ID Mapping Attacks in P2P Networks

Davide Cerri[†] Alessandro Ghioni[†] Stefano Paraboschi[‡] Simone Tiraboschi[†]
† CEFRIEL - Politecnico di Milano, Via Fucini, 2, 20133 Milano — Italy
‡ DIGI - Università degli Studi di Bergamo, Via Marconi, 5, 24044 Dalmine BG — Italy
{cerri,ghioni,tiraboschi}@cefriel.it, parabosc@unibg.it

***Abstract*— Current P2P (overlay) networks have been designed with a focus on the construction of efficient mechanisms for the storage and retrieval of information among a community of participating nodes, with limited attention to the protection from malicious behaviours. The paper analyzes the attacks that can be realized on the choice of identifiers by the nodes and proposes adequate countermeasures. The paper uses as a reference the Kademlia protocol, but most of the results are immediately applicable to every DHT network.**

I. Introduction

P2P networks have in the past few years emerged as one of the most successful network applications, with tens of millions of users currently participating in them. They are based on the construction of an *overlay* (on top of IP) network, where a decentralized protocol permits to a community of (typically pseudo-anonymous) users to share resources. Current success in the user community is mostly due to the possibility to share resources that it would be difficult to distribute and retrieve using other channels (e.g., because resources are protected by copyright), but there are significant emerging legitimate uses of the same paradigm that fully motivate their investigation by the research community.

The paper focuses on the analysis of the security weaknesses that arise in many of these networks due to the unconstrained management of network identifiers and presents techniques that are able to solve these problems.

II. Distributed Hash Tables

A Distributed Hash Table (DHT) is a method to organize the storage of a hashtable among the nodes of a P2P network. The DHT techniques define ways to split a global indexing space (the *hashtable*) and to distribute every portion among its participants without any central authority coordinating this task. The most known DHT architectures are CAN [1], Chord [2], Pastry [3], Tapestry [4] and Kademlia [5]. Being a hashtable, a DHT makes it possible to store a ⟨*key, value*⟩ pair and extract the value pointed by a specific key; the ⟨*key, value*⟩ pair can be locally stored by one (or more) of the participating nodes. However, any of these nodes can enter or exit the P2P network without jeopardizing its functionality. Lookups for keys are performed by routing queries through a series of nodes; each of these nodes uses a local routing table to forward the query towards the node that is ultimately responsible for the key. Compared with other decentralized P2P architectures, DHT scales better as the number of participants grows, and this happens without sacrificing response accuracy. Given N system participants, DHT architectures are designed to complete a lookup operation with $O(\log N)$ messages.

A. Kademlia

In this paper we focus on Kademlia [5], because it offers a high degree of redundancy combined with adequate search performance. For this reason Kademlia is currently the object of considerable interest, both by the research and the application communities. In Kademlia, each node has a unique identifier (a 160 bit string) and can choose it freely, typically using a random function. Collisions in the identifiers are statistically improbable, in a 2^{160} elements space. Nodes and resources share the same indexing space, and a node is responsible for resources whose index is *near* to the node identifier. Distance between identifiers (and, therefore, between nodes and resources) is measured using a XOR metric: the result of a binary XOR between the node ID and the key ID represents the distance between the two, in binary positional representation; the greater the number of identical bits in the most significant digits, the lower the distance.

Given the nature of the XOR metric, Kademlia organizes its indexing space by representing it as a tree, whose leaves are Kademlia nodes. Each node enters the tree in a position determined by its ID: starting from the tree root each bit represents a branch in the binary tree. Every node has a partial scope of the whole tree, divided into a series of successively lower subtrees that do not contain the node. The highest subtree consists of half the binary tree not containing the node, the next subtree consists of half of the remaining tree not containing the node, and so forth. For each subtree each node defines a specific table, called *k-bucket*, and every k-bucket is structured to contain up to k triples ⟨*IP address, UDP port, Node ID*⟩ of nodes belonging to the related subtree. k is the redundancy factor that ensures a certain degree of robustness to the infrastructure; each node stores information for up to k nodes in each subtree. Nodes handle the entries in each k-bucket using a *"least-recently seen node at the head, most-recently seen at the tail"* policy.

When looking for the node pointed by a certain key, Kademlia uses a recursive algorithm. The lookup initiator starts by picking some nodes from its closest non-empty k-bucket, then sends them parallel and asynchronous requests whose response is a set of triples ⟨*IP address, UDP port, Node ID*⟩ pointing to nodes closer to the target. This operation is repeated by the lookup initiator in order to approach the target node, until its

This work was supported by the Italian FIRB MAIS project.

IP address and UDP port are discovered. Most operations are implemented in terms of the above lookup procedure; to store a ⟨*key, value*⟩ pair, a participant locates the k nodes closest to the key and calls on them the STORE command. Additionally, each node periodically re-publishes ⟨*key, value*⟩ pairs in order to keep them alive. Kademlia is designed to make ⟨*key, value*⟩ pairs expire after a given time period, so the publisher of a specific ⟨*key, value*⟩ pair has to periodically refresh the overlay network with this information, in order to avoid its expiration. Kademlia protocol messages are transported by UDP, and every node receiving a Kademlia message uses it to populate its routing table (i.e., to populate the corresponding k-bucket).

III. Security in DHT networks

DHT systems and specifically Kademlia are not designed to fully face misbehaving nodes' malicious attacks. Two main classes of security requirements can be defined for a generic P2P middleware: *overlay network security* (this means routing security and dependability of the DHT infrastructure) and *security for applications* that rely on a DHT structure (this means participants identification/authentication, resource authenticity and trust relationships). In [6] some security considerations about the former kind of requirements in DHTs are reported. As in [6], we consider the adversary as *a malicious node that can forge arbitrary IP packets with arbitrary contents, but is able only to examine packets addressed to itself*.

Overlay network attacks can be divided into routing attacks, storage and retrieval attacks and other miscellaneous attacks. Routing attacks can occur when a node behaves maliciously in the task of forwarding other nodes' lookups and when it spreads false routing messages. In order to avoid lookup routing attacks, the querier (1) should be able to check every step of the lookup operation and (2) should ensure that the destination is a correct termination point for the query (having a unique and verifiable address). In Kademlia, the querier iterates the lookup process controlling the full lookup procedure; moreover, at each step it can choose up to k nodes (the nodes in the relevant k-bucket), so that no single node can block the operation. On the other hand, no control exists on the identifiers declared by nodes, so the second part of the lookup attack must still be addressed; this attack can be stopped using a robust Kademlia identifier, as we will show later. Unsolicited messages can be used in order to poison victims' routing tables: Kademlia routing messages are transported by UDP, and no kind of handshake is required, so an attacker could poison its victims' tables by spoofing the IP source address in forged Kademlia routing messages; we also propose a solution to this forgery.

Storage and retrieval attacks occur, for instance, when a node participates in the lookup protocol correctly, but denies the existence of data it is responsible for. Replication can be used to solve this single-point-of-failure problem; Kademlia, which replicates ⟨*key, value*⟩ pairs in the k nodes closest to the key, is compliant with this requirement, as long as measures are taken that forbid nodes to collude on controlling the set of k identifiers closest to the key.

Miscellaneous attacks involve denial of service, churn and unsolicited messages. Since an adversary can generate packets, he can attempt to overload target nodes with garbage packets. This attack may be less effective if the infrastructure is redundant, and if routing identifiers are assigned randomly with respect to the physical network topology; since Kademlia does not specify how the identifier is chosen, this may reduce the effectiveness of the defense against denial-of-service attacks. In some DHT architectures, join or leave operations imply that newcomers must obtain and handle some data stored by other nodes, or remaining nodes have to obtain and handle data stored by leaving nodes. An attacker could destabilize the infrastructure using frequent join/leave operations, causing continuous data transfers between nodes, thus preventing the infrastructure to converge towards a new stable state. Kademlia is not affected by this attack, because it periodically updates resource information, and does not move any kind of resource when join or leave operations occur.

Therefore, Kademlia presents a set of features that make it able to be less affected by some security problems commonly found in DHT infrastructures, by virtue of redundancy, resource expiration and republishing mechanism, and iterative lookups (no message routing by intermediate nodes). But, the full realization of this potential is dependent on the realization of a robust ID generation mechanism, otherwise it is easy to mount significant attacks against its functionality, as we show in the next section.

IV. ID mapping attacks

In a DHT structure, a node may desire to obtain a particular identifier, i.e., a particular position on the overlay network, in order to gain control over certain resources. If a node could directly choose its own identifier, the attack would be trivial. On the other hand, we consider a completely distributed scenario, therefore solutions based on some sort of centralized authority which distributes and/or certifies identifiers are not acceptable. Sit and Morris [6] suggest using identifiers derived from public keys, so that identity claims can be verified. If each node autonomously generates its public key, and gets its identifier by applying a hash function to this public key, it is substantially forced to generate its identifier at random. But even if nodes must effectively generate their identifier at random, the "ID mapping" attack is still possible, given an adequate number of attempts to obtain the desired identifier.

If a node should get exactly a particular identifier, the attack would be infeasible in a 2^{160} ID space. But, given that the number of nodes in the system is much smaller, and therefore the ID space is largely empty, obtaining an ID that is sufficiently close to the target one is enough. In Kademlia each ⟨*key, value*⟩ pair is replicated on k nodes (the k ones with node ID closest to the resource ID), so the attacker must obtain more than one ID close to each target resource; but, as it has been made clear by Doucer in [7], in a (pseudo-) anonymous environment with no central authority, a user can easily create multiple identities and realize the so-called Sybil attack. Some sort of trust mechanism is necessary to solve this problem.

We identify three variants of the attack: complete, dictionary and partial.

a) Complete attack: In this attack, the attacker wants to gain total control over the target resource, i.e., he wants to impersonate all the k nodes which are responsible for the target resource. In order to do so, the attacker must obtain k valid identifiers which are closer than any other to the target.

Considering a system with N nodes randomly distributed in an ID space of M identifiers (ranging from 0 to $M-1$), we want to compute the success probability of a single try, i.e., the probability that a randomly chosen ID is the nearest one to the target resource. What “nearest” means depends on the particular DHT being considered; here we consider it to be the lowest identifier greater than the target one. As we assume that nodes and resources are uniformly distributed over the ID space, the success probability is the same for any resource, so we choose as target the resource with ID equal to 0. Under these assumptions, we can say that the success probability of a single try is equal to the probability that a randomly chosen identifier is lower than the lowest existing identifier.

Let Y be the discrete random variable representing the ID resulting from the try; Y has a uniform distribution, i.e. ($0 \le y \le M-1$):

$$p(y) = \frac{1}{M}$$

Let X be the discrete random variable representing the lowest identifier among the N existing nodes; the probability that X is greater than or equal to a value x ($0 \le x \le M-1$) is:

$$P(X \ge x) = \left(1 - \frac{x}{M}\right)^N$$

So, the X distribution (for $0 \le x \le M-1$) is:

$$p(x) = \left(1 - \frac{x}{M}\right)^N - \left(1 - \frac{x+1}{M}\right)^N \quad (1)$$

The success probability of the single try is then the probability that $Y < X$, which, being Y and X independent, can be expressed as:

$$\begin{aligned} p_s &= \sum_{x=0}^{M-1} \sum_{y=0}^{x-1} \frac{1}{M} \left[\left(1 - \frac{x}{M}\right)^N - \left(1 - \frac{x+1}{M}\right)^N \right] \\ &= \frac{1}{M} \sum_{x=0}^{M-1} \left(\frac{x}{M}\right)^N \end{aligned} \quad (2)$$

This is hard to compute, but for large values of M (and in our case M must be a very large number, e.g., 2^{160}) it can be seen that p_s is in practice independent of M, and in particular:

$$p_s \approx \frac{1}{N+1} \quad (3)$$

This can be understood by considering a continuous, rather than discrete, ID space. For large values of M, in fact, the probability of picking up a particular ID is in practice 0, and we only consider the probability of getting an ID *near* the target one, which is consistent with the continuity approximation. Considering a continuous ID space, X cumulative distribution and density (for $0 \le x \le M$) are respectively:

$$F(x) = P(X \le x) = 1 - \left(1 - \frac{x}{M}\right)^N$$

$$f(x) = \frac{d}{dx} F(x) = \frac{N}{M} \left(1 - \frac{x}{M}\right)^{N-1}$$

Therefore, in a continuous ID space the success probability is:

$$p_s = \int_0^M \frac{1}{M} \, dy \int_y^M \frac{N}{M} \left(1 - \frac{x}{M}\right)^{N-1} dx = \frac{1}{N+1}$$

So, the success probability of a single try does not depend on the ID space width M, but only on the number of existing nodes N.

Now, let T' be the discrete random variable representing the number of tries needed to have the first success; T' follows a geometric distribution, so its expected value is:

$$E[T'] = \frac{1}{p_s} \approx N + 1$$

Taking redundancy into account, in order to gain total control over the target resource the attacker must obtain k identifiers in the same interval, that is k successes. The discrete random variable T representing the number of tries needed to have the first k successes follows a negative binomial distribution, so its expected value is:

$$E[T] = \frac{k}{p_s} \approx k(N+1) \quad (4)$$

So, the complexity of this attack depends only on the number of nodes the DHT has, and grows linearly with the redundancy factor: redundancy, which makes the system much more robust against denial-of-service and churn, does weakly protect the infrastructure from this malicious attack.

b) Dictionary attack: Up to now we described the attack to a single resource, but the same result can be extended to an attack to the whole indexing space. In the complete attack, the attacker, when targeting a single resource, runs an algorithm that extracts a random identifier and keeps it if close enough to the target index, discarding it otherwise. Every identifier obtained by the attacker is the result of a random try having uniform density. If the attacker has enough storing space, he can keep all the identifiers (and keys) in order to build a dictionary of the indexing space.

The number of tries needed to build such a dictionary is on the average the same as for the complete attack. In fact, the number of successes obtained in t tries (each one being a Bernoulli trial) follows a binomial distribution, and its expected value is tp_s. This means that, with $k(N+1)$ random extractions, the attacker can “sample” the whole indexing space so thickly that he can gain control over every resource just by picking values from its stored dictionary, because on the average he will find k successes independently of the particular resource being considered as target. We observe that the precomputation and indexed storage of the dictionary, for reasonably large networks, is well within the capabilities of an inexpensive computing system.

c) Partial attack: In this attack, the attacker does not want to control all the k nodes responsible for the target resource, he just aims at impersonating a fraction of the nodes deputed to control a resource. We define c as the fraction of responsible nodes the attacker wants to impersonate, and $r = (1-c)k$ as the maximum number of residual nodes he admits within the responsible set. So, under the same assumptions made for the complete attack (and considering again the resource with ID equal to 0 as target), we can say that the success probability of a single try is the probability that, given a randomly chosen identifier, there are no more than r existing nodes with a lower identifier.

Let X be the discrete random variable representing the existing identifier such that there are r nodes with a lower identifier; the probability that X is greater than a value x $(0 \leq x \leq M-1)$ is:

$$P(X \geq x) = \sum_{i=0}^{r} \binom{N}{i} \left(\frac{x}{M}\right)^i \left(1 - \frac{x}{M}\right)^{N-i}$$

So, the X distribution (for $0 \leq x \leq M-1$) is:

$$p(x) = \sum_{i=0}^{r} \binom{N}{i} \left[\left(\frac{x}{M}\right)^i \left(1-\frac{x}{M}\right)^{N-i} + - \left(\frac{x+1}{M}\right)^i \left(1 - \frac{x+1}{M}\right)^{N-i}\right] \quad (5)$$

The above equation is the generalization of (1) when a maximum number of r nodes are within the attacked ID interval; in fact (1) can be obtained from (5) setting $r = 0$.

As in the complete attack, the success probability of the single try is the probability that $Y < X$ (Y is uniformly distributed), which, being Y and X independent, can be expressed as:

$$\begin{aligned} p_s &= \sum_{x=0}^{M-1} \sum_{y=0}^{x-1} \frac{1}{M} p(x) \\ &= \frac{1}{M} \sum_{i=0}^{r} \binom{N}{i} \sum_{x=0}^{M-1} \left(\frac{x}{M}\right)^{N-i} \left(1 - \frac{x}{M}\right)^i \end{aligned} \quad (6)$$

Again, the above equation is a generalization of (2), which can be obtained setting $r = 0$. The exact value of p_s is hard to compute, but for large values of M it can be seen that it is in practice independent of M, and in particular:

$$p_s \approx \frac{r+1}{N+1} \quad (7)$$

In comparison with (3), the above equation increases the success probability of a single try of an $r+1$ factor. This can be intuitively explained as follows: the N existing nodes and the one added by the attacker partition the ID space in $N+1$ intervals. For the attack to be successful, the target resource can be in $r+1$ intervals, that is the one for which the closest node is the attacker plus the r for which the closest node is one of the "admitted" ones. In other words, whereas in the complete attack there is only one (out of $N+1$) "good" interval for the attacker, in this case the "good" intervals are $r+1$, therefore the success probability of the single try is $r+1$ times the one found for the complete attack.

As in the complete attack, the discrete random variable T representing the number of tries needed to have the desired number of successes follows a negative binomial distribution, so its expected value in order to get $k-r$ successes is:

$$E[T] = \frac{k-r}{p_s} \approx \frac{k-r}{r+1}(N+1)$$

That is, if we express r by means of the "control factor" c:

$$E[T] \approx \frac{kc}{k(1-c)+1}(N+1) \quad (8)$$

Equation (8) is equal to (4) if $c = 1$ (i.e., complete control). This attack is substantially easier than the complete one even for significant values of c. For example, if $k = 20$ (as suggested in [5]), in order to gain a control factor of 0.75 the attacker should make an average of $\frac{15}{6}(N+1)$ tries, instead of the $20(N+1)$ required for the complete attack. Therefore, this attack is 8 times easier than the complete one. If we consider $c = 0.5$ (i.e., the attacker wants to impersonate half of the responsible nodes), for $k = 20$ the attack is 22 times easier than the complete one. For a fixed value of c, increasing k makes the partial attack easier compared to the complete attack.

V. Constrained ID selection

If each node can autonomously choose its own identifier, either directly or indirectly (i.e., randomly as we studied above), the system is vulnerable to the ID mapping attack. The solution is to introduce an identifier that depends on some piece of information outside of the control of the node. A possible solution could use a certification authority, which each node would have to contact to obtain an identifier, for instance when it initially connects to the network. We already presented this as an inadequate solution since it would introduce a single point of control, which would be against the egalitarian and open principles that are at the foundation of the P2P paradigm. A possible compromise could require the use of a family of certification authorities, among which the user has freedom to choose, in order to avoid the centralization aspect; nonetheless such an approach would introduce considerable maintenance issues and a significant burden in terms of cryptographic and key management functions.

Another solution consists in using a *constrained ID selection mechanism*, requiring the node to derive its identifier from its IP address and port number by means of a hash function. Each node can autonomously define its identifier without asking a central authority, but it cannot choose a particular identifier or indefinitely ask for a new one until it gets a desired value; the port number can be freely chosen by the user, but the number of possible identifiers is limited, because we consider in the hash function only a limited subset of the bits identifying the port number (if a conflict arises due to the fact that a participant with the same IP address of another chooses a port characterized by the same bits, we assume that the new node simply detects the collision and selects another port).

An identifier derived from the IP address and port number offers the advantage that it is *immediately verifiable*: a node claiming a certain identifier, which corresponds to a certain IP-port address, can be challenged by sending a nonce to that IP-port address and asking for a reply. In Kademlia, routing tables are updated using a conservative policy: old contacts, if still alive, are never removed in favor of new ones[1]; this makes it more difficult to poison other nodes' routing tables, because "good" contacts are never removed. Anyway, if the appropriate k-bucket is not full, the mere receipt of any message from another node causes the insertion of that node (i.e., its IP address, port number, and node identifier) into the recipient's routing table; as Kademlia messages use UDP transport, spoofing messages in order to insert fake data into other nodes' routing tables is simple. Moreover, nodes give their contacts to other nodes in response to queries, so fake entries can spread through the network. To avoid this, we require that every node, before inserting a ⟨*IP address, UDP port, Node ID*⟩ triple in its routing table, checks its validity by performing the following operations: first, it hashes the IP-port address and checks that the result corresponds to the node ID; second, it sends a Kademlia PING command to that IP address and port with a random nonce. Only if the node receives a reply with the correct nonce it inserts the new contact into its routing table. In this way, the node knows that there really is an alive node at that IP address and port (since it was able to answer the Kademlia PING), and that the node is the legitimate owner of the identifier. In other words, the node that inserts a contact in its table can have a good degree of assurance about it, and this is achieved with a lightweight mechanism (it does not involve heavy cryptographic operations and does not require a certifying entity).

The main reason we combine IP address and port number in the hash computation is because hashing only the IP address makes it impossible for nodes behind a NAT to take part in the system. We concatenate the 32 bit IP address and the last n bits of the 16 bit port number before hashing (e.g., $n = 6$). It is true that this allows an attacker to choose between 2^n possible identifiers for each IP address he controls; but if we assume that $| IP_M | \cdot 2^n \ll N$ (where $| IP_M |$ is the number of IP addresses controlled by a malicious user and N is the number of nodes in the network), the analysis presented above tells us that this is not sufficient for a malicious user to gain control of a specific resource in the network.

VI. ID Rotation

The use of a constrained ID selection mechanism, as presented above, strongly increases the difficulty of obtaining an ID near a desired objective in the DHT indexing. But, a determined attacker is still able to evaluate which are the points in the network that are able to produce a given identifier and he may expend resources to control them. E.g., he may try to acquire one or more IP addresses able to produce IDs "near" a target resource, in order to gain a certain degree of control over that resource.

Another consequence of the use of constrained IDs is that they may reduce the dynamicity of the network and may lead to persistent situations where a node with limited resources, which is accidentally given control of a critical ID in the network, may become a bottleneck that cannot be easily removed.

Since the source of the problem is the immutability of the relationship between the network and the resource identifier, we introduce variability in their binding. We briefly present two strategies: variability in the resource identifier and variability in the node identifier. In both contexts the approach uses the same principles. We first present the solution for the resource identifier.

Each resource has a *permanent* identifier ID_p, which is the usual identifier, but also a *temporary* identifier ID_t, computed by hashing ID_p along with a piece of temporal information. A node looking for a certain resource takes the resource permanent identifier and the current timestamp, computes the current temporary identifier and searches for it on the network. In this way, the association between resources and responsible nodes is temporary.

In order to have variable resource identifiers, there must be a mechanism that periodically "moves" resources from the old temporary identifier to the new temporary identifier. In general, this can be a serious problem, because moving resources can impose a heavy load on the overlay network; moreover, if all the resources were moved at the same time the network could crash. But in Kademlia, unlike other DHTs, there is no mechanism for moving resources when a node joins or leaves the network: nodes periodically refresh the ⟨*key, value*⟩ pairs they want to publish. Using the same refresh mechanism, we can achieve resource identifiers variability: nodes will simply publish resources using always the current temporary identifier. If the refresh time is one hour and temporary identifier validity time is one week (168 hours), a node will publish a particular resource using a certain identifier for 168 times, and then, at the first publication that occurs after the identifier change instant, it will start using the next identifier. This introduces a transitory period in which a resource can be present both with the old and the new identifier: after the identifier change instant, and until a complete refresh has occurred, "old" publications made before the change coexist with "new" publications made after the change. During this period, in order to get all results, searches must be duplicated: a resource must be looked up using both the old and the new identifier, and the two result sets must be merged. Assuming that nodes can autonomously decide when resources should be refreshed, such a period lasts a refresh time and occurs every temporary identifier lifetime, that is, with the above values, one hour a week (about 0.6% of total time).

The mechanism we propose also minimizes the impact of

[1]If the appropriate k-bucket is full, the *new* contact is discarded.

double searches: instead of making all resources expire at the same time, we uniformly distribute temporary ID expiration events. If the refresh time is T_r and the temporary identifier validity time is $T_v = nT_r$ (e.g., $T_r = 1$ hour, $T_v = 1$ week, $n = 168$), a node can compute $ID_p \mod n$ to define in which of the n refresh intervals the temporary identifier change for that resource will occur. This means that in each refresh interval $1/n$ of all resources will change temporary identifier, so $1/n$ of all searches will need to be duplicated; this implies, with the above values, an average increase in the lookup traffic of about 0.6%, which is negligible.

It is worth noting that there is no need for strict clock synchronization among nodes: the level of synchronization required is easily realizable using services already available in most computing systems. If a node has a considerable clock skew, it simply will not be able to find resources in the network, and other nodes will not be able to find the resources it publishes, but its presence will not damage the network.

The alternative that uses a variable node identifier works in the same way, except that it is the network identifier of the node that is built starting from a permanent ID derived from IP address and port number, which is then combined with a piece of temporal information that guarantees a degree of variability. With this solution, the temporal information is used only when a node must determine its identifier, while it does not affect lookup operations[2]. For this reason, unlike resource ID rotation, node ID rotation could enable the coexistence between nodes that rotate their ID and nodes that keep their permanent implicitly assigned ID; anyway, since the cost of the introduction of ID variability is small compared with the advantages, all the nodes should support the rotation service.

In terms of cost, both resource and node ID rotation introduce extra operations, needed to “move” resources or nodes, but their impact can be made arbitrarily small with an adequate choice of $n = T_v/T_r$. A comparison between resource and node ID rotation shows as more critical node ID rotation, because in DHT networks each node knows in detail its “neighbourhood”. Forcing nodes to change their ID, and thus their position in the overlay network, means that they must periodically acquaint themselves with their new neighbourhood (i.e., rebuild their routing tables), which has an impact on network stability and performance.

For all of the above considerations, in Kademlia resource ID rotation is certainly more appropriate, because it exploits the refresh mechanism, which does not require to explicitly move resources, whereas node ID rotation would require changes in nodes' k-buckets.

VII. Conclusions

We analyzed the security issues that arise in a DHT due to the free choice of node identifiers and proposed a solution. This work contributes to a research area that has recently received a significant interest, as testified by [8], [9]. Compared with previous work, our analysis combines a rigorous mathematical treatment with the proposal of concrete approaches that exploit the features of a modern DHT solution like Kademlia, in order to increase its robustness. The combined use of constrained identifier selection and a rotation mechanism promise to be a significant improvement to the security of current solutions.

Since in our solution the node ID depends on the peculiar characteristics of its connection to the network, the ID becomes variable in a way outside of user control and it cannot be used as a basis for the construction of reputation management services, which are emerging as an important opportunity for the construction of trust in the open and decentralized environment of P2P networks [10]. This motivates the definition of a second node ID, stable and independent from the network connection. This ID should be probably a hash of a public key autonomously computed by the node, in order to permit the node to self certify its identity, with an approach similar to [6], [10]. We envision that the decoupling between the dynamic node ID used for routing and the stable ID used to support its identity will characterize all the future DHT networks.

[2]However, since this technique is combined with ID verification related to constrained identifiers, temporal information must be used in verifying other nodes' IDs, so it affects routing operations.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, “A scalable content-addressable network.” in *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001, pp. 161–172.

[2] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A scalable Peer-To-Peer lookup service for Internet applications,” in *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001, pp. 149–160.

[3] A. I. T. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems.” in *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16*, ser. Lecture Notes in Computer Science, R. Guerraoui, Ed., vol. 2218, 2001, pp. 329–350.

[4] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, “Tapestry: An infrastructure for fault-tolerant wide-area location and routing,” UC Berkeley, Tech. Rep. UCB/CSD-01-1141, Apr. 2001.

[5] P. Maymounkov and D. Mazières, “Kademlia: A peer-to-peer information system based on the XOR metric.” in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8*, ser. Lecture Notes in Computer Science, vol. 2429. Springer, 2002, pp. 53–65.

[6] E. Sit and R. Morris, “Security considerations for peer-to-peer distributed hash tables.” in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8*, ser. Lecture Notes in Computer Science, vol. 2429. Springer, 2002, pp. 261–269.

[7] J. R. Douceur, “The Sybil attack.” in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8*, ser. Lecture Notes in Computer Science, vol. 2429. Springer, 2002, pp. 251–260.

[8] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach, “Secure routing for structured peer-to-peer overlay networks.” in *5th Symposium on Operating System Design and Implementation (OSDI 2002), December 9-11, Boston, Massachusetts, USA. USENIX Association*, 2002.

[9] M. Srivatsa and L. Liu, “Vulnerabilities and security threats in structured peer-to-peer systems: A quantitative analysis,” in *20th Annual Computer Security Applications Conference (ACSAC 2004), December 5-9, Tucson, AZ, USA*. IEEE Computer Society, 2004.

[10] F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, “Choosing reputable servents in a P2P network,” in *Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.