Hindawi
Security and Communication Networks
Volume 2018, Article ID 4963932, 16 pages
https://doi.org/10.1155/2018/4963932

Research Article

A Balanced Trust-Based Method to Counter Sybil and Spartacus Attacks in Chord

Riccardo Pecori [1] **and Luca Veltri** [2]

[1] *SMARTEST Research Centre, eCAMPUS University, Novedrate, CO 22060, Italy*
[2] *Department of Engineering and Architecture, University of Parma, Parma, PR 43124, Italy*

Correspondence should be addressed to Riccardo Pecori; riccardo.pecori@uniecampus.it

Received 23 February 2018; Revised 27 September 2018; Accepted 14 October 2018; Published 12 November 2018

Academic Editor: Carmen Fernandez-Gago

A Sybil attack is one of the main challenges to be addressed when securing peer-to-peer networks, especially those based on Distributed Hash Tables (DHTs). Tampering routing tables by means of multiple fake identities can make routing, storing, and retrieving operations significantly more difficult and time-consuming. Countermeasures based on trust and reputation have already proven to be effective in some contexts, but one variant of the Sybil attack, the Spartacus attack, is emerging as a new threat and its effects are even riskier and more difficult to stymie. In this paper, we first improve a well-known and deployed DHT (Chord) through a solution mixing trust with standard operations, for facing a Sybil attack affecting either routing or storage and retrieval operations. This is done by maintaining the least possible overhead for peers. Moreover, we extend the solution we propose in order for it to be resilient also against a Spartacus attack, both for an iterative and for a recursive lookup procedure. Finally, we validate our findings by showing that the proposed techniques outperform other trust-based solutions already known in the literature as well.

1. Introduction

By now, peer-to-peer (P2P) networks, be they structured or not, concern a significant and mature slice of the overall Internet traffic. Their usage ranges from file-sharing to VoIP applications [1], from on-line alerting systems to intrusion detection, and so forth. Moreover, their deployment is experiencing a new flourishing in all those scenarios involving the Internet of Things (IoT) [2]. The most important type of current structured P2P networks is the one relying on Distributed Hash Tables (DHT) algorithms. Among these, we can mention the well-known Kademlia [3], Chord [4], CAN [5], and Pastry [6], but other ones are emerging. A DHT is a distributed structure that maps identifiers to values, similar to a hash table. The lookup is performed through an efficient routing mechanism leading to the peer that actually maintains the mapping. In a DHT-based P2P network, the identifiers generally refer to both the peers (peer IDs) and the resources (resource IDs). The logical overlay and the routing tables of DHTs feature a solid structure allowing them to perform in a quick and simple way, but, at the same time, make them subject to various malicious attacks. The so-called Sybil attack [7] is a prominent example of these types of attacks. This attack usually involves an attacker managing a great amount of multiple false IDs, called *sybils*, able to taint the routing tables and, as a consequence, capable to disrupt or degrade the main operations of the DHT.

1.1. Novelty, Contribution and Motivation. In this work we consider a scenario that is implicitly infected by *sybils*, but we also address Spartacus-behaving nodes (*spartaci*), i.e., nodes that steal the IDs of other nodes. This is one of the main novelties of this work, since a Spartacus attack, i.e., a variant of the Sybil attack, is not very studied in the relevant literature yet. Differently from [8], which provides an admission control system, we do not focus on limiting the access of malicious nodes to the P2P network, but, similarly to the contribution in [9], involving a clean routing process, we tried to devise a trusted lookup and storage mechanism, involving only those nodes turning out to be the most trustworthy. As a consequence, the querying node will be able to take a decision by itself about which nodes to trust, by

considering, in a dynamic and evolving way, the interactions it experienced.

In the aforementioned framework, we investigate and compare some reasonable trust-based techniques to avoid, or better still, to moderate the misconduct of malicious peers, be they *sybils* or *spartaci*. In particular, we focus on the well-known Chord as DHT, trying to modify its lookup as well as its storage and retrieval procedures in a way similar to what has already been done for Kademlia in [10]. Since Chord has a different way of computing the distance between pairs of nodes and its lookup and storage and retrieval procedures are different from those of Kademlia, we needed to devise proper modification with respect to the solution presented in [10].

The contribution of this work is multifaceted and recapitulated in the following points:

(1) We readapted a trust-based mixed strategy, which already proved effective in Kademlia both for routing [11] and for storage and retrieval procedures, to Chord, verifying its effectiveness during lookup as well as storage and retrieval procedures in a network infected by *sybils*.

(2) Then we tested the proposed solution, called S-Chord, in a Spartacus attack scenario, considering both iterative and recursive lookup procedures, and we propose some improvements in order to make S-Chord resilient also to a Spartacus attack.

(3) Finally, we compare the proposed strategy with other trust-based solutions published in the literature along the years.

1.2. Structure of the paper. The remainder of this article is structured in this way: in Section 2 we summarize the main concepts about the Sybil and Spartacus attacks and we describe some possible solutions already analyzed in the literature, and in Section 3 we investigate the effects of a Sybil attack in Chord, while we accurately present the extension of the technique in [10] to Chord in Section 4. In Section 5 we study the proposed strategy in a Spartacus attack scenario showing how the aforementioned procedure may result effective also in presence of *spartaci* through some degrees of fine-tuning. Finally Section 6 seals up the work and provides some suggestions on possible future followups.

2. Background and Related Works

The Sybil attack, firstly described in 2002 by Douceur in [7], exploits the redundancy of DHT networks, and it is usually launched through a malicious physical entity owning multiple virtual and logical identities. The trick is to introduce into the P2P network many fake identities (the so-called *sybils*), which are controlled by one single attacker in the physical layer. This allows attackers to monitor the traffic, to partition the network, e.g., through an eclipse attack, or to misuse the DHT in different ways, e.g., performing a Distributed Denial of Service (DDoS) attack [12].

Different types of threats can be generated by the *sybils* and they can be classified into the following categories: (i) routing table invasions, (ii) storage and retrieval malfunctions, and (iii) miscellaneous attacks [13]. The first types encompass incorrect lookups and wrong routing table updates, and the second ones regard denying to store resources or simulating to have resources the nodes actually do not own, while the third types may concern inconstant behaviors, overloading of specific nodes, quick joining and leaving the network, and so on.

Many solutions have been devised during the years for solving or mitigating the effects of a Sybil attack: they range from the introduction of trusted certification and computational puzzle approaches to costs and fees and trusted devices [10].

Some renowned solutions are Whanau [14], X-vine [15], and Persea [16]. However, these are protocols that leverage on a further social network of trusted relationships between the users of the P2P network and they need proper datasets to be evaluated. These are important limitations and overheads that, if not available, do not allow one to enact any countermeasure against the *sybils*. Moreover, these solutions take for granted that friends in a social network are trusted users in a P2P network, something not always true. Furthermore, they restrict the access of the *sybils* to the P2P system by constraining the number of possible paths among good and malicious peers. Conversely, we resort to a solution that applies to an open scenario where the *sybils* may freely join the network: we do not pose limitations to the possibilities of an attacker.

The Sybil attack is still investigated nowadays, as demonstrated by some recent works such as [17, 18]. In [17] the authors focus on solving a limitation of Persea through lookup inspection for detecting the *sybils*, while the latter presents VoteTrust, a system to detect *sybils* through the friend invitation graph of social networks. However, both of these systems focus on *sybils* detection. The scenario of our work envisages a P2P network inherently infected by malicious nodes instead. Moreover, both works provide an active collaboration among good nodes for *sybils* detection and this may introduce a further overhead that our solution carefully avoids through an automatic trust computation.

In this work, we analyze routing as well as storage and retrieval attacks in a Chord DHT and provide a solution, already proven effective in Kademlia [10], using trust metrics in a balanced way. We chose to focus on Chord, as it is one of the most well-known, relevant, and representative DHTs, and even if introduced in the early 2000s, it is still currently studied and considered a reference DHT for its simplicity and efficiency. This is, for example, shown by the following: a recent work that uses Chord in large scaled P2P networks in conjunction with a dynamic trust computing model [19], as well as recent studies on the further improvement of Chord inherent efficiency [20] and correctness [21], or on the joining time of Chord nodes through the usage of anchor peers in an educational scenario [22]. Further recent contributions have concerned the usage of Chord in mobile networks [23] and for location-based services [24]; therefore, Chord's security and reliability are still important aspects to be carefully assessed and studied.

Some solutions for improving Chord's security are already present in the relevant literature, such as [25] where the authors deploy certificates and signatures. Nevertheless, an increased number of messages, compared with standard Chord, are required; moreover, the authors try to remove malicious nodes from the network, something leading to a scenario that differs from the one considered in this work where we preferred another approach, i.e., accepting *sybils* in the network and trying to overcome their malicious activity, through direct trust, in such a way an attacker cannot recognize a countermeasure has been enacted. This is performed by following some successful solutions mainly studied in [10, 26].

Concerning trust and reputation in general, in [27] a method for comparing trust models based on a hierarchical fuzzy inferring model is proposed. However, the contribution focuses only on a file downloading scenario, like the mechanism proposed in [28], which is moreover applied only to nonstructured P2P networks. Conversely, the mechanism proposed in [29] can be applied in P2P networks based on DHTs, but it is difficult to implement in a real world scenario.

Concerning the application of trust and reputation directly to Chord, some ideas can be found in [30–33]. In [31] the authors apply some of the strategies proposed by Koyanagi in [30], extending them to security purposes and not limiting them to maintenance aims. More precisely, the authors of [31] exploit Bayesian networks in order to enrich Chord finger tables with a further column, called "trust," whose value will be used in order to finalize trustworthy lookups. They took advantage of a sort of variable central entity. In contrast, in our work, we prefer to keep a kind of decentralized web-of-trust, as it is the case with [34], and we consider a more general trust score, not limited to some features like age or downloads and uploads that could be specific to some P2P network usages.

In [32], Koyanagi et al. propose a solution similar to ours where a general trust score computation and its weighing are used. However, our work differs in the fact that we consider a different balanced strategy, a mix between standard and trusted Chord using a tunable parameter. We also take into account the so-called environmental risk, analyzing the outcomes of a growing number of *sybils*, not only monitoring a constant percentage of them. Moreover, unlike [32], (i) we consider churn a normal action of peers, (ii) our solution does not require a look-ahead phase, and (iii) the computation of trust is simpler as we do not consider trust propagation but only direct experiences.

Finally, the work in [33] is the most recent one attempting to apply a trust model to Chord by leveraging onto *guaranteeing* and *archive* peers, whose reputation is evaluated together with the one of *service* peers. The proposal provides also an incentive system and an anonymous reputation management strategy; nevertheless, the authors consider at least 4 different roles for a peer and they envision a complex system for establishing guarantee relationships (at least 10 messages) and a lookup transaction (at least 14 messages). This is in contrast with our idea of keeping the system as simple as possible, featuring a minimum number of changes and a minimum overhead with respect to standard Chord; moreover, we do not set up different roles for the peers, which would lead to extra overhead, but we prefer a distributed approach, where all peers are at the same level.

The Spartacus attack, a variation of the Sybil attack, has not yet been thoroughly examined within the relevant literature. In this attack, itself especially dangerous in an environment focusing on trust and reputation, the malicious physical entity does not generate a great amount of pseudonymous logical IDs but simply steals them from other real nodes, inheriting their trust value. Such an action takes place in the bootstrap phase, when a *spartacus*, intending to enter the network, looks for a possible bootstrap node, and later for nodes with high trust score. The *spartacus* pings these nodes and, when they appear to be disconnected, it replaces them by copying the hash of their IDs. If no way exists to bind a node to its virtual ID, and such is the scenario we consider, and the node replaced by the *spartacus* needs to choose another virtual ID and must start again to create its own trust score from the beginning. This malicious activity allows *spartaci* to acquire a higher trust score than *sybils*, at least initially, and if churn is enacted, this attack is much more feasible and dangerous. An example of possibly disrupting consequences can be found in a Fog computing IoT scenario, where smart gateways, belonging to the fog layer, may be part of a virtual peer-to-peer overlay. If one of these fog gateways is attacked by a spartacus node, all the information coming from the sensors managed by the attacked node may be easily lost or counterfeited.

Some solutions against the Spartacus attack exist, but they are based on public cryptography, as it is the case with the "kad-spartacus" extension of the "kadtool" package (http://www.npmjs.com/package/kad-spartacus), on a central server, on the binding between IP address and virtual ID or on an active verification of the node about to join the network, carried out by those nodes having already joined the network. However, these are not fully decentralized solutions. They require certification authorities and a public key infrastructure, leading to possible bottlenecks due to the presence of central servers or further operations on the joining procedure, such as the binding with network or physical addresses or other verification activities, which limit the entering of the peer-to-peer network and can cause additional overheads. Conversely, in our scenario we consider the joining procedure free from further constraints or limitations, and we do not use central authorities or servers.

In the last part of this work we propose a new adaptation of the solution firstly proposed for a Sybil attack, called S-Chord, to work effectively also against a Spartacus attack. Furthermore, we provide also a comparison between the mixed trust-based technique and those described in [32, 33] and demonstrate they reach worse results when considering successful lookups and the average number of hops.

3. Sybil Attack in Chord

In this part of the paper, after briefly recalling Chord [4], we study the degradation of the performances of the DHT in presence of a Sybil attack by means of simulations,

considering both lookups and storage and retrieval operations. Some distinctions are made also between iterative and recursive lookup procedures.

3.1. A Brief Overview of Chord. Peers and resources are associated with a unique *m-bit* identifier (m is the ID length), calculated by using consistent hashing of the peer addresses. In the following, we indifferently name Chord entities as "peers" or as "nodes." Chord logical IDs are ordered following a numerical circle called *Chord ring*, where distances are computed modulo 2^m. The owner of a resource key k is defined as the first node with an ID matching or following k in the key-ring modulo 2^m. This node is named $successor(k)$.

The definition of distance between a node n_1 and a node n_2 is $(n_2 - n_1) \ mod \ 2^m$; therefore it is asymmetric. Often, due to reliability reasons, redundancy is added and more than one node may be in charge of a certain key. Hereafter we separately describe the procedures for (i) lookup and (ii) join.

3.1.1. Chord Lookup. A lookup operation is used to find out what node is in charge of a specific identifier. Chord features two lookup procedures, called, respectively, *basic* and *accelerated*. In the basic lookup, each node simply contacts its current successor, since the requests are transferred around the ring through successor pointers until they find a pair of node identifiers spanning the requested key identifier; the second in the pair is the target of the lookup query.

The accelerated lookup takes advantage of further routing information: a so-called *finger table* made up of m entries. The i^{th} entry in the table of node n is the first node s_i following n by a distance of minimum $d_i = 2^{i-1}$. Essentially, s_i is defined as $successor(n + 2^{i-1}) \ mod \ 2^m$, with $1 \leq i \leq m$. s_i is termed as the i^{th} finger of peer n, and it can be denoted by $finger(n, i)$. An entry on any finger table includes both the identifier and the peer address.

It is possible to summarize the accelerated lookup procedure for any given key through the following steps:

(i) A node is requested for a certain key.

(ii) Should the node be in charge of the key being requested, the lookup ends.

(iii) Otherwise, should the node's successor be in charge for that key, the requested node responds with the successor node.

(iv) Else the requested node returns the highest entry in the finger table that comes before the key.

This version of the lookup algorithm assures that lookups, in a N-node network, require at most $O(\log_2 N)$ hops in order to succeed. This is because the topological distance between the current requested node and the target node is divided by two at each repetition of the algorithm. Both the basic and the accelerated lookup may be performed in an iterative or recursive way [35]; during the former the querying peer has full control of the lookup process, while in the latter case at each step a different peer is in charge to forward the request.

3.1.2. Chord Join. A new node joins a Chord network by computing its ID and contacting a bootstrap node. This is performed by executing a *find successor* procedure, using the joining node ID as an argument. Once the joining node has discovered its successor, it establishes its successor node to be the admitting peer and sends to it a $notify$ message that informs the admitting peer that a new predecessor has joined the network.

3.2. Simulation Conditions. In order to perform numerical simulations, we employed DEUS [36] together with its inherent Chord implementation. This simulator is based on discrete events and on virtual seconds (*vs*) as regards simulation time. For the sake of simplicity, we only considered stationary conditions so that, regardless of peers churning, a fixed number of them are constantly active in the considered scenario. The number of peers that behave well, i.e., according to the standard Chord algorithm, is constant and set to 10,000, while malicious peers, i.e., the *sybils*, vary across the different simulations, and they can acquire the following figures: 0, 2,000, 4,000, 6,000, 8,000, 10,000, 15,000, and 20,000. Those values reaching more than 10,000 are considered as cases of overwhelming presence of malicious nodes. The simulation time is set to 30,000 *vs*. Good nodes are subject to a churning Poisson process of parameter λ (equaling 10 joins/leaves every 1 *vs*); malicious nodes, instead, join and leave following a random period process. The ID length m is 128 bits, while the observation cycle amounts to 1,000 *vs*, allowing for 30 observations for each simulation.

The DEUS implementation of Chord does also feature a parameter, accounting for a maximum waiting time, named M_W, which is set to 1,000 *vs*. This parameter is used to simulate those nodes responding too slowly to a request. Indeed, we only concentrate on lookup procedures that correctly terminated, that begin only when the network is in a stationary condition, and that do not cause M_W to expire. Should M_W expire, we labeled these lookups as "unended."

As a final remark, for all parameter sets, we considered the averages of the results computed over the 30 datasets coming from each observation cycle and over various simulation seeds. This was done in order to achieve a confidence interval amounting at least to 95% for the obtained results.

3.3. How the Sybils Affect Standard Chord. We study both lookup attacks as well as storage and retrieval attacks and both iterative and recursive accelerated lookups. In particular two malicious basic behaviors have been considered for lookups. *Sybils* asked for a next hop could either

(1) refuse to answer or

(2) respond with a randomly chosen peer ID.

Concerning storage and retrieval, we analyze different behaviors:

(3) refusing to store a resource,

(4) accepting a resource and then refusing the relative retrieval queries,

(5) returning a null resource in response to a retrieval request,

(6) blocking storage and retrieval operations for certain specific resources chosen randomly.

In particular, since just the first and third behaviors are able to effectively disrupt standard Chord operations, the second and the fourth behaviors are studied only against our trustbased proposal in the following sections, in order to stress its effectiveness.

Although various metrics to study a Sybil attack are present in the literature ([37, 38]), we mainly focused on the following:

(i) for lookups, the average amount of successful requests and the mean hops per lookup;

(ii) for storage and retrieval operations, the mean of successful store (PUT) and retrieval (GET) procedures.

In Figure 1 we show lookups in a scenario where *sybils* refuse to provide an answer to the querying peer (case (1)) of the aforementioned list). We refer to successful operations when lookups reached the peer correctly in charge of the searched key, regardless of whether the lookup procedure has involved *sybils* or not. The outcomes demonstrate an intense decrease in successful lookup procedures while malicious nodes grow. Unsuccessful operations, which overtake successful lookups if the *sybils* overtake the 8,000 threshold, refer to lookups that have not yet finished once the M_W timeout has expired; therefore they are labeled as "unended."

Furthermore, in the same figure, we show that a little fraction (2,000 or 4,000) of malicious nodes does not influence too much the overall performances, so there should be a minimum amount of malicious nodes that makes a Sybil attack successful. Conversely, those lookup requests being unsuccessful are somehow bounded when malicious peers are less than 6,000, while they increase very much and exceed successful lookups from 8,000 on. It is also relevant to the saturation behavior experienced by the curve when the *sybils* are more than 10,000. This could be due to the fact that the *sybils* have reached and overtaken good nodes and their negative effect tends to stabilize.

In Figure 2 we consider the other malicious behavior: a random response (case (2)). In this case, the *sybils* respond to lookup queries with the ID of a random peer, regardless of whether it is malicious or not. In these simulation conditions, we point out a different assessment for genuine and non-genuine lookups, as this could be useful to analyze the behavior of iterative and recursive lookups. As a matter of fact, when we consider a no response attack, the iterative and recursive lookups are similarly affected; that is, the procedure is unended, whereas when we consider a random response, this could lead to different outcomes according to the number of *sybils* encountered during the procedure. More specifically, there are three cases: (i) successful genuine lookups, (ii) successful non-genuine lookups, and (iii) unended lookups. Genuine lookups refer to those lookups not affected by *sybils* at all, whereas non-genuine lookups are the lookups in which at least a *sybil* has been contacted during the procedure. It

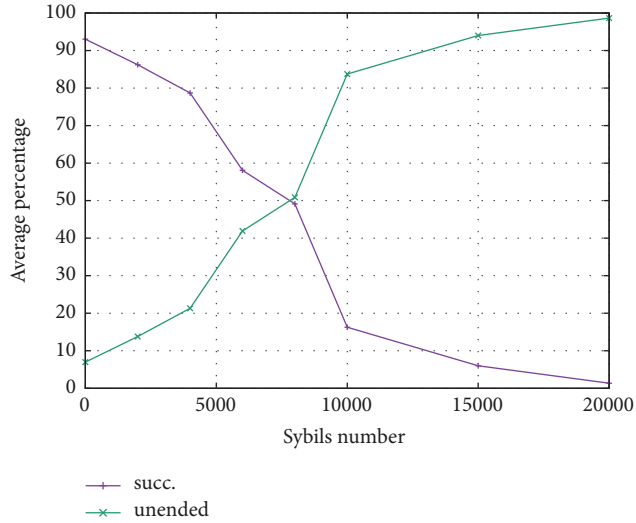

Figure 1: Successful and unended lookups versus the number of *sybils*, whenever *sybils* do not respond.

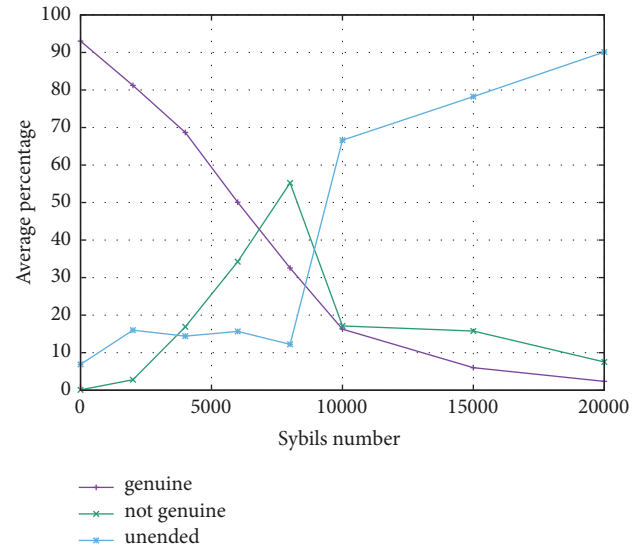

Figure 2: Successful genuine, successful non-genuine, and unended lookups versus the number of *sybils*, whenever *sybils* respond with random IDs.

is evident from these definitions that in case of no response from the *sybils* this different behavior could not be detected.

Returning to Figure 2, while the number of genuine lookups decreases gently in a way similar to the one depicted in Figure 1, the number of non-genuine ended lookups experiences a peak in correspondence to a number of *sybils* equaling 8,000 and then it sharply decreases reaching the performances of genuine successful operations. The opposite behavior can be observed for unended lookups that, when the *sybils* number more than 8,000, undergo a sharp increase.

While non-genuine lookups have the chance of considering good nodes leading to the correct target, when *sybils* are not overwhelming, this cannot take place anymore when the quantity of *sybils* surpasses a given threshold, which, looking

at the trends in the graph, can be set to 8,000. On the other hand, when this threshold is passed, the probability for the lookups to encompass a great majority of *sybils* increases and, as a consequence, the lookup operations may be stalled into never-ending cycles. Figure 2 shows aggregated data for both iterative and recursive lookup; however, we noticed that successful lookups belong more to the iterative procedures, rather than to the recursive ones. This is because the Chord requesting node can check whether a lookup is moving towards the queried key and, in the case of the iterative procedure, can decide not to follow the wrong suggestion of a *sybil* and to query again the previous peer. This is not possible in a recursive lookup procedure. We will return on the difference between recursive and iterative lookups later, as a difference will emerge when we apply our trust-based extension to Chord.

In Figure 3 we compare the two malicious behaviors considering the average of hops per lookup, regardless of the fact that the lookup is genuine or not, and not considering if it succeeds or it is unended after the M_W timeout has been reached. As it may be noticed, the curve concerning the *sybils* not responding, experiences a first increase and then a decrease, whereas the curve of the second malicious behavior undergoes a constant growth of the hops per lookup. This could be due to the fact that in the second case an increasing number of *sybils* may lead to a never-ending lookup process, while in the first case, above a certain threshold of *sybils* in the network, the chance to reach good behaving nodes decreases and so lookups experience a longer waiting time without increasing the number of hops that tends to decrease on average. It must be highlighted that in the first case the mean quantity of hops does not surpass the typical Chord threshold ($\log_2 N$, with N the overall nodes in the network: good nodes plus *sybils*), ranging from 13.28 to 14.87 depending on the variable amount of *sybils* in the considered scenario. In the second case this happens whenever the *sybils* equal or go over 10,000.

In Figure 4 we show the trend of successful storage and retrieval operations when the amount of *sybils* in the network increases. We encompass both PUT and GET procedures together and the basic malicious behaviors related to these two operations: refusing to accept a legitimate PUT and responding with a null value when receiving a GET request. However, it is unuseful to consider whether the procedure is iterative or recursive or whether it is genuine or not, because we do not regard maliciousness in the lookups: the nature of the last peer reached in the lookup process is the only thing that counts in this case. In the figure, we report also the curve of successful lookups in presence of no response from the *sybils* for the sake of comparison, since all these attacks are quite similar, regarding the refusal of performing the requested operation.

As can be inferred from the figure, the decrease of the successful storage or retrieval operations is more evident than that of lookups (already shown in Figure 1), especially when the *sybils* number more than 4,000. We can explain this trend considering that, in this case, we consider two malicious behaviors at the same time, i.e., both for PUTs and for GETs, while the lookups were affected by only one

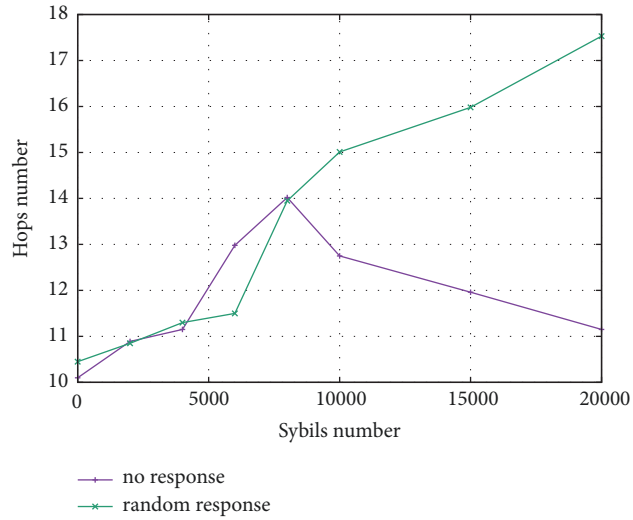

Figure 3: Average number of hops versus the number of *sybils*, when comparing two different malicious behaviors.

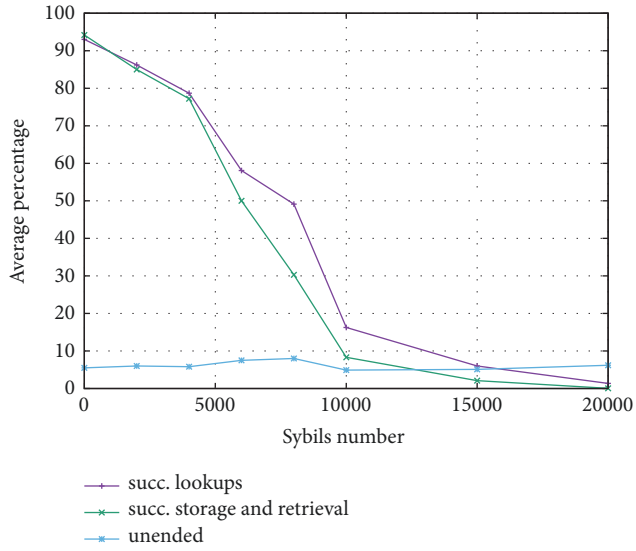

Figure 4: Comparison between successful lookups and successful storage and retrieval operations versus an increasing number of *sybils* in standard Chord.

form of attack, i.e., no response. This trend is not so marked in those cases where there are few *sybils* in the network, confirming again that a small number of malicious nodes do not influence the performances very much. Moreover, we can notice that the number of unended storage and retrieval procedures is almost constant and very limited, and this is correct since this percentage is only dependent on the M_W parameter rather than on malicious behaviors in the storage and retrieval procedures.

4. S-Chord

In this section, we investigate some possible countermeasures to limit the impact of *sybils* on lookups, as well as on

storage and retrieval procedures, by integrating a simple but effective trust-based mechanism into the standard Chord environment and procedures, called S-Chord. The proposed approach is similar but yet different, to the one proposed for Kademlia in [10].

We propose improving the resilience of Chord to a Sybil attack, using trust information in the lookup as well as in storage and retrieval operations, but with a differentiated trust management for the two types of operations. The proposed solution considers a trust metric when sorting local finger tables. We suppose, contrarily from Koyanagi [32], that sorting the entries of such tables regarding only trust would not be effective in some cases, since the best solution is obviously the one of standard Chord. We introduce a novel metric for computing the distance of peers and call it "new distance" (nd_i). This new distance takes into account also trust, and it is computed, for each peer i in the finger table, according to

$$nd_i = b \cdot 2^{i-1} + (1 - b)\, T_i \cdot 2^{i-1} \quad (1)$$

where nd_i is the new distance, b is a balancing term, whose values may range from 0 to 1, while T_i is the trust factor, with values from 0 to 1, of the i^{th} entry of the finger table. The formula in (1) guarantees that the actual successor is not skipped, as the base 2 exponentiations are always multiplied by factors less than 1. The farthest node according to the new distance is chosen as the candidate for the next step of the iterative or recursive lookup process. This is mandatory also in case a recursive lookup is considered and the peer currently in charge for the lookup is a *sybil*.

4.1. Trust Score. The trust score is calculated following a definition coming from the PET model [26], and not from the model in [32] or in [33] as regards Chord, or from the model in [9, 39] for Kademlia. In this way, we try to maintain the procedures as simple as possible and, at the same time, to insert the least overhead as possible. In case there are no previous interactions with other peers, we account for a global risk factor, which allows us to escape the grace phase used in [9] and the indirect trust employed in [32]. Conversely, we propose a proper mixture of direct trust and risk by means of two numerical weights. The trust score (T) is defined as follows:

$$T = W_{R_e} \cdot R_e + W_{R_r} \cdot (1 - R_r) \quad (2)$$

where R_e and R_r are the direct trust and the risk, respectively, whereas W_{R_e} and W_{R_r} are the weights assigned to the direct trust and to the global risk of the network (values ranging between 0 and 1, and one the complement to 1 of the other one), respectively. We simplify further the model with the introduction of a unique negative level of interaction (L = low grade). This implies that the risk R_r is calculated with the subsequent equation:

$$R_r = \frac{N_L}{N_T}. \quad (3)$$

N_L accounts for all interactions, with a low grade of service, provided by a certain peer in the whole network. It may concern various behaviors, depending on the particular scenario:

(i) considering only the trust in lookups, N_L may encompass timeouts of lookups, no responses, and the like;

(ii) considering only the trust in PUT and GET procedures, N_L encompasses storage refusals, retrieval of null resources, and so on.

(iii) considering trust in all possible operations, N_L is computed considering all the previously mentioned malicious responses.

N_T represents all the requests generated by the nodes of the considered scenario towards the peer under evaluation, be they lookups or PUT or GET operations.

The risk of a certain peer is a global measure that in a realistic implementation could be maintained through mechanisms similar to those of blockchain [40].

Conversely, the value of the direct trust depends on the particular trustor node, which stores various values of trust, one for each trustee peer, in a local table. If no communications with a particular peer have taken place yet (e.g., new joining node), t is defined only through $W_{R_r} \cdot (1 - R_r)$; in the opposite situation it comprises the direct trust (R_e) contribution as well, which is computed and updated according to the following steps:

(i) when only lookups are examined: $+1/M$ to all peers involved in a lookup that leads to a correct resource, or $-1.5/M$ to the peers leading to dead-points or wrong targets;

(ii) when only PUT and GET procedures are of interest: $+1/M$ to all nodes that accept to store a legitimate PUT or that return a correct resource, or $-2/M$ to all nodes refusing a storage request or providing a null value as a response to a legitimate GET;

(iii) in case both lookups and storage and retrieval operations are considered: the update of the single direct trusts is included in considering the before mentioned procedures.

M represents the total number of queries, launched by the trustor node, in which the trustees have been involved; they can be lookups, PUTs, or GETs.

This asymmetry in rewarding or punishing the involved peers is devised to foster the correct behavior of nodes and, at the same time, to discourage malicious behaviors by punishing a bogus node more than it could recover with a single good interaction. The greater punishment in storage and retrieval operations ($-2/M$ in place of $-1.5/M$) is given by the fact that we consider jointly a malicious behavior in both PUTs and GETs and this leads to worse performances than in the lookup case as shown in Figure 4.

Similarly to what was done in [9], trust values expire after a certain time (we set this time to 24 virtual hours in our simulations) in order to avoid temporal attacks. This is to prevent that a bogus peer, by colluding with other malicious nodes, may obtain from them a high trust score before starting behaving badly. The negative direct trust update is

a significant difference with [32] and it could be tuned in a more fine-grained way to involve only the last in charge peer, or a certain subgroup of peers, for a single lookup path. This is done with the aim of considering the chance that not every peer of a wrong path acted in a bad way.

4.2. Evaluation of S-Chord

4.2.1. Parameter Tuning. In order to evaluate the effectiveness of S-Chord, we consider the same simulation conditions as those in Section 3.2, and we analyze its performances in a network scenario with a growing number of *sybils*. The tuning of parameter b is very important since it may determine outcomes that may be worse than those obtained with traditional Chord. Therefore, a great simulation campaign was carried out for optimizing parameter b, obtaining a final value of 0.68 when considering only lookups and of 0.60 when considering only storage and retrieval operations. This may be explained by the fact that more importance should be given to the standard procedure for an actual convergence of the lookups rather than for the successfulness of storage and retrieval operations, something depending only on the nature (malicious or not) of the last queried peer.

Concerning the other parameters of the model, W_{R_r} is 1 for a node joining for the first time and 0.3 for those nodes having already joined the network, and W_{R_e} is, by definition, 0 for new joining nodes (this accounts for the lack of direct trust) and 0.7 for already joined ones (this allows having a greater impact of direct trust over the environmental risk, whenever such data are available).

In Figure 5 we compare the performances of different values of the b parameter in case of no response from the *sybils* during a lookup procedure: it is evident that if the balance factor b is not properly set, it can lead to performances worse than in the case of standard Chord. This happens when b approaches zero and trust becomes preponderant, to the detriment of the optimum strategy of standard Chord. Standard Chord strategy is obviously the best one and would be based only on a distance given by 2^{i-1}, where the base 2 exponentiation is not affected by the trust factor. On the other hand, whenever b gets close to 1, the outcomes appear to be similar to standard Chord, and this is correct, as the formula in (1) reduces to 2^{i-1}, the standard form of the distance in classic Chord. From the same figure, it may also be inferred that S-Chord, whenever b is properly set, can limit or mitigate, at least partly, the negative effects of a Sybil attack. This is especially marked whenever malicious nodes are more than 6,000, since the percentage increase in the performances, compared to standard Chord, ranges from 28% (6,000 *sybils*) to 184% (10,000 *sybils*).

In Figure 6 we compare standard Chord and S-Chord in case of *sybils* responding with a random node in a lookup procedure and variable values for b. In this case, we do not consider genuine and non-genuine lookups separately, but they equally contribute to successfully ended lookups as the objective is to tune parameter b. As one can see, whenever the balancing factor is correctly set (b equal to 0.68) the performances of S-Chord are far better than standard Chord, with an improvement ranging from +4.27% (4,000 *sybils*) to +94.84% (20,000 *sybils*). On the contrary, when b is not correctly set to its optimum value, two main cases are possible and they are described in the following.

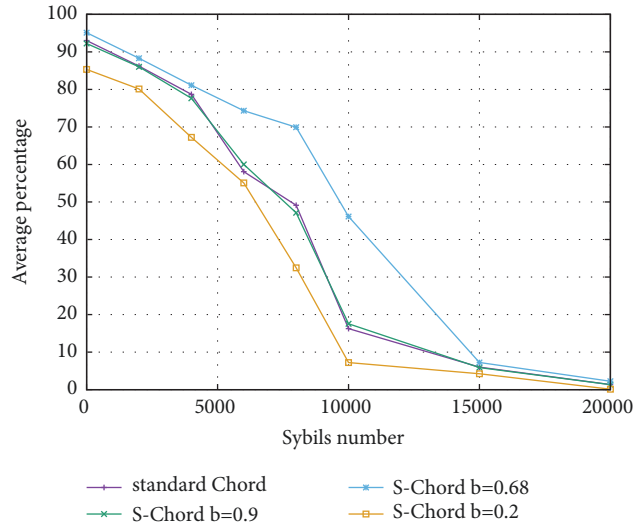

Figure 5: A comparison between standard Chord and S-Chord for different values of b considering the percentage of successful lookups versus the number of *sybils* in the network; "no response" is considered as malicious behavior.

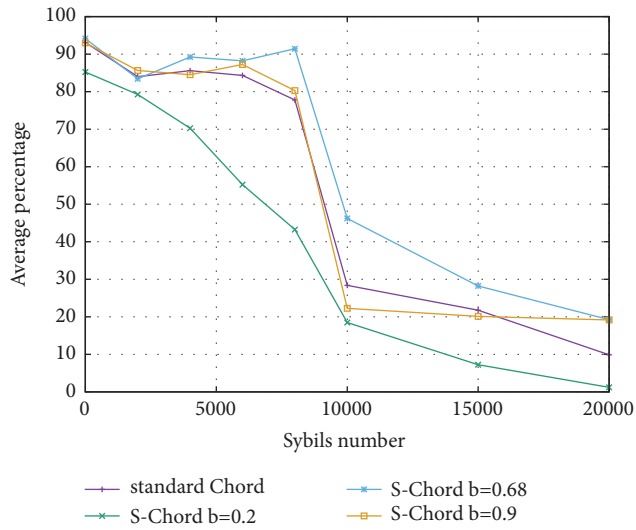

Figure 6: Comparison of successful lookups versus *sybils* number for standard Chord and S-Chord for different values of b. The considered malicious behavior is random response.

The first one of such cases is when b is close to zero: in this situation, represented in Figure 6 by the case with b equal to 0.2, the performances are far worse than standard Chord, even worse than in the case of no response attack shown in Figure 5. This could be explained considering that, in this situation, the new distance is mainly driven by the trust score and the distance metric reduces, more or less, to $T_i \cdot 2^{i-1}$. This has as an ultimate effect on the reinforcement of the randomness in the responses given by the *sybils* themselves.

The second one is when b is close to 1 and this is depicted in Figure 6 by the case of b equal to 0.9. In this case, whenever the number of *sybils* is small, the performances seem to follow the ones of standard Chord, like in Figure 5. This is because the distance metric reduces to the classic one: 2^{i-1}. However, this is not true anymore when malicious peers are more than or equal to 10,000. As a matter of fact, in these conditions the curve undergoes a saturation behavior across the 20% value. This is a very interesting attitude and a tentative explanation is that there could be a threshold of the overall network nodes, be they *sybils* or not, after which the random response behavior should lead always to the same results. This threshold should depend on the balancing factor, as it is not present for lower values of b, the M_W parameter, since it determines the timeout and the classification of a lookup as "unended," and the relative ratio of malicious nodes versus good nodes.

4.2.2. Effectiveness in the Routing Process. In this subsection, we carry out a comparison between S-Chord, standard Chord, Koyanagi's solution [32], Kohnen's solution [9] readapted to Chord, and GeTrust [33] both in terms of successfulness of the lookups and considering overhead complexity. In these simulations, we set the parameter b of S-Chord to its optimum value.

In Figure 7 we make a comparison in terms of successful lookups with an increasing number of *sybils*. The outcomes prove that a "balanced" strategy is better than a solution based only on trust, particularly whenever malicious nodes are not preponderant. It must be stressed that when the *sybils* number significantly more than or equal to 10,000, the number of good nodes, the chance to obtain a successful lookup is always less than 50%. From the same figure, it can be inferred that our solution reaches similar performances as GeTrust; however, S-Chord is less computationally intensive in terms of simulation time, in comparison both to Koyanagi's solution and especially to GeTrust. This is shown in Figure 8, where we make a graph of the average time spent by each trust model after 30 simulation cycles. The best performances, in terms of temporal overhead, reached by our solution are due to the fact that Koyanagi's method requires trust propagation, while in GeTrust there is an overhead of messages both to establish warranted relationships and in the lookup transactions themselves. The solution of Kohnen is the worst in terms of temporal overhead and this is probably due to the usage of certificates and public key cryptography. Because of this temporal overhead, and since its performances are only slightly better than Koyanagi's ones (see Figure 7), we will not take it into account anymore in the rest of the paper.

In Figure 9 we compare standard Chord, S-Chord, and GeTrust solutions, in case of *sybils* responding with a random ID and performing different analyses for genuine and non-genuine lookups as well as for iterative and recursive lookups. Considering genuine lookups are of no interest, as these decrease with the same trend for both our solution and GeTrust, therefore, they are not shown here. What is interesting, instead, is the analysis of the behavior of

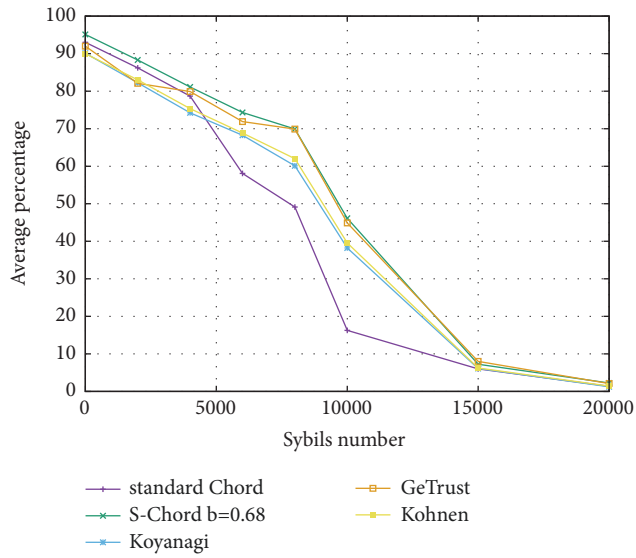

Figure 7: Comparison between standard Chord, the method employed by Koyanagi [32], the method employed by Kohnen [9], GeTrust [33], and S-Chord considering successful lookups versus *sybils* number. The considered malicious behavior is no response.

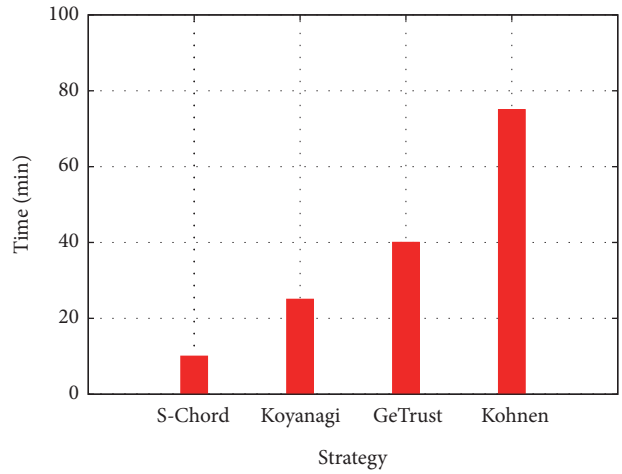

Figure 8: Comparing different strategies using trust in Chord, in terms of computational time of 30 cycles in a simulation.

successful non-genuine lookups and of iterative and recursive procedures separately.

As per the figure, our solution is better than both standard Chord and GeTrust, especially when we look at the recursive lookup procedure. While the iterative lookup procedure undergoes a similar improvement as GeTrust compared with standard Chord (more or less the same curve, and this is why they are not shown), the recursive implementation of S-Chord outperforms them both, particularly whenever the *sybils* number more or less like the good nodes (8,000-10,000). This can be explained thanks to a better spreading of the trust information, since, differently from the iterative procedure, more than a good node is involved in the recursive

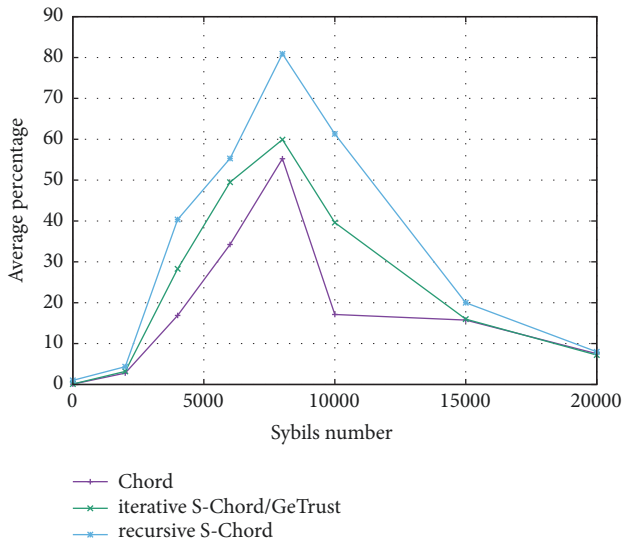

Figure 9: Comparison of standard Chord, GeTrust [33], and S-Chord, considering successful non-genuine lookups versus *sybils* number. The considered malicious behavior is random response.

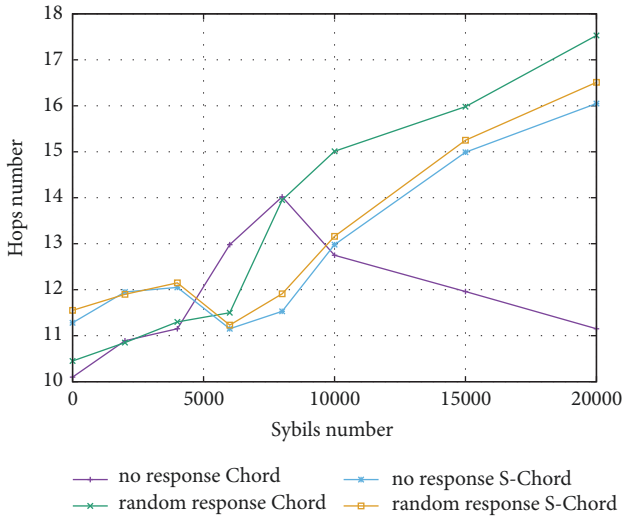

Figure 10: Average number of hops per lookup with standard Chord and S-Chord algorithm when considering two different malicious behaviors.

lookup and thus they can be aware of the right or wrong responses received by other peers contacted in the procedure.

Finally, in Figure 10 we analyze S-Chord concerning the hops mean value per lookup. Under these conditions, when the *sybils* perform a random response behavior our solution seems to lead to better performances (fewer hops), provided that the number of malicious nodes is greater than 4,000. In this case, the increase in the performances provided by S-Chord ranges from +2.35% (6,000 *sybils*) to +14.62% (8,000 *sybils*).

On the other hand, a particular situation takes place in the case of no response from the *sybils*: by using our solution the curve does not experience a maximum anymore; rather it grows constantly from a certain point on, even in those cases when the *sybils* are more than 8,000. Such a behavior is quite different from what happens for standard Chord. However, the threshold of standard Chord performances ($\log_2 N$) is reached and surpassed only from 15,000 *sybils* on, confirming again the goodness of the proposed strategy in presence of a preponderant number of malicious nodes.

Obviously, if our solution is compared with standard Chord algorithm, it leads to worse performances (more number of hops) in some circumstances. This can be mainly inferred from the first parts of the curves, when the number of *sybils* is limited, and thus the performances can be assimilated with an adequate degree of precision to those of standard Chord. As one can notice, in this situation, when bogus peers are fewer than or equal to 4,000, the lines of S-Chord are always above the corresponding ones of standard Chord, meaning that it requires more hops. This is correct, since S-Chord is based on a distance metric that is not optimum but encompasses also the fading value of the trust score.

4.2.3. Effectiveness in the Storage and Retrieval Processes. In this subsection, we perform an evaluation of S-Chord considering storage and retrieval procedures. We do not report, for the sake of brevity, the same comparisons, reported previously for lookups, on the fine-tuning of parameter b, but we take for granted it has been optimized for these procedures to the above-stated value of 0.60. Moreover, this value does not depend on the type of lookup (iterative or recursive), since it obviously depends only on the choice of the last peer in the query. Considering successful PUT and GET operations in presence of basic malicious behaviors leads to graphs similar to the ones provided before for the lookup process, thus in this part we concentrate only on the study of the sophisticated behaviors described in Section 3.3, which consider a good behavior immediately followed by a bad behavior or a selective bad behavior. They are summarized in the following:

(i) accepting a resource and then refusing the relative retrieval queries;

(ii) blocking storage and retrieval operations for certain specific resources chosen randomly.

The first of these malicious behaviors does not influence the effectiveness of storing operations; therefore we focus only on retrieval procedures. In such a situation, the outcomes of the previously proposed solution can decline, as Figure 11 demonstrates; however, this is linked to the quantity of GETs compared to the PUTs, to the number of *sybils* and to the lookup type (iterative or recursive) used to reach the queried node. Since also in this case we experienced the fact that the GET operations preceded by a recursive lookup reach the same performances like the ones of the basic malicious behaviors, we decided to employ the relative curve as a benchmark for the following considerations.

Defining r as the number of PUTs divided by the number of GETs, within a certain period of time (30,000 *vs* in the considered scenario), one observes, according to the following evaluations, that its fluctuation can worsen the

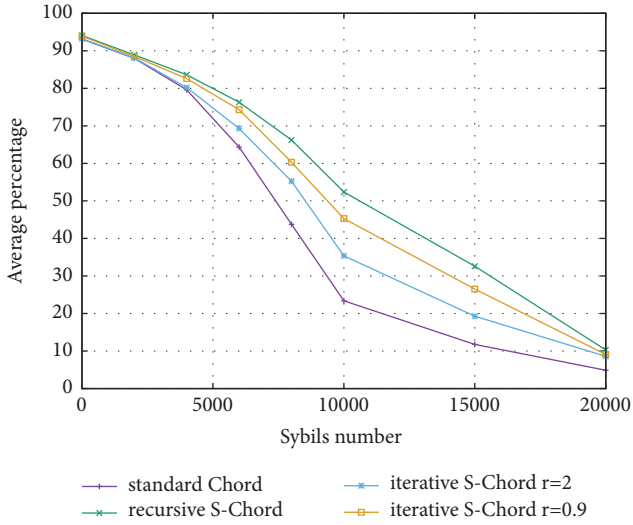

Figure 11: Successful GETs versus *sybils* number with standard Chord and S-Chord considering two different ways of performing the lookup and variable ratio (r) between PUTs and GETs. The malicious behavior consists into accepting all PUTs and into refusing to answer to a retrieval request.

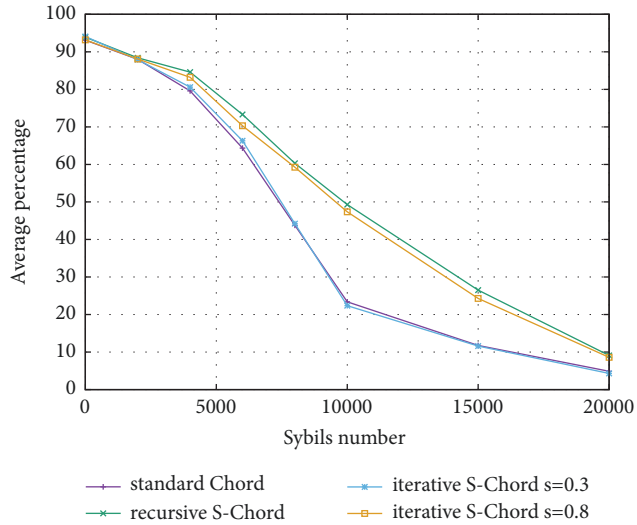

Figure 12: Successful PUTs versus *sybils* number with standard Chord and S-Chord considering two different ways of performing the lookup and variable spreading parameter (s). The malicious behavior consists into blocking storage and retrieval operations for certain specific resources chosen randomly.

outcomes of our proposal, even if they are always better than standard Chord's. This happens especially if r is higher than 1, and malicious nodes pass 6,000 and with an iterative lookup.

Conversely, when r is less than 1, the performances of the proposed solution degrade later in case of GETs preceded by an iterative lookup: when malicious peers become more than 8,000. On the contrary, as already stated, when the GET is preceded by a recursive lookup, the performances are not influenced by r and they mirror the ones experienced when the basic malicious behaviors are enacted.

Conclusively, one can notice that when the *sybils* are not overwhelming in number, the role of r is no more crucial and the outcomes are more or less the same as they are in those situations without these more advanced attacks.

The second sophisticated malicious behavior, concerning, like the first one, storages and retrievals, could influence both PUT and GET procedures. Such an attack is to be evaluated concerning the spreading of a content or resource. In case the resource is very well-liked, S-Chord (be it iterative or recursive) can be employed with no changes, while if the searched contents are not so popular, the outcomes of S-Chord fluctuate according to a spreading factor s, which measures the popularity of the searched resource and can be determined as the number of storages for a certain content divided by all PUT procedures.

This may be easily inferred from Figure 12, where successful storages are analyzed for standard Chord, iterative S-Chord with a varying s parameter and recursive S-Chord. Also, in this case, the curve of recursive S-Chord is used as a benchmark since it is very similar to the one obtained under basic malicious behaviors for PUTs and GETs using S-Chord.

As it may be inferred, if s is higher than 0.5 (widespread content) the trend of iterative S-Chord is very similar to the one of recursive S-Chord, whereas when s is lower than 0.5 the trend becomes irregular, resulting sometimes below and sometimes above standard Chord, according to the number of malicious nodes. In this attack scenario, the trust management could be readapted to execute storages or retrievals in a correct way. The adaptation may influence both the evaluation of the risk, introducing another parameter in addition to N_L, and the update of the trust score. However, such a focused attack could influence the final outcome of the whole attack strategy, since not all storage procedures are compromised and executing various attacks for every existing content could be computationally very heavy.

As we have shown, recursive S-Chord usually obtains better results and features fewer problems compared to the iterative one. This is due to the fact that, in the considered implementation of recursive S-Chord, the last searched peer transmits back, to every previously contacted peer, some data about the outcome of storages or retrievals, while, in the considered implementation of iterative S-Chord, these data are available only to the peer that originated a storage or retrieval query. As a consequence, such an implementation could foster the diffusion of trust data across nodes.

5. Spartacus Attack in S-Chord

In this section, we study S-Chord in further detail, trying to understand its effectiveness and resilience under more complex attack scenarios. For this purpose, we consider only the recursive way of performing lookups, as it proved to be the best solution (see Section 4.2). Furthermore, we assess the effects of a network infected by *spartaci* against S-Chord and we try to provide some slight modifications in the proposed algorithms, to soothe the negative consequences of a Spartacus attacker. We focus solely on routing attacks coming from *spartaci* and particularly the already studied

cases of (i) no response or of (ii) random response, since the results for storage and retrieval operations are very similar. Moreover, the balancing factor corresponds with the optimum value we found, i.e., 0.68 for S-Chord routing, unless otherwise stated. The two main differences in the following analysis, compared to the previous ones, regard the presence of the temporal dimension on the abscissas of the graphs and the constancy of the number of peers in the system; i.e., we do not consider a variable amount of bogus peers. The reasons for these changes are the following: (i) first of all, because the effectiveness of a Spartacus attack varies in time, reasonably maximum in the first instants and then decreasing and, (ii) secondly, because the Spartacus attack provides the replacement of already valid IDs rather than the addition of new nodes. In order to properly assess the temporal dimension, in the simulation sets of this section the churn period of malicious nodes is no more random but it is established to 150 *vs*, and the observation interval is set to 50 *vs*, in order to achieve more granular observations. Obviously, in a real network malicious nodes could have different times in which they enter the network; however, the simplified assumption of a constant and common churn period is reasonable for an attacker characterized by bounded computational resources. Despite the churning, the average amount of bogus peers is constant during a simulation cycle.

5.1. The Effects of a Spartacus Attack onto S-Chord. In this subsection, we analyze the effects of a Spartacus attack directly onto the proposed S-Chord. We focus our attention on the performance degradation concerning both the number of successful lookups and the average hops per lookup. As we have already explained in advance, the analyses are performed in time, limiting to the first 2,000 *vs*, but the results are still an average over various simulation seeds to get a confidence of 90%. The limitation to the first 2,000 *vs* is due to the inherent nature of a Spartacus attack: this is more effective in its beginning and its effects fade afterward. In particular, we report only this time interval as in the following the curves do not experience other significant trends.

Concerning the first metric, we focus only on the case of no response, since this was the case of better improvement of S-Chord compared to standard Chord, for both 8,000 and 10,000 malicious nodes (see Figure 7). As it may be inferred from Figure 13, when we consider a Spartacus attack the performances decrease over time, something happening, by a lesser degree, also on the standard Chord algorithm. This may be caused by the fact that Chord, unlike other DHT, e.g., Kademlia, has no inherent proximity routing procedures that reinforce the whole algorithm during the joining phase (see Section 3.1.2). The performances of S-Chord dramatically drop, especially in the case of 10,000 *spartaci* and this is quite obvious because the good nodes have to face the same number of malicious counterparts. Particularly, the performances in case of 8,000 *spartaci* decrease of about 26%, while in the case of 10,000 *spartaci* they fall of about 36%. The worsening of the performances of S-Chord under a Spartacus attack may be motivated by the combined effect, caused by malicious nodes, of both inheriting high trust

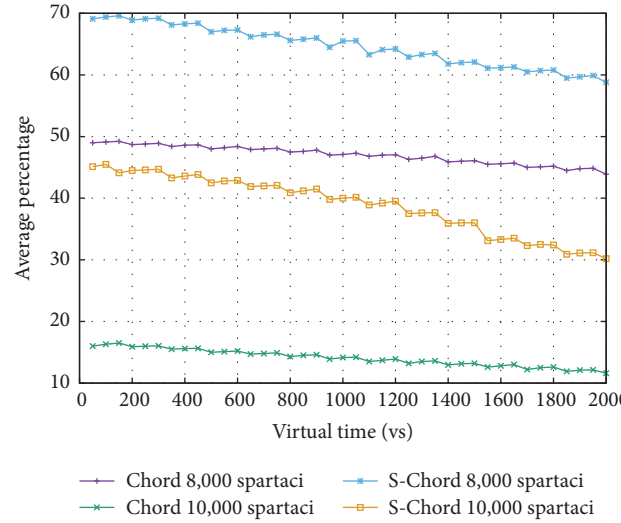

Figure 13: Successful lookups versus *sybils* number in standard Chord and S-Chord with a different quantity of *spartaci*.

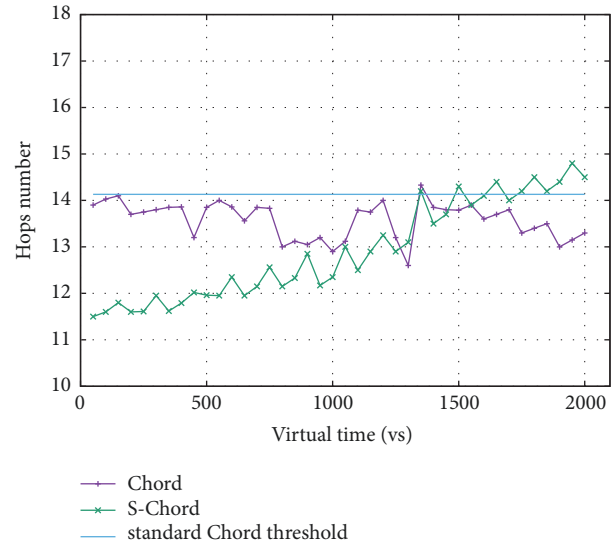

Figure 14: Average hops per lookup versus time in standard Chord and S-Chord with 8,000 *spartaci* in the network.

scores and replacing of good behaving nodes. That is, not only are malicious nodes trusted by good nodes in the beginning, but their presence also decreases the number of good behaving peers and, therefore, the overall performances drop accordingly.

What is to be highlighted is also the presence of a periodic trend in the curves, with period circa equivalent to the churning period of malicious nodes, i.e., 150 *vs*.

In Figure 14 we assess the effects of *spartaci* on Chord and S-Chord whenever the mean number of hops per lookup is concerned. In this circumstance, we consider only the case of 8,000 malicious nodes, since this is one of the two points of the graph in Figure 10 where our trust-based solution obtains better results than the classic algorithm. As the performances are very similar for both the no response and

the random response cases, we consider an average value and, as a consequence, only one combined malicious behavior. The graphs depicted in Figure 14 show an almost constant behavior for classic Chord, whereas the performances of S-Chord quickly degrade, overtaking the standard algorithm and even the $\log_2 N$ threshold ($\log_2(18,000) = 14.13$) by 1,500 *vs*. Furthermore, in this situation, the periodic tendency, following the churn of malicious nodes, appears again, especially in the curve of S-Chord.

5.2. Improving S-Chord. In this subsection, we propose some improvements to S-Chord to combat a Spartacus attack. We focus the improvement efforts mainly as regards the percentage of successful lookups, since, also in presence of *sybils*, the advantage of our balanced trust-based method, considering average hops, is visible only in some limited circumstances, namely, when malicious peers are limited to 6,000 or 8,000 (see Figure 10).

As a consequence, we vary S-Chord mainly using the following two countermeasures:

(i) the first one regards the opportune tuning of the weight W_{Rr} of the risk environment.

(ii) the second one concerns the utilization of an opportune time decay function to be applied to the trust score.

The first improvement we devised involves augmenting the weight of the network scenario. This seems a straightforward consequence since the Spartacus attack appears more dangerous than the Sybil-based one. However, we do not simply increase the environment weight to a higher value for those peers having already joined the network, since, following our first tests, this static countermeasure would have affected only the first moments of the simulation. Therefore, we make W_{Rr} vary according to a period equal to the one of the churn of the *spartaci*, increasing till 0.5 and decreasing to 0.3. This obviously implies the knowledge of such a period; however, it could be easily detected through a cooperation between those peers that are going to admit possible malicious nodes.

The second aforementioned countermeasure encompasses the introduction of a time decay function. More in detail, we multiply the trust score by a negative exponential function of time, with mean value corresponding to the churning period of the *spartaci*, i.e., 150 *vs*. This is better explained in (4), where $\widetilde{T}$ stands for new trust, T for old trust, and D_f for decay function.

$$\widetilde{T} = T \cdot D_f \tag{4}$$

The decay function is, instead, expressed by the following formula:

$$D_f(t) = \frac{1}{150vs} \cdot e^{-(1/150vs)t} \tag{5}$$

Therefore, also $\widetilde{T}$ in (4) becomes a function of time. $\widetilde{T}$ is going to replace T in the formulas of (2) and (1).

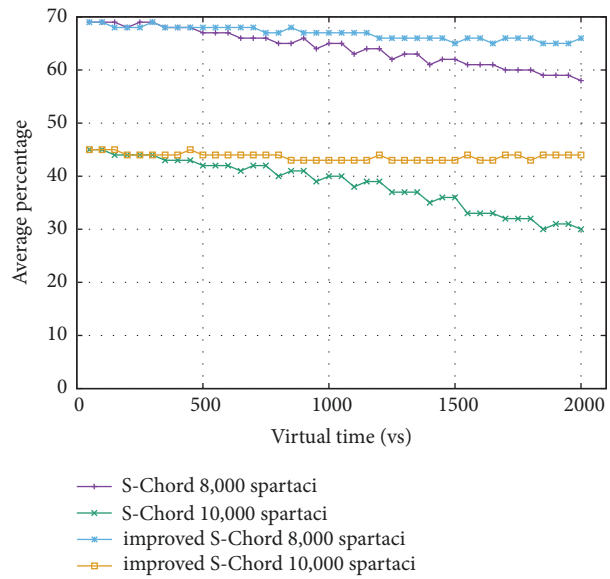

FIGURE 15: Successful lookups versus *sybils* number in S-Chord and enhanced S-Chord featuring different *spartaci* in the network.

What can be inferred, from Figure 15, is a sort of improvement in the performances of improved S-Chord: the curves, both for 8,000 and for 10,000 *spartaci*, maintain an overall constant trend. This does not take place with standard S-Chord. As a consequence, we could manage to get a stable behavior in time and thus soothe a little the drop in the performances. This is more evident in the case of 10,000 *spartaci* than in that of 8,000 malicious nodes; as a matter of fact, in the first case, the decrease of improved S-Chord is only about 1.89%, greatly better if compared with 33% of S-Chord, while in the second case the drop is more than three times, i.e., 4.35%, however, better than 15% of S-Chord. Moreover, the periodic trend, following the churning period of malicious nodes, tends to disappear both in the case of 8,000 and in the case of 10,000 *spartaci*.

5.3. Comparisons with Other Techniques. Finally, in this section, we assess the effectiveness of our proposal in comparison with other solutions, already known in the literature, under a Spartacus attack. We address the comparison between improved S-Chord, Koyanagi's solution [32], and GeTrust both in terms of successful lookups and in terms of average hops. The simulation conditions, as well as the utilized metrics, are the same as the ones of the previous subsections.

In Figure 16 we analyze the performances of S-Chord, GeTrust, and Koyanagi's solution with the ones of the improved version of S-Chord, in terms of successful lookups with a growing simulation time and a combination of malicious behaviors regarding the routing process (no response and random responses). The number of malicious nodes is 8,000. We can see that GeTrust performs more or less like S-Chord, especially for a growing simulation time, according to the analyses showed before in the paper, but its curve is always under the one of improved S-Chord, which tends to

Table 1: Comparison between different trust-based solutions in terms of decrease of successful lookups with a variable amount of *spartaci* in the network.

	S-Chord	improved S-Chord	GeTrust	Koyanagi's
8,000 spartaci	-14.93%	-4.49%	-15.77%	-26.96%
10,000 spartaci	-33.07%	-2.00%	-32.81%	-58.14%

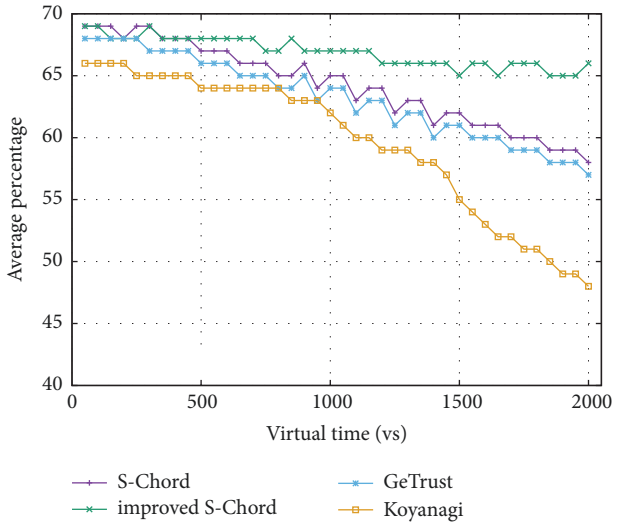

Figure 16: Successful lookups versus time in S-Chord, enhanced S-Chord, Getrust, and Koyanagi's solution with 8,000 *spartaci* in the network.

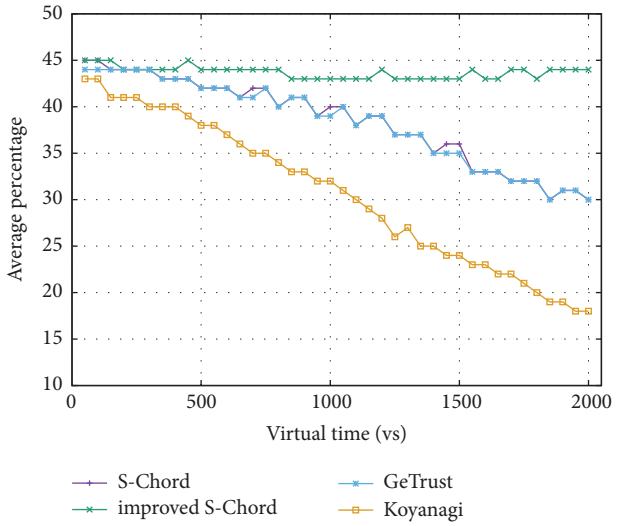

Figure 17: Successful lookups versus time in S-Chord, enhanced S-Chord, Getrust, and Koyanagi's solution with 10,000 *spartaci* in the network.

be almost constant. The decreasing performances of Getrust may be due to the chance that *spartaci* could assume the role of guarantors: this does not lead to a complete worsening of the overall performances, since the direct trust towards the guarantors decrease, but they, however, tend to follow the decreasing trend of S-Chord, where no guarantors are

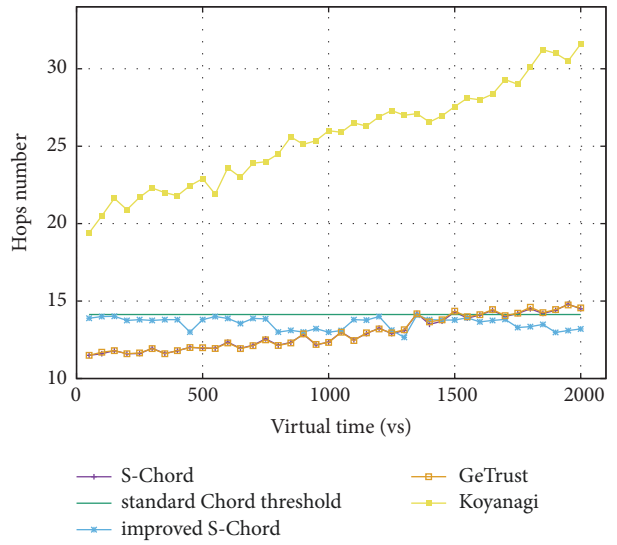

Figure 18: Average number of hops versus time in S-Chord, enhanced S-Chord, Getrust, and Koyanagi's solution with 8,000 *spartaci* in the network.

present. The worst performances are those of Koyanagi's solution. This may be due to its reliance on trust aggregation and propagation that may boost the collusion activity of *spartaci*.

In Figure 17 we show the performances of S-Chord, GeTrust, and Koyanagi's solution with the ones of the improved version of S-Chord, when bogus peers are 10,000 and considering successful lookups. The same conditions used for simulations of Figure 16 apply and we can draw more or less the same considerations. The curves of S-Chord and GeTrust seem to overlap much more and the decrease in the performances is more marked as reported in Table 1. From that table, it can be inferred that the improved S-Chord solution is better than S-Chord and than GeTrust and much better than Koyanagi's strategy.

In Figure 18 we show a comparison between the aforementioned solutions in terms of average hops per lookup and considering 8,000 *spartaci*, the most critical situation as seen previously. As we can see, Koyanagi's solution is the worst since it is based solely on trust and its propagation and aggregation and thus much more vulnerable to a Spartacus attack, even if its hop number is on average already the double of standard Chord by default [32].

In order to compare GeTrust and the other solutions effectively, we do not consider the messages exchanged with guarantor nodes and archive nodes, but only those used to actually perform the lookup. GeTrust performs similarly to

S-Chord, with a trivial growth for the hops according to the simulation time, while the improved version of S-Chord succeeds in maintaining an almost constant trend, similarly to standard Chord, and to remain under the $O(\log_2 N)$ threshold.

6. Conclusions

In this article a deep analysis of some trust-based countermeasures for Chord, under a Sybil or a Spartacus attack, has been presented. We have numerically studied the consequences of a Sybil attack in routing as well as in storage and retrieval operations, and we introduced a solution (namely, S-Chord), based on direct trust, to make Chord procedures more resilient. Moreover, we evaluated the still not deeply investigated Spartacus attack, both in Chord and in S-Chord, proposing some effective improvements to S-Chord itself. The results of our simulations are encouraging compared to standard Chord and to existing methods using exclusively trust, or other complex trust management systems. In conclusion, our approach may be regarded as a good candidate for a security solution applied to P2P networks. Using simple trust metrics is, as a matter of fact, far less power consuming than other cryptography-based solutions, opening its applicability to new emerging scenarios featuring low power devices, like the Internet of Things one. This is a matter for possible future research work, along with considering other DHTs, focusing on the application of trust to lookups and PUT and GET operations jointly, analyzing the spreading of trust information across the peers, or varying the optimum value of the balancing factor according to various scenarios.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

Dr. Riccardo Pecori and Dr. Luca Veltri declare that no conflicts of interest, regarding the publication of this paper, are present at the moment of submission.

Acknowledgments

Dr. Riccardo Pecori would like to thank Mr. Antonio Enrico Buonocore for carefully proof-reading the paper and polishing English and Stefano Marmani for making him discover the Spartacus attack.

References

[1] R. Pecori and L. Veltri, "3AKEP: Triple-authenticated key exchange protocol for peer-to-peer VoIP applications," *Computer Communications*, vol. 85, pp. 28–40, 2016.

[2] C. Liao, S. Cheng, and M. Domb, "On Designing Energy Efficient Wi-Fi P2P Connections for Internet of Things," in *Proceedings of the IEEE 85th Vehicular Technology Conference (VTC Spring '17)*, pp. 1–5, Sydney, NSW, June 2017.

[3] P. Maymounkov and D. Mazières, "Kademlia: a peer-to-peer information system based on the XOR metric," in *Peer-to-Peer Systems*, vol. 2429 of *Lecture Notes in Computer Science*, pp. 53–65, Springer-Verlag, 2002.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pp. 149–160, San Diego, Calif, USA, August 2001.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ser. SIGCOMM '01*, pp. 161–172, San Diego, California, USA, 2001.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware 2001*, vol. 2218 of *Lecture Notes in Computer Science*, pp. 329–350, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[7] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., vol. 2429 of *ser. Lecture Notes in Computer Science*, pp. 251–260, Springer Berlin Heidelberg, 2002.

[8] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta, "Limiting sybil attacks in structured P2P networks," in *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM '07)*, pp. 2596–2600, May 2007.

[9] M. Kohnen, "Analysis and optimization of routing trust values in a Kademlia-based distributed hash table in a malicious environment," in *Proceedings of the 2nd Baltic Congress on Future Internet Communications, BCFIC '12*, pp. 252–259, Lithuania, April 2012.

[10] R. Pecori, "S-Kademlia: A trust and reputation method to mitigate a Sybil attack in Kademlia," *Computer Networks*, vol. 94, pp. 205–218, 2016.

[11] R. Pecori and L. Veltri, "Trust-based routing for Kademlia in a sybil scenario," in *Proceedings of the 22nd International Conference on Software, Telecommunications and Computer Networks, SoftCOM '14*, pp. 279–283, Croatia, September 2014.

[12] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation," *Security and Communication Networks*, vol. 2018, pp. 1–30, 2018.

[13] S. Delgado-Segura, C. Pérez-Solà, J. Herrera-Joancomartí, G. Navarro-Arribas, and J. Borrell, "Cryptocurrency Networks: A New P2P Paradigm," *Mobile Information Systems*, vol. 2018, Article ID 2159082, 16 pages, 2018.

[14] C. Lesniewski-Laas and M. F. Kaashoek, "Whanau: A sybil-proof distributed hash table," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'10*, 2010.

[15] P. Mittal, M. Caesar, and N. Borisov, "X-vine: Secure and pseudonymous routing in dhts using social networks," in *Proceedings of the in 19th Annual Network and Distributed System Security Symposium, (NDSS '12)*, San Diego , California, USA, 2012.

[16] M. N. Al-Ameen and M. Wright, "Persea: A sybil-resistant social dht," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy, ACM*, p. 169, New York, NY, USA, Feburary 2013.

[17] M. N. Al-Ameen and M. Wright, "IPersea: Towards improving the Sybil-resilience of social DHT," *Journal of Network and Computer Applications*, vol. 71, pp. 1–10, 2016.

[18] Z. Yang, J. Xue, X. Yang, X. Wang, and Y. Dai, "VoteTrust: Leveraging friend invitation graph to defend against social network sybils," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 4, pp. 488–501, 2016.

[19] Z. Tan, X. Wang, and X. Wang, "A Novel Iterative and Dynamic Trust Computing Model for Large Scaled P2P Networks," *Mobile Information Systems*, vol. 2016, Article ID 3610157, 12 pages, 2016.

[20] L. Shi, J. Zhou, Q. Huang, and W. Yan, "A modification on the Chord finger table for improving search efficiency," in *Proceedings of the 13th IEEE/ACIS International Conference on Computer and Information Science, ICIS '14*, pp. 395–398, China, June 2014.

[21] P. Zave, "Reasoning About Identifier Spaces: How to Make Chord Correct," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1144–1156, 2017.

[22] C. T. Min and L. T. Ming, "Investigate SPRON Convergence Time Using Aggressive Chord and Aggressive AP-Chord," in *Proceedings of the 12th International Conference on Information Technology: New Generations, ITNG '15*, pp. 61–66, USA, April 2015.

[23] I. Woungang, F.-H. Tseng, Y.-H. Lin, L.-D. Chou, H.-C. Chao, and M. S. Obaidat, "MR-Chord: Improved Chord Lookup Performance in Structured Mobile P2P Networks," *IEEE Systems Journal*, vol. 9, no. 3, pp. 743–751, 2015.

[24] T. Amft and K. Graffi, "Moving peers in distributed, location-based peer-to-peer overlays," in *Proceedings of the International Conference on Computing, Networking and Communications, ICNC '17*, pp. 906–911, USA, January 2017.

[25] W. Zhang, B. Sun, and Y. Sun, "Trustchord: chord protocol based on the trust management mechanism," in *Proceedings of the International Conference on Advanced Intelligence and Awareness Internet (AIAI '10)*, pp. 64–67, Beijing, China.

[26] Z. Liang and W. Shi, "PET: A PErsonalized Trust Model with Reputation and Risk Evaluation for P2P Resource Sharing," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS '05*, pp. 201b–201b, Big Island, HI, USA.

[27] J. Wang and J. Liu, "The comparison of distributed P2P trust models based on quantitative parameters in the file downloading scenarios," *Journal of Electrical and Computer Engineering*, vol. 2016, Article ID 4361719, pp. 1–10, 2016.

[28] L. Mekouar, Y. Iraqi, and R. Boutaba, "Reputation-based trust management in peer-to-peer systems: Taxonomy and anatomy," in *Handbook of Peer-to-Peer Networking*, X. Shen, H. Yu, J. Buford, and M. Akon, Eds., pp. 689–732, Springer US, 2010.

[29] X. L. Xie, "Creditability assessment of dealers in P2P e-commerce," in *Proceedings of the 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, IMCEC '16*, pp. 1326–1333, China, October 2016.

[30] X. Ding and K. Koyanagi, "Study on trust-based maintenance of overlays in structured P2P systems," in *Proceedings of the International Conference on Computational Problem-Solving, ICCP '11*, pp. 598–603, China, October 2011.

[31] R. R. Rout and D. Talreja, "Trust-based decentralized service discovery in structured Peer-to-Peer networks," in *Proceedings of the 11th IEEE India Conference, INDICON '14*, India, December 2014.

[32] Y. Han, K. Koyanagi, T. Tsuchiya, T. Miyosawa, and H. Hirose, "A trust-based routing strategy in structured P2P overlay networks," in *Proceedings of the 27th International Conference on Information Networking, ICOIN '13*, pp. 77–82, Thailand, January 2013.

[33] X. Meng and D. Liu, "GeTrust: A Guarantee-Based Trust Model in Chord-Based P2P Networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 54–68, 2018.

[34] R. Pecori, "A comparison analysis of trust-adaptive approaches to deliver signed public keys in P2P systems," in *Proceedings of the 7th International Conference on New Technologies, Mobility and Security, NTMS '15*, pp. 1–5, France, July 2015.

[35] F. Tomonori, N. Yoshitaka, Y. Shiraishi, and T. Osamu, "An effective lookup strategy for recursive and iterative lookup on hierarchical dht," *International Journal of Informatics Society (IJIS)*, vol. 4, no. 3, pp. 143–152, 2012.

[36] M. Amoretti, M. Picone, F. Zanichelli, and G. Ferrari, "Simulating mobile and distributed systems with DEUS and ns-3," in *Proceedings of the 11th International Conference on High Performance Computing and Simulation, HPCS '13*, pp. 107–114, Finland, July 2013.

[37] J. Zhang, R. Zhang, J. Sun, Y. Zhang, and C. Zhang, "TrueTop: A Sybil-Resilient System for User Influence Measurement on Twitter," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2834–2846, 2016.

[38] M. S. Khan and N. M. Khan, "Low Complexity Signed Response Based Sybil Attack Detection Mechanism in Wireless Sensor Networks," *Journal of Sensors*, vol. 2016, pp. 1–9, 2016.

[39] M. Kohnen, "Applying trust and reputation mechanisms to a Kademlia-based Distributed Hash Table," in *Proceedings of the IEEE International Conference on Communications, ICC '12*, pp. 1036–1041, Canada, June 2012.

[40] A. Anjum, M. Sporny, and A. Sill, "Blockchain Standards for Compliance and Trust," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 84–90, 2017.