Available online at www.sciencedirect.com

ScienceDirect

Journal of Systems Architecture 54 (2008) 607–618

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Authentication in stealth distributed hash tables

Andrew MacQuire [*], Andrew Brampton, Idris A. Rai, Nicholas J.P. Race, Laurent Mathy

Computing Department, Lancaster University, InfoLab 21, Lancaster, Lancashire LA1 4WA, United Kingdom

Received 18 February 2007; received in revised form 20 January 2008; accepted 28 January 2008
Available online 2 February 2008

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. We now 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.
© 2008 Elsevier B.V. All rights reserved.

Keywords: Distributed hash table; Peer-to-peer; Security; Public key infrastructure; Stealth DHT

1. Introduction

Distributed Hash Tables (DHTs) have been shown to be a useful form of decentralised, structured peer-to-peer overlay [27,34,36,23,18]. 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 lookup substrates for many varied applications, such as large-scale file storage [11,8,14] and multicast [28,39], 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 final 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., attacks on confidentiality, integrity or availability respectively). Using the simulation platform described in Section 5 we can demonstrate how a relatively small number of untrustworthy nodes can have a large impact on a generic DHT. Fig. 1 shows the percentage of messages which would be affected as the corresponding percentage of malicious nodes increases. Observe that when a quarter

[*] Corresponding author.
E-mail addresses: macquire@comp.lancs.ac.uk (A. MacQuire), brampton@comp.lancs.ac.uk (A. Brampton), rai@comp.lancs.ac.uk (I.A. Rai), race@comp.lancs.ac.uk (N.J.P. Race), laurent@comp.lancs.ac.uk (L. Mathy).

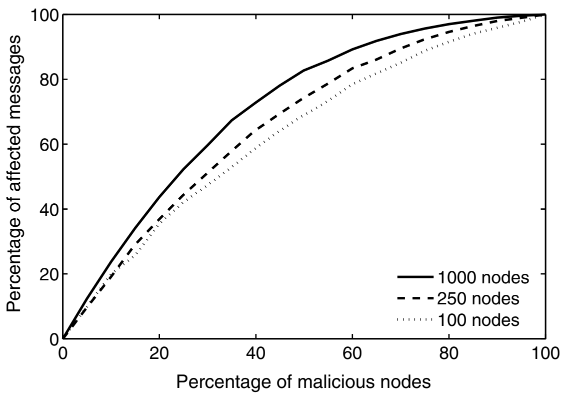


Fig. 1. The effect of malicious nodes forwarding messages.

of a 1000 node DHT's population is malicious, over half of the messages sent are subject to 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. Furthermore, this provides an ideal environment for users to infringe intellectual property law or distribute illegal content. This is a major factor that hinders commercial use of existing DHT architectures, especially now as peer-to-peer application/service providers are increasingly liable for illegal activity carried out on their networks [13].

To avoid these security problems, it is clear that some method 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 equal. 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. Instead, such nodes should still be allowed to 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. We therefore demonstrate how our recently proposed Stealth DHT concept [3] can be used to provide this precise functionality with almost any existing DHT algorithm, following minor modifications to nodes' announcement and/or state-gathering protocols. Our original evaluation covered the benefits of this approach from a performance standpoint, where excluding potentially slow or unreliable nodes from DHT forwarding paths was shown to be beneficial. Such an approach is particularly useful for users with poorly provisioned hardware (e.g. mobile devices), where access can still be granted to the DHT without degrading the overlay's overall forwarding performance. We now, however, consider the advantages gained in terms of security.

The separation between nodes that a Stealth DHT provides can be exploited to enable secure content distribution, as when coupled with a suitable authentication mechanism, it aids in returning network control to the service provider. To clarify, the ability to create fine-grained permissions for nodes on the DHT means that security policies are much easier to enforce. For example, by allowing only authorised nodes to perform certain operations, Stealth DHTs can ensure that only legitimate and useful content is served, thus 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 (PKI). 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, with further discussion of optional functionality in Section 6. Section 7 goes on to examine related work in the field of DHT authentication, and finally Section 8 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 (an already-connected node discovered through some alternate mechanism). Nodes along the message's path then reply directly with relevant routing information, as to allow the joining node to construct its routing tables. 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, as originally proposed in [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. In the case of DHT implementations which passively gather routing state from forwarded messages, a flag within the sender's certificate can indicate whether the sender is suitable for routing. The resultant effect of such modifications 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 table built by stealth nodes is used only for selecting the locally-optimal node to forward each of their own requests to, thus maintaining routing performance while removing the possibility of a single point of failure that many similar DHT super-peer schemes suffer from [19,38]. In the case of the locally-optimal node for a given message failing, a stealth node can select the next-best alternative from its table. Further detail on how a stealth node can also keep its routing table from becoming stale can be found in our previous work [3].

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. However, a Stealth DHT provides an individual or a service provider with the control to command such assignment. Therefore 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, 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, as service nodes never retain any knowledge of stealth nodes, 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 [24,15]. 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 occur and so no maintenance messages are required. 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. These properties of a Stealth DHT are examined in greater detail in [3].

Of course, in a system such as this there still must be a means of ensuring that stealth nodes cannot masquerade as service nodes (e.g. by simply announcing their presence to other service nodes). This can be achieved in a Stealth DHT through the use of an appropriate authentication scheme to effectively enforce the separation between node types. Accordingly, this is the focus of this work, wherein we discuss how an authentication scheme based on a Public Key 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 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 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 5.5.

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. 2. Each certificate contains a version record for compatibility reasons, as well as a serial

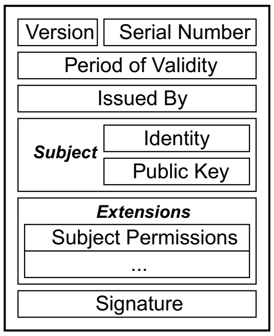


Fig. 2. Certificate format.

number to aid in certificate management. The certificate should also have two discrete dates associated with it, indicating its period of validity.

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 owners public key is also 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 digital signatures, 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, allowing for application-specific additions.

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. Furthermore, if the PKI supports revocation, an approach similar to those discussed in Section 5.5 would have to be performed for each certificate in the chain.

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, as to prove its authority. In the simplest case, stealth nodes would not require certification. If, however, restrictions were placed on any of the operations that stealth nodes could perform, certification would be required to prove that a given stealth node is authorised to perform a particular operation.

The certificate extension fields may therefore contain a list of authorised operations that its owner may carry out (see Fig. 2). Simple examples could be the right to join as a service node, or the right to put content 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. PKI composition

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 the functionality is provided by a set of operations on the DHT, supported by some subset of the Stealth DHTs 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*).

Exactly how the PKI elements are organised is entirely application-specific, and internal/external elements may be mixed. For example, as the RA and CA must be trusted entities, they may consist 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. As such, creating a distributed PKI of this type would involve exactly the same procedures as setting up the DHT normally.

4.2. Joining the DHT

The first step for a new user to take will typically be certificate acquisition. The user must therefore generate a public/private key pair, and then securely pass his or her public key along with any requested proof of identity to the Registration Authority. Secure delivery to the RA is required here as to avoid confidentiality issues, and can be achieved through the use of a well-known and implicitly trusted global key. Following successful verification by the RA, the details can then be 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 can simply send a join message containing his or her certificate to a suitable DHT bootstrap node, with the separate PKI server(s) being contacted as required. The same process is required for a Stealth DHT with an internal PKI, although the components of the PKI system are contained within the DHT instead. Note that in the latter case a new, uncertified user could also simply pass all his or her 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.

4.3. Sending messages

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, a stealth node and a service node respectively who have joined a Stealth DHT with an internal PKI. Alice wishes to send a message to Bob, who requires that messages be authenticated. The correct procedure (as shown in Fig. 3) 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. Naturally, 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.

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. This increased overhead should, however, be weighed against the cost of using a centralised PKI, which potentially represents a single point of failure for all nodes on the network. Many such trade-offs may have to be made in implementing this sort of system, as discussed further in Section 5.

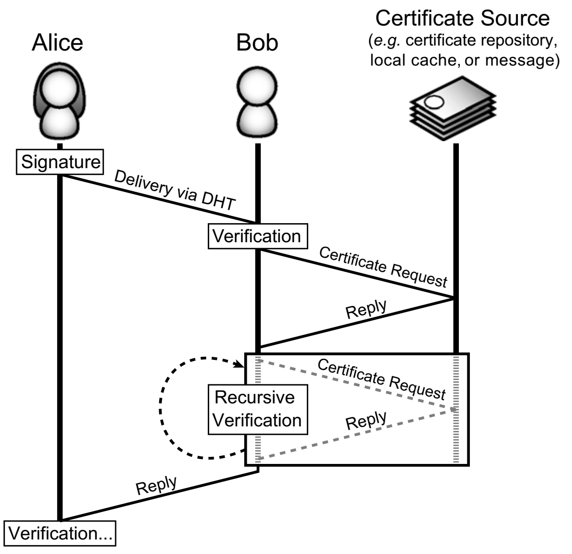


Fig. 3. Sequence of events for message authentication.

Further to this, if Alice and Bob wish to ensure that their messages are kept confidential from even the service nodes, 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 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.

The observant reader may note that the order of operations is of importance here (i.e., encrypting the message and then signing it, or vice versa) [9]. If the encrypt-then-sign approach is taken, then a malicious intermediary could replace the signature on the encrypted message, and thus claim to be its original source. For this reason, the source identifier should be included in the encrypted portion of the message. If, upon decryption, the source identifier does not correspond with the attached signature, the destination can deduce that the message was tampered with.

5. Implementation considerations

As Stealth DHTs may be tailored to a wide range of applications based on the assignment of roles to nodes, it follows that there are numerous application-specific decisions to be made regarding the implementation of any security system, as discussed in the following sections.

For evaluation purposes, we used our own discrete-event simulator based on a Stealth DHT implementation of Pastry [27] (validated in our previous work [3]). Packet-level simulations were conducted to allow for detailed examination throughout our experimentation, although the metrics shown in this paper are only at the DHT level. Each simulation was performed several times 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.

Results from our real-world C++ implementation of both Pastry and a Pastry Stealth DHT are also included in the following sections. Experiments were run with a randomly selected set of active PlanetLab nodes [21], which were connected to the DHT in a random fashion. Multiple iterations were run (at least 5 in all cases), with the network being given suitable time to reach a steady-state in all cases. Any obviously anomalous data due to instability on PlanetLab was discarded before analysis. The algorithm used in all authentication operations was the OpenSSL 0.9.8d implementation of RSA, with 1024 bit keys [12,25].

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 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. Asymmetry vs. symmetry

There are two obvious manners in which service and stealth nodes may authenticate each other: *asymmetric* (service or stealth node authentication only) or *symmetric* (mutual authentication). To clarify, service node authentication may be required if a stealth node is placing sensitive information into the network and wants to be sure of the recipient. Conversely, stealth node authentication may be required if a service node needs to verify if a stealth node is authorised to retrieve sensitive information from it. Finally, mutual authentication may be required if a combination of these factors arises. By taking this approach of authenticating the nodes only as required, processing and messaging overhead can therefore be minimised.

5.3. 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 all messages on a per-hop basis. For most DHTs, this results in all required authentication operations (such as retrieving appropriate certificates or checking revocation lists) being performed on average log$N$ 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, 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. Regardless, studies such as [29] have shown how there may be large numbers of legitimate, small flows in peer-to-peer systems, making per-message authentication for all messages an expensive choice in terms of overhead.

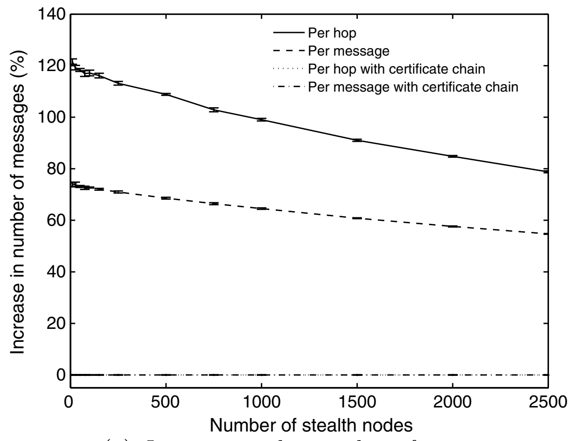
Considering a slightly coarser approach, if service nodes were to simply validate each message (or even a particular type of message) upon receipt at its final destination, then all associated authentication operations need only be performed once. Obviously this means that any messages from unauthorised 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. A possible solution to this issue would be for service nodes to check any message received from a node with which they do not have an established secure connection. By doing so, any messages from unauthorised nodes would be dropped upon insertion to the DHT, whereas relatively few authentication operations would be required once common routes between service nodes become established. Of course, the benefits of such an approach are offset by the overhead of maintaining the connections in the first place.

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 permitted to send or receive 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.

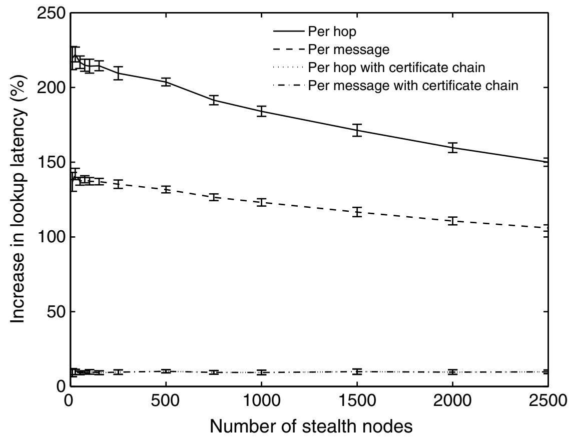
Session-based authentication therefore requires that some state be stored and validated for stealth nodes. If the session is based on a length of time, then this can be as simple as issuing a certificate with a corresponding period of validity. Otherwise, it is likely that a more complex system is required, such as an accounting mechanism, as discussed in Section 6.1 or a session authentication protocol such as Kerberos [20]. Of course, whether or not such schemes would be feasible to use in terms of overhead would likely be an application-dependent issue.

To measure the effects of the different authentication schemes we ran simulations with a fixed number of 1000 service nodes, where the number of stealth nodes was varied between 10 and 2500. These values were used as to strike a balance between simulation size and the processing resources available to us – the simulation platform used is described in greater detail at the beginning of Section 5. Every service node in the DHT initially held its own certificate as well as several content keys. During the simulation itself, stealth nodes performed 100 random *get* requests for these keys at regular intervals. Authentication operations were 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. Each simulation was repeated nine times, as by this point the 95% confidence interval of the results was suitably small, as can be observed from the figures.

Fig. 4a 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,



(a) *Increase in the number of messages*



(b) *Increase in lookup latency*

Fig. 4. Stealth DHT with 1000 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. Error bars are used to show the 95% confidence intervals.

thus increasing the average message size, but resulting in no further requests to the CR. Note that in this case, the larger the certificate hierarchy required for authentication, the greater the average message size.

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. 4b shows how lookup latency (the time elapsed between a node requesting, receiving and fully verifying a

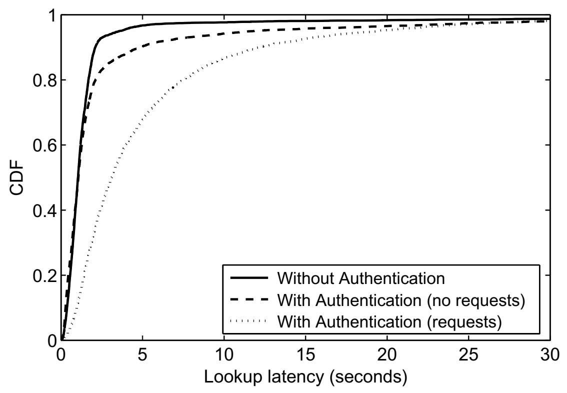


Fig. 5. Impact of per-hop authentication upon lookup latency with/without certificate requests.

key from the DHT) 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.

The same conclusions can be drawn from Fig. 5, which shows the CDF of lookup latencies for our implementation running on PlanetLab. Simple certification hierarchies of two and three levels were used. In the two level case all certificates were signed by a global key, whereas the three levels had a intermediate certificate which had to be requested if it was not already cached. This is plotted on the figure as “no requests” and “requests” respectively. We again witness that when requests for certificates are required, lookup latency is increased relative to a system without authentication. The larger message size required for authenticating messages also results in the “no request” scenario exhibiting slower lookups, although the difference is minor for the majority of cases.

As seen in Fig. 4a, the relative increase in the number of required authentication messages decreases with larger numbers of stealth nodes. Fig. 4b 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.

5.4. Processing overhead

An inevitable overhead involved with providing a PKI-based authentication system is the increase in processing time required for signing and verifying certificates. The effect this might have on a DHT is difficult to realistically quantify through simulation alone, and so we investigated the impact of using authentication on the performance of our real-world implementation.

As noted in Section 5.3, Fig. 5 shows the CDF of lookup latencies for our implementation running on PlanetLab, with and without authentication. Approximately 300 nodes were used, where each node would send 10 requests overall, delivered at exponentially distributed times around a mean of 6 seconds. The requests were made for a set of 1000 keys, with a Zipf popularity distribution using an α parameter of 1.2 (thus providing a realistic popularity function as commonly observed in peer-to-peer networks [32]).

From the figure, it is clear that approximately a quarter of the lookups made were noticeably slower when per-hop authentication (without requests) was used, relative to a DHT with no authentication mechanism. However, upon varying the authentication granularity to per-message (not shown), no improvement was witnessed. This implies that processing overhead may not be a major factor in the performance decrease, as the amount of processing in per-message authentication would be greatly decreased relative to per-hop.

Further investigation involved repeatedly benchmarking the authentication code on the PlanetLab nodes used. In all benchmarks, timing precision was ensured by recording the aggregate elapsed times over many iterations. It was found that over 90% of nodes were able to sign or verify a 512 byte message with a 1024 bit RSA key in less than 100 ms (and 70% less than 50 milliseconds). In comparison with network latency, it therefore seems unlikely that processing overhead played an significant role in the performance decrease observed. Even though encryption was not used in the implementation, benchmarks reveal that a PlanetLab node on average can encrypt 30,000 512 byte messages per second using AES, and is thus not a major bottleneck. The huge difference in speed between signing and encrypting is attributed to the symmetric and asymmetric algorithms in use.

Also, note that many of the values in this section are somewhat indicative of the high load on PlanetLab: the same test performed on a typical unloaded PC signed messages in less than 3 milliseconds and encrypted 50,000 messages a second.

It appears the actual reason for the longer lookup latencies is simply the delay involved in delivering the extra authentication data. Without authentication, messages were around 256 bytes in length; with authentication, messages were around 512 bytes larger. It is clear therefore, that a lack of processing power is not a major problem; instead, minimising the need to transmit certificates (e.g. by using suitable certification hierarchies) and ensuring network links are suitably provisioned are more important issues.

5.5. 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 [37].

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. Such an approach would use the credentials supplied by the individual as a unique identifer, and so they would have to be suitably difficult to counterfeit as to avoid attackers simply rejoining under, for example, a different name. Furthermore, this approach 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.

Another common approach 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 creates 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 the list could also be kept in its entirety on multiple nodes instead of split between them (i.e., replication vs. division, respectively).

There also exist several approaches for the manner in which nodes access such lists. In general terms, these fall into the *pull* and *push* distribution models. As an example of the pull model, nodes may poll the holder(s) of the CRL at regular intervals, or even for every transaction, depending on the level of granularity required. An example of the push model would be the Certificate Revocation List being broadcast to all nodes upon any update, or at a regular interval. Obviously, such an approach may result in a great deal of initialisation overhead if a large number of nodes which require the CRL exist; multicast overlays may have to be built, and ideally with some method of guaranteeing delivery. For small numbers of nodes, however, the cost of broadcasting updates and/or occasionally broadcasting the full list may be lower than having nodes repeatedly request it unnecessarily. The scalability of such a system would depend entirely on the number of nodes that require the CRL (which equally depends on how restrictive the DHT is). DHT-based application-level multicast systems such as SCRIBE [6] and Bayeux [39] have been demonstrated as scalable (given they leverage the scalability properties of the DHT on which they are constructed), so a broadcast approach may be beneficial even in some large-scale circumstances.

6. Optional considerations

Beyond the mandatory decisions regarding the structure of the Stealth DHT PKI, there are also a number of additional considerations that may be required for certain applications, as discussed in the following sections.

6.1. Permissions management

Allowing a node to have elevated permissions on the DHT requires some knowledge of how trustworthy it is. Indeed, the importance of having a good trust metric is proportional to the level of damage that a restricted operation could inflict. For instance, if a node is required to store sensitive data, it should probably be hand-picked by the service provider. Conversely, if a node is only required to store publicly available data an automatic measure of trust may be used. Something as simple as a record how of successful the node has been at serving requests previously could be useful here; the system would then begin to trust successful nodes more, and unsuccessful or increasingly malicious nodes less. Of course, more complex approaches may also be suitable in various scenarios. For instance, many systems rely on a graph-theoretic "web-of-trust" approach, where trustworthiness is inferred through trust relationships between known and unknown nodes. Choosing exactly what approach to take is, of course, another application-dependent issue, and would be based on the level of security required by any given application on the Stealth DHT.

Actually 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 to either grant or restrict its current access level would require a certificate to be re-issued, and the original invalidated. 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.

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 unknown individual's permissions, they would have to look up and also validate the relevant data. This cost could be offset somewhat by using a number of techniques, such as permission tokens that remain valid for a day or so.

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

As with most of the issues discussed so far, there are numerous possible approaches to the problem of providing an accounting mechanism for nodes in a Stealth DHT, each with advantages and disadvantages. Nevertheless, the common goal of each is identical: to provide a system which offers ACID properties. These are *Atomicity* (an action is either performed completely or not at all), *Consistency* (an action cannot place the data in an invalid state), *Isolation* (actions cannot interfere with each other's intermediate states) and *Durability* (the result of an action will persist).

The most straightforward approach is to have a simple centralised server handle all transactions. This means messaging overhead is minimal, although it also means that a single point of failure exists for the system. Furthermore, dependent on the number of nodes considered, the server may be placed under load beyond its capabilities.

Another possibility would be to improve load-balancing by distributing the transaction processing across multiple nodes. For instance, this could be achieved by mapping a given node's state to a key in the DHT. However, it is important to note that without replication, such a system would be prone to losing data from time to time. With replication, however, the management overhead required to maintain the aforementioned ACID properties will intuitively increase due to the need for agreement amongst the replicated nodes.

The classical approaches for distributed accounting are divided into a few main categories, *local accounting*, *quorum-based* [7] and *token-based* [17,35,26]. Local accounting refers to the case when nodes do not collaboratively maintain a count, but only keep values locally instead. In contrast, quorum-based systems require a group of nodes to maintain the count in a cooperative fashion. Before the count is modified the quorum must reach a consensus on its old and new value. The more nodes that exist in the quorum the more robust the system is to malicious nodes and network failures.

Token based systems may also use similar approaches, but the key difference is that each count is represented by many immutable tokens. Each token represents a currency that nodes can exchange for service. These tokens are cryptographically signed to restrict who can use them. Nodes acting as "banks" may be used to store and certify the tokens to ensure that double spending does not take place. Tokens have the benefit that they can either be self-issued [35], or can be issued by the system [26].

Variations on mechanisms such as these may also be used to provide logging functionality in the DHT, thus making users accountable for their actions.

6.2. Node promotion

A key function of a Stealth DHT with authentication support is the strong separation between stealth and service nodes. Sometimes, however, it may be useful to allow nodes to alter their roles. For instance, if a Stealth DHT with a small number of service nodes is heavily loaded, the ability to 'promote' stealth nodes to a service role for offloading purposes could be advantageous.

In most cases, it would perhaps be unwise to offer a stealth node full privileges outright; without a careful selection process and suitable permissions management, the potential for abuse would be high. Determining suitable stealth nodes would most likely be a two stage process: first, discovering which nodes have the required resources available, and secondly, ascertaining which nodes within that set can be classed as trustworthy.

Finding stealth nodes with spare resources would not necessarily be difficult. On a heavily loaded DHT, service and stealth nodes would be communicating often, and so service nodes could simply attach a request for support to any replies they send. A suitable stealth node could then reply to this request, offering its spare capacity in whatever form it may take. Stealth nodes may also offer their services, either due to purely altruistic reasons, or for incentives placed by the service provider (which may take the form of access to exclusive resources, monetary compensation or similar rewards).

Exactly what resources are being offered has an important bearing on how rigourous the service node's subsequent examination of the stealth node should be. As a simple example, storing replicated certificates requires little to no trust at all; the certificates cannot be maliciously altered without it being obvious, and should the stealth node disappear, the original certificate is not lost. In contrast, if a stealth node were to be enlisted to store the only copy of a piece of unprotected content, it would have to be trusted not to alter it, or to disconnect without replicating it. Note that neither of these examples actually require the stealth node to be part of the DHTs routing, as the stealth node could simply register with a service node as auxiliary storage space.

In the case of a former stealth node becoming involved in forwarding messages for the DHT, even greater care is required when validating the node's trustworthiness. For instance, if a malicious node is only used for storing protected data, the worst it can do is deny service for that data alone. If, however, the same malicious node is involved in DHT routing, it could supply legitimate service nodes with false routing data, potentially damaging the entire DHT.

Compromises are therefore required, and thus the concept of promotion is inextricably linked with the previous discussion on managing node permissions (see Section 6.1). Furthermore, it is important that any new privileges given to a stealth node can be revoked easily, an issue discussed in Section 5.5.

6.3. Digital rights management

As has been previously noted, a Stealth DHT with authentication support may prove to be a useful platform for content distribution. However, while a Stealth DHT as discussed thus far can ensure the content is delivered safely, commercial services may also require that the data be placed under certain restrictions. A typical example would be copy protection, where users are prevented from accessing content they are not licensed to use.

A common, generalised approach to this problem involves the use of licensing servers. All content is encrypted during the delivery process, and the licenses granted to users via the servers act as decryption keys. In some cases, every time the content is played back, a license has to be acquired. Naturally, this means that the license servers have to be reliable to ensure user satisfaction.

A natural extension for a Stealth DHT serving such content could therefore be to distribute licensing servers across the DHT. Many of the issues discussed thus far regarding Public Key Infrastructures also apply to DRM systems, as PKIs are often used within them. A Stealth DHT could therefore offer both user authentication and content protection within a single system.

7. Related work

Previous works have discussed the varied problems associated with untrustworthy nodes in DHTs, noting that security is an issue commonly overlooked in algorithm proposals. Sit and Morris broadly define three types of malicious behaviour: routing, storage and other miscellaneous attacks [31].

A commonly encountered technique often used in all three categories is the "Sybil" attack, as originally described by Douceur [10]. 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. Such a solution is effective because certificates are associated with individual nodes; a single compromised node cannot assume the identity of many without having the extra associated certificates. Therefore, this paper 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. [30]. 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 [16]. Srivatsa and Liu suggested the approach to obfuscate the location on the DHT of specific keys from those not authorised to access them [33] Similarly, 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 [11]. 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 work.

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 [24,15]. Possible solutions to this problem are suggested in [5], such as forcing nodes to solve crypto-puzzles before they may join as a means of slowing down attackers attempting to run multiple logical nodes on a single physical machine. In contrast, our Stealth DHT approach means that stealth node churn has a significantly reduced effect on DHT performance relative to nodes churning in traditional DHTs. Exactly how churn affects a Stealth DHT is studied in detail in our original proposal [3].

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 [22]) [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 [23,34]. We, however, believe that our system is implementable on almost any existing DHT, and should function regardless of the percentage of malicious stealth nodes. Note, however, that the same can be said of a simple server cluster - indeed, a small Stealth DHT is effectively just that, albeit with the beneficial self-organising, load-balancing and reliability properties of the DHT on which it is based.

8. 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. We have demonstrated 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, the operations that such nodes can perform need only be selectively limited. A Stealth DHT coupled with the authentication mechanisms described could therefore form an ideal secure location substrate for the numerous varied applications that DHTs can support.

References

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

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

[3] A. Brampton, A. MacQuire, I.A. Rai, N.J.P. Race, L. Mathy, Stealth distributed hash table: a robust and flexible super-peered DHT, in: Proceedings of the 2nd Conference on Future Networking Technologies (CoNEXT), Lisbon, Portugal, 2006.

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

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

[6] M. Castro, P. Druschel, A.-M. Kermarrec, A. Rowstron, Scribe: a large-scale and decentralized application-level multicast infrastructure, IEEE Journal on Selected Areas in Communication (JSAC) 20 (8).

[7] M. Castro, B. Liskov, Practical byzantine fault-tolerance, in: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI), 1999.

[8] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, Wide-area cooperative storage with CFS, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), 2001.

[9] D. Davis, Defective sign & encrypt in s/mime, pkcs#7, moss, pem, pgp, and xml, in: Proceedings of the General Track: 2002 USENIX Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 2001.

[10] J.R. Douceur, The Sybil attack, in: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

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

[12] <http://www.openssl.org>, OpenSSL: The open source toolkit for SSL/TLS.

[13] <http://www.supremecourtus.gov/opinions/04pdf/04-480.pdf>, Supreme Court of the United States, no. 04.480. Argued March 29, 2005. Decided June 27, 2005.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, OceanStore: an architecture for global-scale persistent storage, in: Proceedings of ACM ASPLOS, 2000.

[15] J. Li, J. Stribling, T.M. Gil, R. Morris, M.F. Kaashoek, Comparing the performance of distributed hash tables under churn, in: Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS), 2004.

[16] J. Liang, N. Naoumov, K.W. Ross, The index poisoning attack in P2P file sharing systems, in: Proceedings of IEEE INFOCOM, 2006.

[17] N. Liebau, V. Darlagiannis, A. Mauthe, R. Steinmetz, A token-based accounting scheme for p2p-systems, Tech. Rep. TR-2004-05, Technische UniversitSt Darmstadt (January 2004).

[18] P. Maymounkov, D. Mazières, Kademlia: A peer-to-peer information system based on the XOR metric, in: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

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

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

[21] L. Peterson, D. Culler, T. Anderson, T. Roscoe, A blueprint for introducing disruptive technology into the Internet, in: Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I), 2002.

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

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of ACM SIGCOMM, 2001.

[24] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz, Handling churn in a DHT, in: Proceedings of the USENIX Annual Technical Conference, 2004.

[25] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.

[26] R.L. Rivest, A. Shamir, PayWord and MicroMint: Two simple micropayment schemes, in: Security Protocols Workshop, 1996.

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

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

[29] S. Saroiu, P.K. Gummadi, S.D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking, 2002.

[30] A. Singh, T. Ngan, P. Druschel, D. Wallach, Eclipse attacks on overlay networks: threats and defenses, in: Proceedings of IEEE INFOCOM, 2006.

[31] E. Sit, R. Morris, Security considerations for peer-to-peer distributed hash tables, in: Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

[32] K. Sripanidkulchai, The popularity of gnutella queries and its implications on scalability, in: OReillys, 2001. <http://www.openp2p.com>.

[33] M. Srivatsa, L. Liu, Countering targeted file attacks using location keys, in: Proceedings of the 14th USENIX Security Symposium, 2005.

[34] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for Internet applications, in: Proceedings of ACM SIGCOMM, 2001.

[35] W. Thigpen, T.J. Hacker, L.F. McGinnis, B.D. Athey, Distributed accounting on the grid, in: Proceedings of the 6th Joint Conference on Information Sciences, 2002.

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

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

[38] Y. Zhu, H. Wang, Y. Hu, A super-peer based lookup in structured peer-to-peer systems, in: Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems (PDCS), 2003.

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