Authentication in Stealth Distributed Hash Tables

Andrew MacQuire Andrew Brampton Idris A. Rai Nicholas J. P. Race Laurent Mathy
Computing Department
Lancaster University
{macquire,brampton,rai,race,laurent}@comp.lancs.ac.uk

Abstract

Most existing DHT algorithms assume that all nodes have equal capabilities. This assumption has previously been shown to be untrue in real deployments, where the heterogeneity of nodes can actually have a detrimental effect upon performance. In this paper, we acknowledge that nodes on the same overlay may also differ in terms of their trustworthiness. However, implementing and enforcing security policies in a network where all nodes are treated equally is a non-trivial task. We therefore extend our previous work on Stealth DHTs to consider the differentiation of nodes based on their trustworthiness rather than their capabilities alone.

1. Introduction

Distributed Hash Tables (DHTs) have been shown to be a useful form of decentralised, structured peer-to-peer overlay [17][22][23][15]. They allow for the provision of simple hash table functionality – that is, the ability to *put* and *get* pieces of data indexed via hash codes – across multiple nodes in a scalable and resilient fashion. Primarily, DHTs have been used as location substrates for many varied applications, such as large-scale file storage [8][6][9] and multicast [18][25], amongst others.

Theoretically, most DHTs consist of numerous nodes which organise themselves and behave in a well defined manner. Each node is associated with a unique identifier (ID), randomly selected from a large, sparsely-populated address space. When an object is *put* into a DHT, its contents are hashed in some way as to produce an identifier which also maps into this address space. The originating node then routes the object to the local node that it knows to have the closest-matching identifier. If the recipient knows of an even closer node, it then forwards the data on. Eventually, the object reaches a node which does not know of any closer match. This node is considered the final destination, and must then take responsibility for the storage and/or any other handling of the object. Any individual who then wishes to *get* the same object at a later date can then do so based on the knowledge of the object's hash alone, as any message routed to the same hash should arrive at the same node that the object originally reached. Of course, this can only occur if all nodes follow the DHT protocol correctly.

Unfortunately, in a real-world deployment it would be naïve to assume that all nodes can be relied upon to conform to any prescribed behaviour. Without appropriate security policies in place, numerous problems may exist for public DHTs. For instance, a malicious node may examine, alter or deliberately drop messages passed through it (*i.e.* a sniffing, man-in-the-middle or denial of service attack). The ability to inject unsolicited messages into the DHT, or to alter those in transit, also allows untrustworthy nodes to corrupt the routing tables of others. Possible consequences of such actions could be legitimate nodes being denied service, or malicious nodes improving their standing on the network at the expense of others. In terms of content in the DHT, if all nodes are allowed to perform *put* operations, then the pollution of the system with unwanted or illegal data may become an issue.

To avoid these security problems, it is clear that some methods for handling untrustworthy nodes in DHTs is required, especially if DHTs are to be used commercially. Sadly, most traditional DHTs make the assumption of homogeneity amongst peers, treating them all as equals. This means that untrustworthy nodes are regrettably granted exactly the same privileges and responsibilities as the trustworthy. A common approach to solving this problem is to make use of an authentication mechanism to ensure that only trustworthy nodes are allowed to join the DHT. However, determining the veracity of a node can be a difficult process, and simply alienating all those who cannot prove themselves trustworthy may be unwise. A better approach is to instead allow such nodes to still make use of the DHT, but in a limited capacity.

To this end, a means of separating sets of nodes on the same DHT proves necessary. In this paper, we intend to show how our recently proposed Stealth DHT concept [3]

can be used to provide this precise functionality. In this previous work, we explained how almost any existing DHT algorithm could be adapted into a Stealth DHT, thus creating two distinct sets of nodes with differing routing properties on the same overlay. Our original evaluation covered the benefits of this approach from a performance standpoint, whereas this paper now considers the advantages gained in terms of security.

The separation of nodes in Stealth DHTs can be exploited to enable secure content distribution in peer-to-peer networks, since, with an appropriate authentication mechanism, it can return network control to the service provider. That is, the ability to create fine-grained permissions for nodes on the DHT means that security policies are much easier to enforce. For example, by having only authorised nodes storing and retrieving content, Stealth DHTs can ensure that only legitimate and useful content is served, thus a providing a platform for Digital Rights Management (DRM) in peer-to-peer networks, as well as aiding in the prevention of pollution attacks.

The remainder of this paper is structured as follows: Section 2 gives a brief overview of the differences between traditional and Stealth DHTs. Section 3 discusses the structure of a typical Public Key Infrastructure. Section 4 then explains how a Stealth DHT and a PKI may interoperate. Section 5 then highlights and evaluates a number of implementation concerns for such a system. Section 6 goes on to examine related work in the field of DHT authentication, and finally Section 7 concludes the paper.

2. Overview of a Stealth DHT

The join process for most DHT implementations involves a node first gathering state. Usually, this is achieved by routing a join message addressed to its own ID into the DHT via a bootstrap node[1]. Nodes along the message's path then reply directly with relevant routing information for the joining node. Once the joining node receives notification that its message has reached its destination, it announces its presence on the network so that other nodes may route messages through it.

Stealth DHTs [3] modify this procedure slightly to create two types of nodes on the network: *Service* and *Stealth*. Service nodes provide the routing infrastructure for the overlay, whereas stealth nodes communicate with and through service nodes only. This separation is achieved by halting the join procedure for stealth nodes after they have gathered state, but before they announce their presence on the DHT. The resultant effect is that stealth nodes do not appear in any routing tables, and thus are not used to forward any messages or store any keys. Therefore, they are incapable of interfering with message delivery or object storage in any direct manner. The routing data gathered by stealth nodes is used only for selecting the locally-optimal location to forward their own requests to, thus aiding routing performance while removing the possibility of a single point of failure that many similar DHT super-peer schemes suffer from [12].

It is important to note that the assignment of the stealth and service node roles is application-dependent, and is not prescribed or constrained by the Stealth DHT itself [3]. However, a Stealth DHT provides an individual or a service provider with the control to command such assignment. Therefore, from a security perspective, and since service nodes are responsible for handling all messages, they should consist of verifiably trustworthy machines. Conversely, any nodes which are potentially untrustworthy should be forced to join as stealth nodes, thus prohibiting them from interfering with DHT operations to any extent.

In contrast with traditional DHTs, the distinction between trustworthy and untrustworthy nodes provided by a Stealth DHT results in an architecture where the implementation of security policies is more straightforward. In addition, the fact that service nodes never retain any knowledge of stealth nodes means that Stealth DHTs are poised to offer significant advantages from a security perspective.

For example, when a stealth node joins or leaves the network, no service nodes need to update their routing tables, which prevents them from being affected by stealth nodes churning. This is especially important from a security standpoint, as the effects of heavy churn have been identified as being particularly harmful to many DHT implementations [16][10]. Malicious individuals have accordingly used this knowledge as a means of facilitating Distributed Denial of Service (DDoS) attacks. By making numerous nodes under their control rapidly rejoin DHTs, an attacker can cause floods of maintenance messages as nodes struggle to keep their routing tables up to date, resulting in inconsistent routing and overloaded nodes. If, however, a Stealth DHT is used where the potentially untrustworthy are forced to join as stealth nodes, no routing table updates and thus no maintenance messages are required. Of course, the inevitable cost of such an advantage comes in the form of increased stress upon the service nodes, although these are assumed to be relatively powerful machines capable of handling such load.

Of course, in a system such as this there still must be a means of ensuring that stealth nodes cannot masquerade as service nodes. This can be achieved through the use of an appropriate authentication scheme to effectively enforce the separation between node types in a Stealth DHT. Accordingly, this is the focus of this paper, wherein we discuss how an authentication scheme based on a Public Key

[1] An already-connected node discovered through some alternate mechanism

Infrastructure can be implemented in a Stealth DHT.

3. Overview of a Public Key Infrastructure

A Public Key Infrastructure (PKI) is a security platform which allows multiple users who have not previously exchanged any secret information to validate each other's identities, be sure of message integrity and even set up confidential communication. This is usually achieved via digital certification signed by mutually trusted third parties, where the certificates themselves are used to verify the identity of their owner through public/private key cryptography.

A typical PKI is composed of several logically separate entities, although the functionality offered by each may, in fact, be contained within a single physical machine:

A **Registration Authority** (RA) is a trusted entity which acts as the first point of contact for an individual requesting certification. The RA is used to check the requestor's supplied credentials and, if deemed valid, pass them on to the Certification Authority.

A **Certification Authority** (CA) is a trusted entity responsible for the creation and, if supported, revocation of certificates. As it is a mutually trusted third party, individuals may authenticate each other with confidence if they sign their messages using a certificate verifiably issued by a CA.

A **Certificate Repository** (CR) simply acts as a database of existing certificates. A CR need not be a trusted entity as the certificates it holds are immutable; if any attempt is made to alter an existing certificate, the digital signature will no longer match the contents. Note that if nodes are made responsible for the storage and dissemination of their own certificates, a dedicated CR may be redundant.

If supported, the CR also contains the **Certificate Revocation List** (CRL), indicating which certificates in the database have been forcibly revoked. Certificate revocation, however, is often an unsupported feature in actual PKIs due to implementation difficulties; an issue discussed further in Section 5.4 of this paper.

Many PKI implementations use certificates which conform to the ITU-T X.509 standard [1], a simplified version of which can be seen in Fig. 1. For compatibility reasons, certificates must contain a version record. To aid certificate management, unique serial numbers are also considered useful. Each certificate should also have two discrete dates associated with it, indicating its period of validity.

Version
Serial Number
Validity Duration
Subject - Identity - Public Key
Issued By
Issuer's Signature
Extensions - Subject Permissions

Figure 1. Format

The most important element of any digital certificate, however, is the subject. That is, the individual or organisation whose identity the certificate may be used to authenticate. Typically this information is comprised of the owner's name and any other pertinent information (address, organisation name *etc.*). The owner's public key is also usually included in this section of the certificate. This key is cryptographically paired with a corresponding private key that each individual must keep secret, as it serves as their means of creating a digital signature, as well as decrypting any messages encrypted with their public key.

Finally, the certificate must indicate the authority which issued it, and must also contain the signature of that authority. The key point here is that once the certificate has been signed in this manner, the data contained within cannot change without invalidating the signature. Note that certificates may also contain optional extension fields, which may be used, for instance, to indicate the operations an individual is authorised to carry out within a given system.

It is also notable that as any individual who owns a signed certificate from a higher authority may also sign certificates themselves, lengthy certification hierarchies often exist. As requesting and verifying each level of certification separately may prove to be time-consuming, many PKIs allow for the *chaining* of certificates. That is, all certificates up to the initial, self-signed certificate (created by some intrinsically trustworthy entity) are included in a single collection. While such a certificate-chain will obviously be larger than any single certificate, it intuitively reduces overhead if verification up to the highest level is known to be required.

4. Authentication in a Stealth DHT

A straightforward approach to implementing a PKI on a Stealth DHT is to require each service node to be issued with a certificate that it alone possesses the private key for. However, stealth nodes are required to possess certificates only when they need to perform optionally restricted operations, such as the ability to join the DHT itself, or to get/put content of particular types.

Beyond the standard identifying fields (see Fig. 1), each certificate should contain a list of authorised operations that its owner may carry out. Simple examples of such permissions could be the right to join as a service node, or to put keys into the network. It would then be mandatory for the relevant DHT messages to contain a field which identifies the node's certificate in some way, as well as a digital signature to prove that the same node was indeed the message's creator. The node's certificate or certificate-chain could also be attached to the message to aid in the authentication process.

4.1. Authenticating With a Stealth DHT

There are two broad approaches to supplying the PKI's constituent elements for the Stealth DHT: *external* and *internal*. In the former, the RA, CA and CR all exist entirely separately to any part of the Stealth DHT itself, whereas in the latter they exist as some subset of the Stealth DHT's service nodes. This subset may simply be a single, well-known service node (*centralised PKI*) providing the entire PKI functionality for the Stealth DHT, or in contrast it may consist of several or all the existing service nodes (*distributed PKI*).

To acquire a certificate, a user must first generate a public/private key pair, and then pass his or her public key along with any requested proof of identity to the Registration Authority. Following successful verification of these credentials by the RA, they are then passed to the Certification Authority. The CA then creates the certificate, signs it, and passes it back to the user via the RA. It may also be passed to a Certificate Repository, if necessary.

In a Stealth DHT with an external PKI that requires authenticated join operations, the user simply sends a join message containing their certificate to a suitable DHT bootstrap node, with the separate PKI server(s) being contacted as required. Conversely, the authenticated join procedure for a Stealth DHT with an internal PKI is similar, but involves messages being passed between the nodes on the DHT that constitute the various components of the PKI. In both cases, the DHT ID that the node joins as is irrelevant (unless associated with their certificate); the real identity of a user is always determined by the certificate they own.

Exactly how the PKI elements are organised is entirely application-specific. For example, as the RA and CA must be trusted entities, they may be comprised of a small number of highly-trusted service nodes with well-known identifiers. The CR, on the other hand, does not require a high level of trust due to the immutability of digital certificates. As a result, it could consist of several or all service nodes on the DHT, with certificates being hashed and stored as if they were normal DHT keys. Also, note that a new, uncertified user may simply pass all their relevant details to a bootstrap node as their join message, with no need for any further action on their part. As all elements of the PKI are contained within the DHT, a certificate can automatically be returned to them whilst an appropriate DHT join message with the newly created certificate attached is simultaneously forwarded, thus minimising join delay.

4.2. Actions following Authentication

Following a successful join, nodes may then communicate as normal over the Stealth DHT, authenticating each other as necessary. As an example of this, assume we have two users, Alice and Bob, who have joined a Stealth DHT with an internal PKI as a stealth node and a service node respectively. Alice wishes to send a message to Bob, who requires that messages be authenticated. The correct procedure would therefore be as follows:

Alice first creates a message, signs it, and delivers it to Bob via the DHT. Bob can then verify the signature, and thus the message integrity, using Alice's certificate. He may have acquired Alice's certificate from his certificate cache, a Certificate Repository, or from within the message itself. Bob can then recursively verify the issuers within the certificate chain, starting with Alice's certificate. This process continues until Bob reaches a certificate that he intrinsically trusts. At this point, the authentication is complete, and Bob can continue to handle Alice's message appropriately. Of course, if the certificate chain does not eventually lead to an certificate that Bob trusts, his attempt to authenticate Alice fails. Following any reply from Bob, Alice may perform the same process on his message to authenticate his identity, but only if mutual authentication is required.

Further to this, if Alice and Bob wish to ensure that their messages are kept confidential from even the other service nodes[2], they can simply use the public keys contained within each other's certificates to encrypt the messages' contents. To clarify, if Alice wants to send sensitive data to Bob, then she first uses the CR (*e.g.* service nodes hashing and storing certificates) to retrieve his certificate beforehand. Following this, she uses his public key to encrypt the message contents and her own private key to sign the message. Only Bob's private key can decrypt data encrypted in this manner, so Alice can be sure that only Bob is able to understand the message contents. Further to this, her signature ensures that Bob can be sure the message came from Alice, and that it was not tampered with.

Note that in an internal PKI, if the Certificate Repository functionality is spread across many service nodes, and certificate chains are not included in messages, users performing *get* operations may experience longer retrieval delays due to the need for the relevant certificates to also be requested from the DHT. Of course, this increased overhead should be weighed against the cost of using a centralised PKI, which potentially represents a single point of failure for all nodes on the network.

5. Implementation Considerations

5.1. Certification Hierarchy

Some thought should be placed into how the certification structure is organised within the PKI. The simplest approach for a Stealth DHT could be to have a single globally

[2]Recall that as stealth nodes are not involved in message forwarding, they cannot possibly access these messages via the DHT.

trusted key, used to sign certificates for service nodes only. However, users may require a more complex hierarchy for economic, political or security reasons; again, this is an entirely application-specific issue. A possible example could be that each department within a typical commercial organisation is granted a certificate signed by a single, highly-trusted master key. It may be that the master key needs to be kept physically secure and is therefore inconvenient to access. By introducing this extra level of hierarchy, however, the need for it to be used to sign certificates on a regular basis is negated. This approach also allows for a finer level of control, as if the privileges of an entire department needed to be revoked, for instance, it is simply a matter of invalidating the associated departmental certificate.

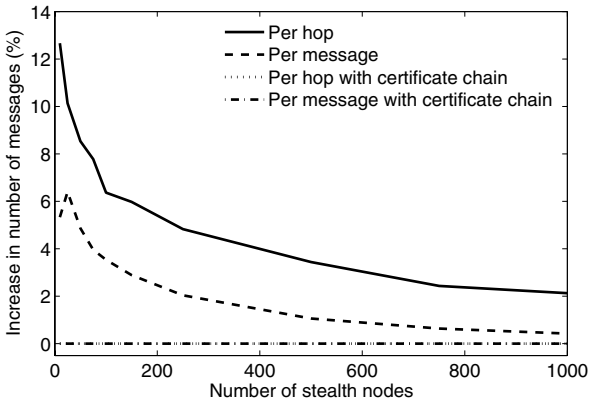
5.2. Authentication Granularity

Exactly how authentication is performed on the DHT presents an issue of balance between overhead and security. A fine-grained approach would be to verify messages on a per-hop basis. Naturally, this results in all required authentication operations (such as retrieving appropriate certificates or checking revocation lists) being performed an average of $logN$ times for each message, where N is the number of service nodes in the network. However, it also results in any invalid messages being dropped almost immediately, thus making it difficult for unauthenticated malicious nodes to get service nodes to pass invalid messages around; a tactic often used in DHT denial of service attacks, typically resulting in the network becoming overloaded with useless traffic.

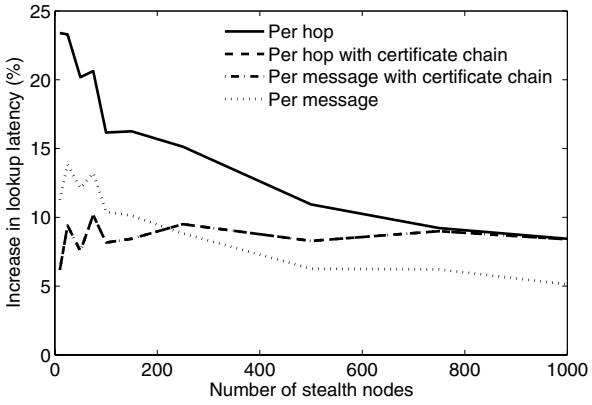
Considering a slightly coarser approach, if service nodes were to simply validate messages upon receipt at their final destination, then all associated authentication operations need only be performed once per message. Obviously this means that any messages from unauthorised nodes may be unnecessarily routed by multiple service nodes on their journey, consuming bandwidth and processing time. However, it also means that the number of authentication operations performed may be significantly reduced.

Beyond per-message authentication, service nodes may make use of the coarse concept of "sessions". In other words, a typical stealth node may be permitted to consume a certain amount of resources, make use of a given service or be active on the DHT for a pre-determined length of time. Examples here could include a node being allocated a set number of messages on the network, being given the ability to download a particular number of pieces of content or being provided with a "day-pass" to use the DHT, respectively.

Session-based authentication therefore requires that some state be stored and validated for stealth nodes. If the session is based on a set length of time, then this can be as simple as issuing a certificate with a corresponding period of validity. Otherwise, it is likely that a more complex system is required, such as an accounting mechanism, as discussed in Section 5.3 or a session authentication protocol such as Kerberos [13].



(a) *Increase in the number of messages*



(b) *Increase in lookup latency*

Figure 2. *Stealth DHT with 100 services nodes using an internal, fully distributed PKI relative to a Stealth DHT with no authentication. Note that the certificate-chain simulations are overlaid in these figures.*

Figs. 2 show results for simulations conducted with a fixed number of 100 service nodes, where the number of stealth nodes was varied between 10 and 1000. The simulations were carried out using our own discrete-event packet-level simulator, based on Pastry [17]. Each simulation was performed on a GT-ITM [4] generated transit-stub topology of 1,000 routers, with 4% transit nodes. Service, stealth and normal Pastry nodes were connected to this topology in a random fashion.

Every service node in the DHT initially held its own certificate as well as several content keys. During the simu-

lation itself, stealth nodes performed random *get* requests for these keys at regular intervals. Authentication was performed at the source and destination in the *per-message* results and also performed at each intermediate node in the *per-hop* results. For both cases, simulations were run with and without certificate chains contained within messages.

Fig. 2(a) shows that both the per-hop and per-message authentication schemes result in no increase in the overall number of messages when certificate chaining is used. This is because all the certificates required to fully authenticate a message's sender are included within the message itself, thus increasing the average message size, but resulting in no further requests to the CR.

On the other hand, an increase in the number of messages does exist when certificate chains are not used. This increase is due to the distributed nature of the Certificate Repository in an internal PKI; to acquire a certificate in order to perform authentication, a node must retrieve it from the service nodes through further DHT queries. Expectedly, per-hop authentication results in markedly more messages than per-message authentication. Also, as the number of stealth nodes increases, the percentage increase in the number of messages relative to a system without authentication falls. The reason for this is that the increased number of stealth nodes results in an accordingly increased number of requests for certificates. As all nodes cache certificates upon receipt, there is no need for certificates to be re-acquired.

Fig. 2(b) shows how lookup latency[3] is affected by these factors. As expected, the cases which involved querying the network (i.e. those without certificate chains) result in increased lookup latencies. Note, however, that the cases with certificate chains also incur increased lookup latencies despite the lack of extra authentication messages. This is attributed to the larger average message size that occurs as a result of including certificate chains within messages.

As seen in Fig. 2(a), the relative increase in the number of required authentication messages decreases with larger numbers of stealth nodes. Fig. 2(b) therefore displays a correlated reduction in lookup latency. As the cases with certificate chains do not generate extra authentication message, they remain unaffected by the number of stealth nodes in the DHT.

5.3. Permissions Management

Managing the permissions of nodes on the network is yet another issue with multiple solutions. One possibility is to simply place them within each node's corresponding certificate, but this has two notable associated issues. Firstly, certificates are immutable after they are signed, so altering the permissions for a node would require a certificate to be re-issued. Secondly, the average DHT message size would have to increase to accommodate the larger certificates; exactly how large they are depends on the volume of data required to represent all the relevant details. However, such an approach does mean that the relevant permissions are immediately available to any node receiving a message containing the originating node's certificate. If per-node permissions are not likely to change on a regular basis, this is therefore a lower-overhead solution than using a separate certificate repository.

An alternative approach would be to store permissions data within the network somehow, thereby avoiding the lack of flexibility and larger message sizes associated with storing them within certificates. The inevitable cost, however, arises in the form of extra messaging overhead; every time a node needs to check an individual's permissions, they would have to look up and also validate the relevant data.

The permissions for a given node in a system such as those discussed in this paper may require that some state is maintained for each individual. For instance, this may be based on a record of how much in the way of network resources or service usage the node has consumed. To provide such functionality, an accounting mechanism of some sort is therefore required.

5.4. Certificate Revocation

Invalidating a given certificate at an arbitrary point in time within a PKI structure (*i.e.* revoking it) is traditionally a difficult problem, especially in distributed environments. However, the need for the ability to revoke certificate often outweighs the cost of any associated overhead. In other words, being able to remove a node from a network before it can do any lasting damage may be worth the extra associated costs. There are a number of possible methods for certificate revocation, each with advantages and disadvantages (see [24]).

An obvious solution is for stealth nodes to be simply issued with certificates with short expiration times, resulting in a quasi-revocation scheme in which a CA can simply refuse to re-issue a given certificate if it wishes to remove a given entity's privileges within the system. Unfortunately, this intuitively has a high maintenance overhead, as every individual that continues to exist on the network will require a new certificate to be generated and issued at a regular interval.

A common approach to this problem is to use a CRL, or Certificate Revocation List. Any node verifying a certificate must then check the list as part of its normal authentication process. The list itself may be stored in one of a number of ways. For instance, and as with many of the other issues considered so far, a centralised server could be used. Again, the problems common to this sort of approach are that it cre-

[3]The time elapsed between a node requesting, receiving and fully verifying a key from the DHT.

ates a central point of failure, and could potentially result in the overloading of the server if many requests are regularly made. A distributed approach amongst service nodes could potentially be used to alleviate the load placed upon any one node, but this results in increased messaging overhead due to the added complexity of maintaining and retrieving the list. It is important to note that, of course, the list may be kept in its entirety on multiple nodes as well as the approach of splitting it amongst them (*i.e.* replication vs. division, respectively).

6. Related Work

Many previous works have discussed the varied problems associated with untrustworthy nodes in DHTs, noting that security is an issue commonly overlooked in algorithm proposals. For example, [20] and [5] both discuss possible DHT attacks and defenses. In the former, Sit and Morris broadly define three types of malicious behaviour: routing, storage and other miscellaneous attacks.

A commonly encountered technique often used in all three categories is the "Sybil" attack, as originally described by Douceur in [7]. This refers to the situation when a single malicious node is able to masquerade as multiple distinct entities within the network in order to gain control of a substantial fraction of it. The conclusion is drawn that a suitable defense against such an attack is to have a logically centralised authority which is capable of certifying the identity of nodes in the network. Therefore, this work can be said to support our approach of implementing a PKI in conjunction with a Stealth DHT.

Routing attacks in a DHT may refer to nodes deliberately providing incorrect lookups or producing incorrect routing updates. More specifically, an example could be the "Eclipse" strategy, as discussed in detail by Singh *et al.* in [19]. This involves multiple malicious nodes deliberately attempting to partition peer-to-peer overlays in a form of Distributed Denial of Service (DDoS) attack. Again, the suggestion is made of a certification scheme as a straightforward solution, as with our approach to such a problem. Beyond this, the authors propose defenses such as constraining the entries placed in routing tables [5] or periodically auditing the connectivity of other nodes to detect anomalies which are symptomatic of those conducting such an attack.

Storage attacks may involve behaviour such as nodes refusing to store objects, corrupting them or simply denying their existence. More resourceful attackers may also use multiple malicious nodes to attempt to take control of specific pieces of content. Some may even try to make it impossible to access useful content by flooding the network with useless data [11]. Srivatsa and Liu suggested the approach to obfuscate the location on the DHT of specific keys from those not authorised to access them [21], although the most commonly suggested solution is to make use of some sort of digital certification scheme, such as the one we have proposed in this paper. Furthermore, a Stealth DHT can ensure that potentially untrustworthy nodes are never even part of the DHT which is responsible for storing keys, although they may still access them if authorised to do so.

Of course, DHT-based storage systems have often considered security in their own right. PAST, for instance, uses the concept of "smartcards", with which users hold associated public/private keys [8]. These smartcards are managed by brokers (trusted third parties). In other words, PAST is yet another system that takes the approach of using a Public Key Infrastructure for security purposes; again, the concept we have suggested and expanded upon in this paper.

In terms of miscellaneous attacks, Sit and Morris also noted that an attacker may attempt to conduct a DDoS attack by causing multiple nodes under their control to rapidly join and leave the network, resulting in degradation of DHT performance [16][10]. Possible solutions to this problem are suggested in [5], such as forcing nodes to solve cryptopuzzles before they may join as a means of slowing down attackers attempting to run multiple logical nodes on a single physical machine. However, such an approach merely makes carrying out an attack slightly more difficult. Our Stealth DHT approach, however, means that stealth node churn has a significantly reduced effect on DHT performance relative to nodes churning in traditional DHTs.

Several works have also considered how such authentication systems may be implemented in a physically distributed fashion over peer-to-peer networks. For example, Aberer *et al.* discussed how a completely decentralised PKI based on a statistical approach could be deployed on many traditional DHTs (although they specifically use P-Grid [14]) [2]. The key difference in comparison with our work is that in this case, the authors consider a method that can function with a network consisting entirely of potentially untrustworthy nodes. However, they note that their system breaks down if more than 25% of nodes are actually malicious, and that it may not function with several DHTs, such as CAN or Chord [15][22]. We, however, believe that our system is implementable on any existing DHT, and should function regardless of the percentage of malicious stealth nodes.

7. Conclusion

The original goal of our Stealth DHT proposal was to provide a distinction between nodes of greater and lesser capabilities as a means of improving routing performance. Powerful nodes were responsible for handling message forwarding within the DHT, whereas the remaining, weaker nodes simply requested services from them. In this paper, we have shown that this separation can be extended to

incorporate both verifiably trustworthy and potentially untrustworthy nodes. By selectively limiting the privileges of untrustworthy nodes on the network, on an individual basis if required, we can accordingly limit the numerous security problems associated with supplying service to them. By further augmenting our approach with a suitable Public Key Infrastructure to enforce the separation between node types, we have shown how a Stealth DHT can be used to supply a secure, resilient overlay that caters to both trustworthy and untrustworthy nodes simultaneously. Stealth DHTs do not necessarily need to deny access to potentially untrustworthy nodes as opposed to previous approaches that addressed security issues in DHTs. Instead, Stealth DHTs only limit the operations that such nodes can perform. Further to this, we have discussed a number of issues and possible solutions related to implementing a PKI in a Stealth DHT.

In the future, we intend to add provisions for node authentication in our existing Stealth DHT implementation, so that it may be evaluated in a real-world environment.

References

[1] ITU-T Recommendation X.509. Information Technology Open Systems Interconnection - The Directory: Authentication Framework, August 1997.

[2] K. Aberer, A. Datta, and M. Hauswirth. A decentralized public key infrastructure for customer-to-customer e-commerce. *International Journal of Business Process Integration and Management*, 1(1):26–33, 2005.

[3] A. Brampton, A. MacQuire, I. A. Rai, N. J. P. Race, and L. Mathy. Stealth Distributed Hash Table: Unleashing the real potential of peer-to-peer. In *Proc. of the ACM Conference on Emerging Network Experiments and Technology (CoNEXT) (Student Workshop Session)*, October 2005.

[4] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[7] J. R. Douceur. The Sybil attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.

[8] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, May 2001.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proc. of ACM ASPLOS*, November 2000.

[10] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proc. of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2004.

[11] J. Liang, N. Naoumov, and K. W. Ross. The index poisoning attack in P2P file sharing systems. In *Proc. of IEEE INFOCOM*, April 2006.

[12] A. T. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proc. of the 3rd IEEE Workshop on Internet Applications (WIAPP)*, June 2003.

[13] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[14] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1997.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, August 2001.

[16] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proc. of the USENIX Annual Technical Conference*, June 2004.

[17] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[18] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Proc. of the 3rd International Workshop on Networked Group Communication (NGC)*, November 2001.

[19] A. Singh, T. Ngan, P. Druschel, and D. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. of IEEE INFOCOM*, April 2006.

[20] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.

[21] M. Srivatsa and L. Liu. Countering targeted file attacks using location keys. In *Proc. of the 14th USENIX Security Symposium*, July 2005.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of ACM SIGCOMM*, August 2001.

[23] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, USA, April 2001.

[24] P. Zheng. Tradeoffs in certificate revocation schemes. *ACM SIGCOMM Computer Communication Review*, 33(2):103–112, April 2003.

[25] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2001.