **Persistent storage of trust information.** In Sophia, each user synthesizes direct observations on the routing behavior of his neighbors in the form of trust ratings. 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. Employing a robust identity scheme [2,3], we can benefit from persistently identifying (honest) nodes and store their past behavior to improve future routing operations.

- **Isolation of parallel lookup paths.** There are several approaches to implementing routing redundancy (i.e. sending a lookup through several neighbors in parallel). In Sophia, we defined a lookup as a set of parallel and isolated paths to a certain key. Each parallel path corresponds to a single routing transaction in our system. A node evaluates the behavior of nodes involved in each routing path irrespective of the result exhibited by the other paths belonging to the same lookup. This design decision yields two important benefits: (i) A malicious node cannot disrupt more than one path at each lookup step. (ii) Considering each path is a single routing transaction, we benefit from the degree of routing redundancy to accelerate the convergence of trust information.

4.2. Sophia in a nutshell

In this section we overview the architecture and functioning of Sophia. Fig. 3 depicts an example of how Sophia reacts in the face of a routing failure. The example shows the interaction between the main components of Sophia. We overview each component in order of use. Upon the detection of the failure, the first step is for the requestor to apply a trust policy to evaluate the routing behavior of each intermediate hop. After the evaluation, the second step is the update of the trust ratings associated with the intermediate hops using a trust calculation algorithm. Trust information about neighbors is stored into the Personal History (PH).
of the source node. The next time the requester starts a lookup, the requester will input these trust ratings into a neighbor selection algorithm to decide the next hop in the route to the destination key. Let us illustrate with a simple example how Sophia updates trust values of nodes (see Table 2). In Fig. 3 we observe that a node launches a lookup, initially routed through node N1. Node N1, which is a priori honest, returns the node it knows closest to the requested key. This node is N6. However, when the source node routes through N6, it does not respond. The source node faces a failed routing transaction and Sophia reacts to this event. First, Sophia assigns a value to each forwarder representing the behavior exhibited during the transaction (e.g., 0 or 1 depending on whether a node forwarded a message correctly or not). Subsequently, Sophia updates this information in the corresponding entries of the PH. Finally, Sophia executes the trust algorithm in order to update the routing behavior value of each node. This update process of trust information can be observed in Table 2.

In what follows we describe each component of Sophia’s architecture.

4.3. Personal History

The Personal History (PH) is the main data structure of our architecture. In it, each Sophia node stores the trust ratings for each known neighbor. The PH presents the same structure as a standard routing table. For each entry of the PH, we have a list of $h$ candidates. The value of $h$ is taken so that there exists at least one honest candidate for each entry w.h.p. By a reasoning similar to that developed for the sibling zone in Section 3.3, it can be shown that $h = \Omega(\log N)$ is sufficient to guarantee this property. From now on, we will refer to each entry of the PH by the term $h$-bucket.

4.4. Local trust policies

The provisioning of a criterion about how to evaluate the nodes involved in a transaction is given by the trust policy. In our system, a transaction corresponds to the process of successfully establishing a routing path between the requester and the node responsible for the key. Hence, it is natural that the result of a transaction is formalized as a binary value: a value of 1 for a successful transaction and 0 for a failure.

In Sophia, a trust policy is an algorithm which determines the input given to the trust calculation algorithm. The decision is based on the result of a transaction and it is independently applied to all forwarding nodes along the routing path. In the case of issuing a lookup through $\alpha$ parallel paths, Sophia applies the trust policy in each of the $\alpha$ paths, independently of each other. Put another way, the trust policy evaluates forwarding nodes depending upon whether their respective paths were successful or not.

To evaluate Sophia, we implemented two trust policies:

Pessimistic policy. This policy gives a positive trust score of 1 to all nodes along the path if a transaction has been successful. On the contrary, it punishes all nodes along the path giving a trust score of 0. Note that this policy could erroneously punish cooperating nodes (false positives). However, since a node evaluates its neighbors only with its own transactions and without any other external information, this policy is more conservative in misclassifying malicious nodes as good ones.

Oracle trust. An 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. However, it is an accurate way to compare any local trust policy with the theoretically best one.

4.5. Trust calculation

In Sophia, the trust calculation algorithm computes a trust rating for each node found in a routing path. To ease description, we will dissect our algorithm following the decomposition proposed in [23]:

Source of information. As input for the trust system, we decided to use only first-hand observations on the success or failure of lookups. This approach is in line with other representative trust systems such as [32]. Based on this input, the information that the calculation algorithm receives are binary values given by the trust policy.

Information type. We decided to incorporate in trust ratings both good and bad routing actions. This two-fold consideration provides a broader notion of user behavior and is key to the response to unexpected behavioral changes (e.g., traitors) [23].

Temporal aspects. In any trust model, it is critical to weigh the importance of recent observations in relation to historical transactions in order to react rapidly to changes. Sophia updates trust rates aggressively using a short-term history to quickly punish defectors.

Trust metric. To represent the degree of trust a node places in a neighbor, we use a real value $[0, 1]$ similar to other reputation systems such as [28].

Calculation. The trust calculation is performed locally at every node. Thus, every node has its own view of the network and for each forwarder, there exist as many local views as nodes have seen this forwarder.

For a node $n_i$, the associated trust rating, called routing behavior value (RB), is calculated as follows:

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

where $\lambda \in [0, 1]$ determines the importance given to the most recent $d$ transactions from the total $t$. Fig. 4 plots how the parametrization of $d$ and $\lambda$ impacts on the trust calculation. To quantify the impact, we have designed two typical scenarios where any trust calculation algorithm is expected to perform well. In the first scenario, a malicious user behaves well and badly alternatively, hoping that he can remain undetected while causing damage. In the second scenario, the malicious user behaves well until the 50th lookup. From now on, he misroutes any lookup from that requester. As can be shown in the figure, for $d = 3$ and $\lambda = 0.7$, our algorithm exhibits a nice tradeoff between fast adaptation to behavioral changes ($\lambda = 0.7$) and stability in routing prediction ($d = 3$). We have used these values for fine tuning Sophia.

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example of updating the PH after a transaction as depicted in Fig. 3. In this case, N6 is responsible for the failure of a lookup path, whereas N1 forwarded the message correctly. This situation is reflected in the routing behavior (RB) values of both nodes at the source node’s PH. Note that in this example, the trust calculation algorithm is just a simple average of the historical routing operations related to a node.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node Historical trans.</th>
<th>RB</th>
<th>Node Historical trans.</th>
<th>RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>1.0, 0.0, 1.1, 1.1, 1.1, 0, 1</td>
<td>0.7</td>
<td>N6</td>
</tr>
</tbody>
</table>

| N1 | 1.1, 1.1, 1.1, 1.1, 0, 1 | 0.9 | N1 | 1.1, 1.1, 1.1, 1.1, 0, 1 | 0.909 |
| N6 | 1.0, 0.0, 1.1, 1.1, 1.1, 1 | 0.7 | N6 | 1.0, 0.0, 1.1, 1.1, 1.1, 0 | 0.636 |
4.6. Neighbor selection policies

At this point, we have developed means to calculate and store the trust values of neighbors. However, we also need a criteria to update routing tables based on the trust ratings in the PH, a role adopted by the neighbor selection policy. To measure the potential of our solution, we implemented the following policies:

- **Select-best.** In this selection policy, we choose the $k$ available nodes with the highest trust ratings from the $h$-bucket.

- **Probabilistic selection.** Cooperating and highly available nodes might be overloaded just because they successfully resolve requests during long periods of time. To avoid such an effect, we propose a neighbor selection policy which probabilistically selects the best on-line forwarders. In this policy, each $h$-bucket contains the nodes sorted in decreasing order of their trust values. That is, the $h$ trust ratings $RB(n_1), RB(n_2), \ldots, RB(n_h)$ of an $h$-bucket are placed in descending order forming a set $(RB(n_i))_{i=1:h}$ of ordered values where $RB_1 \leq RB_2 \leq \cdots \leq RB_h$. Thus, node $n_1$ is chosen with probability $RB(n_1)$, $n_2$ is chosen with probability $RB(n_2)$, etc. This test is performed in the $h$-bucket until collecting the necessary $k$ elements to update the corresponding $k$-bucket.

Note that this policy biases towards the neighbors with the highest trust ratings while giving a chance to the remaining candidates in the $h$-bucket.

4.7. Dealing with dynamism

An important handicap of largely-deployed distributed systems is dynamic node participation, namely churn [38]. In large-scale systems, nodes tend to continuously leave and rejoin the system, participating for an unpredictable amount of time. In our view, it is crucial to make Sophia robust to churn to achieve a satisfactory performance in dynamic scenarios. The current section describes the changes to adapt Sophia to scenarios with high churn rates.

In order to tolerate churn, Sophia keeps track of node availabilities in the Personal History (PH). For each entry, Sophia maintains whether it is currently on-line or off-line. To update availability information without being intrusive to the underlying DHT, Sophia uses two sources of information present in any DHT:

1. **Availability updates via keep-ales.** Whenever a Kademlia node receives a message, the least-recently used node in the bucket is “pinged” to determine whether it is alive. If the node does not respond to the PING, RPC is marked as unavailable in the PH.

2. **Availability updates via routing success/failure.** Another source of information about node availabilities is the result of a routing transaction. In this case, we distinguish two possible situations. First, if the construction of a lookup path has been successful, we assume that all nodes found 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 mark the last hop as unavailable and apply the appropriate trust policy to the entire path (see Fig. 5).

4.8. Potential benefits of Sophia

The architecture described in this section has several important implications. Next, we draw what we consider are the main
benefits that yield the architecture of Sophia:

**Design simplicity.** The design of Sophia is relatively simple to implement in a real DHT. Furthermore, although the complexity of the implementation depends on the algorithms defined to rate and select nodes, the local vision of Sophia significantly limits its complexity. In addition, Sophia can be seen as an external component intended to improve the quality of routing paths entailling a minimal impact on the original DHT substrate.

**Avoiding extra communication overhead.** The combination of local trust over iterative routing is itself sufficient to improve the quality of routing. As we previously mentioned, there is no need of introducing extra communication overhead or distributed trust management.

**Easy to deploy.** Sophia is a local routing security technique that works independently in each node without affecting the regular routing protocol. This yields that an overlay could be formed by both regular nodes and Sophia nodes. To wit, since the selection of the next hop is individually done by each host, nothing prevents a regular user from communicating with any computer armed with Sophia and vice versa. Therefore, our technique can be gradually introduced in a working large-scale system without compromising its correct functioning.

**Genericity.** Sophia is a generic security technique applicable to many DHTs. Concretely, Sophia can be effectively implemented in DHTs with non-deterministic routing geometries [20]. This means that Sophia is suitable for those DHTs which are flexible enough to consider a set of nodes belonging to a section of the identifier space as candidates to be inserted in a routing table entry. Sophia will work for selecting the most reliable candidates, among the possible ones, to fill up a node’s routing table. Thus, DHTs such as Pastry [40], Kademlia [31], Tapestry [63], P-Grid [1] and randomized-Chord [20,62] are candidates for benefiting from our technique.

5. Simulation framework

Our simulation scenario consists of a Kademlia tree formed by \( N \) nodes, identified by \( m \)-bit keys, whose lifetimes are described by input traces. The format of these input traces is the .avt (availability trace) file format which has been extensively adopted in the literature [48,19]. Our simulator has been developed in Erlang\(^3\), a programming language conceived for building large-scale distributed systems. Further, the simulator has been carefully implemented and extensively tested during the progress of this research. For each configuration in Table 3, 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 periodically injects (every \( \delta \) seconds) a lookup for a key chosen uniformly at random. With the exception of the whitewashing scenario, nodes form a fully populated Kademlia tree, i.e., the key space is completely covered by nodes even though a fraction of them may be off-line due to network dynamism.

Additionally, a lookup successfully terminates if any lookup path reaches the requested key **sibling zone**. 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, though the responsible node does not serve the associated data object, which is a storage issue.

In the case a host becomes unavailable for a certain period, its activity is completely interrupted until returning to be available again. Further, the number of lookups issued by each user during the simulation is divided into transitory (\( T_t \)) and stationary (\( S_t \)) operations. The transitory lookups are used to bootstrap the network and are not considered in the evaluation. Also, nodes inject extra traffic into the network as a result of the PING RPCs used for the availability tests and the periodic maintenance lookups (stabilization). These algorithms are configured with their specific timeouts, \( P_t \) and \( S_t \), respectively.

5.1. Routing redundancy counterparts

**Wide paths: Kademia redundant routing.** The concept of wide paths has been proven as a really effective routing redundancy technique [22]. The main strength behind the concept of wide paths is that, in general, appointing \( \alpha \) nodes to form a unique path is more effective than having each node forwarding to a separate path. Intuitively, a wide path resists \( \alpha - 1 \) failures at each routing hop. In [22], the authors argue that this scheme is simple and it gives a much greater fault tolerance than the multiple path technique, which provides asymptotically bounded success guarantees as shown in Section 3.3. Kademia standard routing is as an example of the wide paths technique.

**Disjoint paths: equally-spaced replication** [21]. Compared with independent paths, disjoint paths share no common node other than the source. This fortifies DHTs not only against routing attacks but also against storage attacks. This means that, once \( d \) messages are sent through \( d \) disjoint paths, 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 adds extra object replication costs since it provides a defense against storage attacks.

**Equally-Spaced Replication (ESR)** [21] is a representative example of this technique. ESR exploits key replication to produce disjoint routes and, consequently, increase the probability of routing successfully. Concretely, this technique creates \( \omega \) disjoint routes by equi-spacing \( 2^{\omega - 1} \) replicas along the key space. The main strength of this approach lies in the concept of disjoint paths: lookup paths simultaneously start from different routing table entries targeting equi-spaced replica locations. This fact ensures that any pair of lookup paths share no other common node than the originator node, increasing routing resilience against dishonest forwarders.

5.2. Churn models and datasets

To simulate a dynamic network, we need to model how nodes join and leave the system, for we use .avt traces. Clearly, the availability pattern followed by nodes will considerably differ

---


Table 3

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes (( N ))</td>
<td>8192 nodes</td>
</tr>
<tr>
<td>ID length (( m ))</td>
<td>13 bits, 20 bits</td>
</tr>
<tr>
<td>k-bucket size (( k ))</td>
<td>3</td>
</tr>
<tr>
<td>Redundant paths (( \alpha ))</td>
<td>[1 → 3]</td>
</tr>
<tr>
<td>Sibling zone (( S_z ))</td>
<td>0 bits, 11 bits</td>
</tr>
<tr>
<td>Simulation time</td>
<td>30,000 s</td>
</tr>
<tr>
<td>Transitory lookups (( T_t ))</td>
<td>50</td>
</tr>
<tr>
<td>Lookup period (( \delta ))</td>
<td>30 s</td>
</tr>
<tr>
<td>Stabilization execution period (( S_t ))</td>
<td>5 min.</td>
</tr>
<tr>
<td>Ping/pong timeout (( P_t ))</td>
<td>3 s</td>
</tr>
<tr>
<td>Fraction of malicious nodes (( f ))</td>
<td>[0.0 → 0.6]</td>
</tr>
<tr>
<td>Transactions until whitewash (( \phi ))</td>
<td>50</td>
</tr>
<tr>
<td>k-Bucket size (( b ))</td>
<td>15</td>
</tr>
<tr>
<td>Short term history transactions (( d ))</td>
<td>3</td>
</tr>
<tr>
<td>Short term history weight (( \lambda ))</td>
<td>0.7</td>
</tr>
</tbody>
</table>
depending on the availability trace. 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 mean online 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. [19] 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 min, providing a reasonably good lifetime resolution.

- **Kad Trace**: The second trace is from eMule's KAD overlay, obtained by Steiner et al. in [48], and describes the behavior of 400,000 KAD peers, monitored during 6 months. Since our simulator cannot process such a number of concurrent nodes, we used a filtered dataset from this trace containing the 10,000 peers with longest membership [35].

Synthetic availability traces. For synthetically generated traces, we adopted the Heterogeneous Yao Model [61]. The mean node on-line session was set to $\bar{On} = 1$ h and the disconnection period was also configured to $\bar{Off} = 1$ h. Besides, it has been observed that the distribution of user lifetimes in real P2P systems is often heavy-tailed (i.e., Pareto), in that 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 [25]. For this reason, this trace was generated to measure the impact that very heavy-tailed lifetime durations have on Sophia. The synthetic traces were obtained using a novel churn generator presented in [14] (see Fig. 6).

6. Validation

To shed some light on the performance of Sophia in dynamic and hostile scenarios, we conducted an extensive set of simulations. Specifically, we evaluated three main properties: First, we assessed the Lookup Success Ratio (LSR), which measures the routing reliability provided by each routing technique. Second, we measured the average path length (APL) rendered by each routing technique. Finally, we examined whether transactions were evenly distributed among nodes (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 a property in two ways:

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

  \[
  LSR = \frac{S_{\text{lookups}}}{T_{\text{lookups}}}
  \]

  where $S_{\text{lookups}}$ is the amount of successful lookups divided by the total number of lookups $T_{\text{lookups}}$.

- **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:

  \[
  LSR \text{ Gain Ratio} = \frac{LSR_{S_1}}{LSR_{S_2}}.
  \]

Finally, the proposed simulation framework was designed to answer the following questions:

- In a stable network, How many lookups does Sophia need to provide a more resilient routing service than routing redundancy? (Section 6.1).
- Can Sophia dynamically recover from unexpected attacks? (Section 6.2).
- Does Sophia provide a greater routing reliability than redundancy techniques in dynamic and hostile scenarios? (Section 6.3).
- Regarding the adaptation of Sophia to dynamism: Does it perform as expected in the presence/absence of attackers? (Section 6.3).
- How do highly available attackers affect the routing performance? (Section 6.4).
- What is the trade-off that Sophia poses between reliability and load-balancing? (Section 6.5).
- Finally, What happens to Sophia’s performance if attackers are not persistently identified by the system? (Section 6.6).

6.1. Trust convergence

The performance of Sophia is related to the amount of transactions exchanged with neighbors. Figs. 7 and 9 show the degree of routing robustness under attack (measured as LSR) provided by Sophia and Kademia and how it evolves over 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 Figs. 7 and 9, this reduced number of transitory lookups is enough for Sophia to obtain a greater LSR in the first interval of measurement compared with Kademia, for the same degree of redundancy ($\alpha$).

Results in Fig. 7 correspond 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
Fig. 7. Lookup success ratio evolution of Kademlia and Sophia when 20% of nodes perform a collusion attack for different values of $S_z$ and $\alpha$.

(a) $S_z = 0$, 20% collusion attack. (b) $S_z = 4$, 20% collusion attack.

Fig. 8. Sophia’s malicious nodes avg. in routing table buckets, depending on the amount of lookups sent per node (30% individual attackers, $\alpha = 2$).

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. Not surprisingly, as more observations are available, the ratings about neighbors become progressively more reliable. This fact is clear in Fig. 7; for instance, when $S_z = 0$, Sophia outperforms Kademlia by approximately 30% for the same $\alpha$.

Despite Kademlia slightly enhancing the LSR when increasing $\alpha$, it does not learn from neighbors, so the LSR remains lower than Sophia. Note that, to meet a certain threshold on routing robustness, Sophia needs fewer parallel paths than Kademlia. Thus, reducing $\alpha$ from 3 to 2 represents a network traffic saving of 33%.

It is worth mentioning existing differences in the LSR depending on the $S_z$ configurations (Fig. 7). Kademlia improves noticeably when introducing a larger $S_z$. The reason is that a larger $S_z$ implies a shorter APL, and this fact increases the lookup success probability. However, Sophia takes a greater advantage of having a larger $S_z$ than Kademlia. 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. 8, where we can observe the evolution of a Sophia node routing table, measured as the average fraction of malicious nodes per $k$-bucket.

Fig. 9 corresponds to a hostile scenario where 30% of nodes perform an individual attack.

Results depicted in Fig. 9 follow a similar fashion to 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 during the experiment, compared with Sophia. 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 the Pessimistic Trust Policy versus Oracle Trust (Fig. 10) in a stable scenario. 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 at identifying 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 in identifying the nodes placed in the lowest $k$-buckets of the routing table.

Fig. 9. Lookup success ratio of Kademlia and Sophia when 30% of nodes perform an individual attack for different values of $S_z$ and $\alpha$.

(a) $S_z = 0$, 30% individual attackers. (b) $S_z = 4$, 30% individual attackers.

Fig. 10. Lookup success ratio comparison of Sophia using Pessimistic Trust Policy versus Oracle Trust, when 20% of nodes perform a collusion attack and $S_z = 4$. 
6.2. Dynamic recovery

In this part, we examine how Kademlia and Sophia tolerate unexpected behavioral changes of nodes. Fig. 11 shows the effects of a sudden individual attack where 30% of nodes start dropping all requests at lookup 150. Additionally, both systems are configured with $S_z = 0$ and different degrees of redundancy.

As we can observe in Fig. 11, the attack starts at the second interval of measurement. At this point, we can see how both systems present a similar LSR downturn. 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 of dynamically recovering from unexpected attacks, restoring the lost routing reliability during the attack.

6.3. Handling churn and adversarial nodes

The battery of simulations presented in Figs. 12 and 13 depicts the routing reliability provided by the evaluated systems under individual and collusion attacks. These attacks are performed in all the proposed churn scenarios described in Section 5.2. As can be observed, all the configurations sufficiently tolerate churn in most cases due to availability tests and the stabilization algorithms used by the DHT itself. However, in cases of high churn rates, the results obtained suggest that one path ($\alpha = 1$) is not enough to provide an acceptable routing reliability. Therefore, a certain degree of routing redundancy is needed for any system to ameliorate the effects of high churn rates. Further, 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 into the network, the behavior of systems changes dramatically. This fact is clear on observing the LSR values depicted in Fig. 12. For instance, regarding the Skype Trace, Kademlia for $\alpha = 2$ reduces its reliability approximately by 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 [22]: although ESR paths do not share common forwarders to avoid faulty ones as much as possible, paths are forwarded by one node at each step. This means that a single malicious node is able to destroy an entire path. In [22], this approach of redundant routing is declared less resilient against malicious nodes than the wide paths technique, adopted by Kademlia. 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. 12) of Sophia versus Kademlia and ESR increases as $f$ attains larger values.
In Fig. 13 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 the evaluated systems. In such a scenario, maintaining an historical tracking of nodes is considerably beneficial: for example, in the 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 especially 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. 14 provides a wide view of the impact of malicious nodes on the successful lookup Average Path Length (APL). As can be appreciated in the graphic matrix, adversarial nodes produce a lower impact on Sophia’s lookup APL than 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 affect Sophia’s lookup APL becomes smaller as their trust values reflect their behavior. These graphics are coherent with results obtained in Figs. 12 and 13.

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 in 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 as 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.

### 6.4. Highly available attackers

In this part of the validation we present the effects of varying the fraction of individual attackers that deliberately remain online during the whole simulation process. As can be inferred from Fig. 15, this threat model is remarkably aggressive, especially 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 keeps 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...
Simulations varying the fraction of highly available attackers under churn. In Fig. 15 one can appreciate 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 to 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 h, where nodes inject lookups every 30 s while on-line. This amount of traffic is much more than sufficient to enable Sophia to distinguish between cooperating and adversarial nodes.

An important aspect which is specially noticeable in this experiment is that Sophia persistently rates the quality of a node's neighbors and uses 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, thus affecting the overall network performance. We wanted to explore how not greedily selecting the best forwarders would affect Sophia's routing reliability. For this reason, we executed simulations configuring Sophia with a non-greedy neighbor selection policy (see Section 4.6): Select Probabilistic (SP). Additionally, we analyze the performance in dynamic scenarios of the Pessimistic Trust Policy against the theoretically best local trust policy, the Oracle.

Fig. 16 shows the LSR and Load Balancing obtained by Sophia configured with several policies, adopting the Kad Trace as availability pattern. Let us discuss 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, 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 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 in the routing table.

In Fig. 15 one can appreciate 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 to 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 h, where nodes inject lookups every 30 s while on-line. This amount of traffic is much more than sufficient to enable Sophia to distinguish between cooperating and adversarial nodes.

An important aspect which is specially noticeable in this experiment is that Sophia persistently rates the quality of a node’s neighbors and uses 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, thus affecting the overall network performance. We wanted to explore how not greedily selecting the best forwarders would affect Sophia’s routing reliability. For this reason, we executed simulations configuring Sophia with a non-greedy neighbor selection policy (see Section 4.6): Select Probabilistic (SP). Additionally, we analyze the performance in dynamic scenarios of the Pessimistic Trust Policy against the theoretically best local trust policy, the Oracle.

6.5. Neighbor selection and trust policies

The main reason for the success of Sophia is that it persistently rates the quality of a node’s neighbors and uses 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, thus affecting the overall network performance. We wanted to explore how not greedily selecting the best forwarders would affect Sophia’s routing reliability. For this reason, we executed simulations configuring Sophia with a non-greedy neighbor selection policy (see Section 4.6): Select Probabilistic (SP). Additionally, we analyze the performance in dynamic scenarios of the Pessimistic Trust Policy against the theoretically best local trust policy, the Oracle.

Fig. 16 shows the LSR and Load Balancing obtained by Sophia configured with several policies, adopting the Kad Trace as availability pattern. Let us discuss 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, 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 go 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 a routing bucket size of \(k = 3\) suffices to find or 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 the 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 fairer in the case of SP than SB neighbor selection (Fig. 16). That is, the load assigned to cooperating and stable nodes is less abusive configuring Sophia with SP, avoiding these nodes being eventually flooded by routing requests. As depicted in Fig. 16, the received transactions CDF corresponding to Kademlia presents the same shape, irrespective of whether nodes are malicious or not. Such a result was expected since only Sophia is capable of discerning between both types of node.

As we expected, the Oracle shows the best LSR values and, consequently, the least fair load balancing in Fig. 16. 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.

6.6. Whitewashing

Finally, we want to assess the impact of whitewashing on Sophia’s performance. In this threat model, attackers change their identities after performing \(\theta\) lookups. However, since Sophia assumes that nodes incorporate a mechanism to retain their identities, honest nodes will keep their identifiers during the simulation.

This experiment has a slightly different parametrization from 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 so that attackers can choose several identities during the simulation.

Second, in this experiment we make use of a sibling zone of \(S_z = 11\) bits. This is needed to avoid stale identities produced by the whitewashing activity falsely increasing the lookup success. In the particular case of Kademlia, when nodes are uniformly spread over the identifier space, their lowest \(k\)-buckets tend to be empty. Thus, if attackers periodically change their identities they will slowly fill up a fraction of the lowest \(k\)-buckets of other’s routing tables. The side effect of this fact is that, virtually, the 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. By determining an appropriate size for sibling zone, lookups do not need to use the lowest \(k\)-buckets; providing then a truthful notion of routing success.

As we can see in Fig. 17, for a small fraction of whitewashers, 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 in the PH compared with the scenario without whitewashing, which 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, the \(h\)-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 small to moderate fractions of whitewashers whenever the majority of participants are persistently identified.

7. Conclusions

The scale of large-scale distributed systems is expected to grow more in terms of number of machines, locations and administrative domains in the next few years. To address scalability, many computing platforms exploit DHTs to allow other components of the system to benefit from their high scalability. However, DHTs present some important weaknesses that have slowed their adoption in communities who care strongly about security, such as the Grid community. To address routing vulnerabilities, we have introduced Sophia, a novel and generic security technique which combines iterative routing with local trust to fortify routing in non-deterministic DHTs. Further, we have compared our solution with state-of-the-art routing techniques, in both stable and dynamic

---

Fig. 16. LSR and received transactions CDF of Kademlia and Sophia configured with Select Best (SB) and Select Probabilistic (SP) neighbor selection policies.
scenarios and under different attack models, with excellent results. In both scenarios, Sophia requires fewer parallel paths to tolerate malicious nodes than pure routing redundancy techniques. This result means that Sophia can achieve important network traffic savings compared with redundant routing techniques, increasing the scalability of the underlying DHT. In addition, Sophia is capable of dynamically recovering from unexpected attacks and tolerating whitewashers, a key property for open Grid platforms and P2PGrids.

Based on our analysis and simulation results, we are now in a position to state that the improvement of routing paths is more effective than multi-path routing, which comes at almost no cost as in the case of Sophia, and represents a significant step towards the development of a secure lookup service for large-scale systems.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Innovation through projects DELFIN (Self-Adapting Decentralized Systems for the Future Internet, TIN2010-21410-C03-03) and REALCLOUD (Real Data Center Cloud Services and Environment, IPT-2011-1232-43000).

Finally, we thank the anonymous reviewers whose suggestions and observations have been very useful improving the quality of this article.

References


Raul Gracia-Tinedo is a junior researcher at the University Rovira i Virgili (URV), Tarragona, Spain, in the AST research group. His research interests are related to peer-to-peer applications, Grid and Cloud computing. Contact him at raul.gracia@urv.cat.

Pedro García-López is professor in the Department of Computer Engineering and Maths at the University Rovira i Virgili (Spain). He obtained his Ph.D. in 2003 in the University of Murcia on the subject of Collaborative Distributed Systems. During his Ph.D. he also worked in the University of Ghent (Belgium) and CMI-FIT Bonn. His research topics are distributed systems, peer-to-peer systems, software architectures, middleware systems and collaborative environments. He has published more than 50 papers and participated in several Spanish and European research projects. He currently leads in the “Architectures and Telematic Services” research group in Tarragona. Furthermore, he chaired the IEEE Collaborative Peer-to-Peer Systems Workshop (COPS 2008–2010) and was a member of the IEEE P2P Program Committee (2009–2011). Contact him at pedro.garcia@urv.cat.

Marc Sánchez-Artigas received the Ph.D. degree with European Mention in 2009 from the Universitat Pompeu Fabra, Barcelona, Spain. During his Ph.D., he also worked at Ecole Polytechnique Fédérale de Lausanne (EPFL) under the supervision of Professor Karl Aberer. In 2009, he joined the Universitat Rovira i Virgili, Department of Computer Engineering and Maths as an assistant professor. He received the Best Paper Award from the 32th IEEE Conference of Local Computer Networks (LCN) held in Dublin, 2007. He has participated in several Spanish and European research projects, such as IST-POPEYE, PLANET, P2PGRID, DELFIN and RealCloud. He also is a member of the Organizing Committee of the International Conference IEEE P2P 2012 that will be held in Tarragona. He has coauthored more than 50 papers published in international journals and conference proceedings. He also served as a Guest Editor for Computer Networks Journal, Elsevier.