A Resilient Architecture for DHT-based Distributed Collaborative Environments

Simone Cirani, Natalya Fedotova, Luca Veltri
University of Parma (Italy)
Viale G.P. Usberti 181/A
I-43100 Parma - Italy
{simone.cirani, fedotova}@tlc.unipr.it, luca.veltri@unipr.it

ABSTRACT
Distributed Hash Tables (DHTs) provide a flexible and reliable infrastructure for data storage and retrieval in peer-to-peer communities. We propose to apply Kademlia DHT to organize data management and cooperation between users participating in different work-groups. Particularly, in this paper we propose a mechanism for increasing the resilience and the overall performance of a Kademlia-based distributed work-sharing system, taking into account frequent joins and leaves of network nodes. To achieve this goal we propose a new flexible scheme for resource management that provides more resilience and fault tolerance than other mechanisms used by existent cooperative storage systems with a collaborative nature. In this work we try to extend and generalize our solution to fit several application contexts of collaborative computing, thereby addressing some common problems about resilience of existent distributed collaborative systems.

Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *distributed systems, performance evaluation.*

General Terms
Algorithms, Performance, Design, Security.

Keywords
Distributed Hash Tables, resilience, fault tolerance, collaborative computing, peer-to-peer.

1. INTRODUCTION
Currently, Distributed Hash Tables are widely used by a number of peer-to-peer (P2P) applications as data storage and retrieval infrastructure. DHT mechanisms are realized on the overlay layer by building up a virtual (logical) communication scheme above the existent network topology [1].

SERENE 2008, November 17-19, 2008, Newcastle, UK.

In a DHT-based P2P system, a group of distributed hosts collectively manage mapping from keys to data values according to some specific algorithm. Some well-known DHT algorithms are Chord [2], Kademlia [3], Pastry [4], Tapestry [5], etc. DHTs provide efficient routing performance, high scalability, self-organizing data storage and lookup mechanisms.

In our previous work [6] we tried to extend the limits of application of DHT mechanisms from data storage and retrieval in distributed computer systems to collaborative computing context. We proposed to apply Kademlia DHTs to organize data management and cooperative work between geographically distant users of enterprise networks engaged in a common task in real-time and transparent mode. The presented solution was meant to create a distributed work-sharing system which allows data produced by some department to be maximally available for other interested entities in accordance with a predefined policy of attribution of different data access privileges to users from various departments of the same organization.

In this paper we propose a new DHT-based architecture for collaborative computer systems improving the previous solution and focusing on the following objectives:

- increase of the resilience and the overall performance of the system in the presence of frequent joins and leaves of network nodes;
- enhancement of the fault tolerance and security components of the proposed solution;
- generalization of the proposed solution to fit other application contexts of collaborative computing (besides hierarchical enterprise environments) with the purpose to address some resilience problems that are common for most distributed collaborative systems.

In order to increase the DHT performance, we decided to decouple the information maintained by the DHT and the actual data shared by network nodes in a group-based fashion. This approach lets us remove the group-based ID-assignment presented in the previous solution. This results in a single and larger ID space without any hierarchy, and as a consequence, we obtain an increase of node cooperation, resilience and fault tolerance of the system. We call it "resource management in a unique DHT".

To enforce security, we extend the original Kademlia protocol procedures with a certificate-based challenge-response authentication protocol to achieve effective DHT protection against different possible attacks. The same certificate-based

mechanism is used to encrypt DHT operational communications, when required, providing complete privacy guarantee.

At application level, the same user certificates are used to guarantee end-to-end application security (access control, authentication, confidentiality). Particularly, access control is implemented by introducing specific access policies associated to different user work-groups. This allows authorized users to access only resources for which they have appropriate access rights.

The proposed architecture has been also implemented in a demonstrative testbed as described below.

In section 2 we discuss resilience properties of different DHT algorithms and shortcomings of some existing distributed collaborative systems due to the DHT mechanisms applied. Section 3 recalls our previous work on the Kademlia-based collaborative system for hierarchical environments. In Section 4 we extend and generalize our previous solution in order to address some resilience and fault tolerance issues of distributed collaborative systems based on DHTs. Sections 5 and 6 define the certificate-based mechanisms to enforce DHT protection and application-layer security. The implementation details of the proposed approach are presented in section 7. Finally, in section 8 we draw some conclusions.

2. BACKGROUND AND MOTIVATIONS

Recently, a great number of P2P platforms, distributed computing and cooperative storage systems have adopted lookup and routing mechanisms based on Distributed Hash Tables (eDonkey, BitTorrent, CFS, OceanStore). DHT mechanisms allow to realize a self-organizing system with efficient routing performance, high accuracy of search, high scalability and automatic load balancing. At the same time different DHT-based algorithms offer different levels of system resilience. Let's consider some examples.

2.1 Resilience through DHT mechanisms

The resilience of a Chord-based network is strictly tied to the routing scheme based on a node's knowledge of its predecessor and successor in the identifier space. Each node in Chord maintains two data structures: a "successor list" (which is the list of peers immediately succeeding the node identifier) and a "finger table" (which contains Chord IDs and IP addresses of peers at exponential distances around the ID space from the node, in a data structure that resembles a skip-list) [2]. To locate every key in the network at any moment each node's successor and predecessor pointers and finger lists should be correctly maintained and perfectly updated in order to reflect any single procedure within the DHT. This becomes quite a difficult task due to the dynamical nature of P2P systems, where nodes join and leave the network in an unpredictable manner. So, to avoid failures of lookup processes, nodes should call *stabilize()* and *check_predecessor()* RPCs to periodically repair the routing table entries. When a node n_i fails, nodes whose finger tables include n_i must find n_i's successor in order to prevent the disruption of queries that are already in progress, i.e. to provide the convergence of lookup algorithms launched before to a target node. Obviously, the above mechanisms cause an increase of network traffic load due to specific messages used only for maintaining DHT topology.

In Pastry [4] each node maintains a routing table, a neighbor set and a leaf set. The last is conceptually similar to the Chord's successor list. When some node goes off-line, only leaf sets of its neighbors are immediately updated; routing tables data are corrected on demand, only when some peer tries to contact a node that is currently is not available. So, as in Chord, in Pastry we have a routing scheme depending on the stability of leaf sets of single peers (lists of the closest nodes), even if in Pastry lookups are performed by prefix-matching. Thus, the system resilience in these cases can be provided only through perfectly and timely updated contact lists.

Kademlia [3] provides a tree-like data structure of routing scheme, where network nodes are considered as "leaves" of a binary tree. Every Kademlia peer stores information about IP-addresses, UDP-ports and node IDs for a group of nodes from the interval $[2^i, 2^{i+1})$ for each $0 \le i < 160$ sorted by time last seen. Such contact lists are called k-buckets. Due to the k-bucket mechanism a node *n* in Kademlia has at least one contact in each sub-tree, if that sub-tree contains a node *m* which has somehow previously interacted with *n*. So, any node can locate any other node by its ID without maintaining lists of the closest peers: it is enough to "know" any node with the prefix possibly closer to a target node ID to launch a lookup. Even if some nodes from corresponding k-buckets have failed and the contact lists have not been updated immediately, it cannot cause a failure of the lookup. The last-recently seen eviction policy is applied to minimize changes in the population of k-buckets and to maintain contacts with the highest uptime. This results in routing table robustness against poisoning attacks, where malicious nodes try to pollute contact list in order to disrupt the execution of RPCs. The time required to achieve the goal of such an attack depends on the activeness of honest nodes already present in k-buckets, since as long as they remain online new nodes (malicious and honest) would do not be able enter the k-buckets. Under certain node stability circumstances, such attacks may never succeed.

Lookups are performed iteratively in parallel mode: a host contacts peers with progressively smaller XOR distances to the target ID in turn (prefix matching). The symmetry of the XOR-metric provides peers with a possibility to learn and update routing information from queries they receive during lookup processes: routing tables updates are implemented by nodes automatically, as a "side effect" of ordinary lookups and interactions with other nodes.

Hence, Kademlia provides better resilience in respect of the DHT-based algorithms described above due to the routing scheme based on k-buckets.

Another important aspect in providing system resilience is data replication that consists in storing a <key,value> pair corresponding to a certain resource at nodes with the IDs closest to the key (ID) of the resource in the identifier space. Commonly, DHT algorithms offer two ways of data replication: the "value" can represent information about an "address" of a node where a resource is stored and a brief description of the content (metadata), as well as the resource itself.

In Chord the replication of data is delegated to applications and it is implemented by storing resources under two or more different keys derived from the data's application level identifier through hashing procedure. Alternatively, a node can replicate a key/value pair on each of its successors. Pastry uses the same principle as Chord for data replication on nodes from successor lists. In Kademlia, data are replicated by finding k (usually k=20) nodes closest to a key according to the XOR metric and storing the key/value pair on them. Therefore, Chord and Pastry allow data replication only on the application layer through key redundancy techniques. Kademlia provides data replication on the DHT layer through storage redundancy, i.e. the same key is stored on different nodes. Key redundancy mechanisms can also be applied in order to achieve a suitable data replication level.

So, our interest in Kademlia is motivated by several particular characteristics of the routing scheme that allows us to suit some specific requirements regarding resilience and fault tolerance in distributed collaborative environments. Moreover, the binary tree-based structure of the identifier space permits both the assignment of network node IDs and the management of privileges for diverse work-groups in a simple and intuitive mode. Due to the lookup through prefix-matching the lookup speed can be increased by considering b bits (instead of one bit) at each step, reaching a desired resource in less time.

2.2 Related work

Recently, the idea of applying DHT mechanisms to organize cooperative storage and data management systems has been realized in a number of solutions based on different DHT algorithms: PAST [7], CFS [8], OceanStore [9], OverCite [10], etc. A detailed comparative analysis of the hash-based P2P systems listed above is given in [2,8]. Here we just briefly run through some of them in order to individuate some common problems of such applications regarding system resilience and fault tolerance.

The Cooperative File System (CFS) represent a distributed read-only file storage application based on the Chord data location service. It is designed as a collection of servers that provide block-level storage (rather than storing whole files like in PAST) in combination with a mechanism of virtual servers that helps to handle the data load balance and spreading blocks of resources over different servers. Such way of data storage is quite critical from the point of view of system resilience and fault tolerance in the case of simultaneous failure of several servers where different blocks of the same resource are hosted. So, CFS must copy blocks between servers whenever a node joins or leaves the system in order to maintain the desired level of replication. Moreover, to provide the resilience of routing mechanisms based on Chord principles, clients should maintain an up-to-date server list (successor lists) as it was described in Section 2.

OverCite is a cooperative digital research read-only library based on DHT principles that spreads the data load over a few hundred volunteer servers. It is implemented through partitioning the inverted index among numerous participating nodes, so that each node indexes only some fraction of the shared documents. OverCite uses meta-data-based data storage scheme.

OceanStore is a global persistent storage utility that involves data privacy mechanisms, guarantees durable strorage and allows clients to update shared data. However, any data update (modification) generated by clients can be committed only if it has been approved by nodes of so called "primary tier". These responsible nodes perform a Byzantine agreement algorithm for conflict resolution regarding data updates. The mechanism of "primary tier" nodes introduces into the system elements of hierarchy and centralization that may become critical points of the network architecture (DoS, flooding, etc). The location service is implemented through routing scheme based on Plaxton trees (Tapestry). To provide resilience of the routing mechanisms, Tapestry exploits network path diversity in the form of redundant routing paths. It also uses a surrogate prefix-matching routing scheme, where each nonexistent ID is mapped to some active node with a similar ID. Nevertheless, all the above mechanisms significantly augment complexity of the system. Furthermore, OceanStore assumes that the core system will be maintained by commercial providers.

The particularity of our solution is the possibility for different work-groups to cooperate in real time and not only in read-only manner, but in active regime that allows peers to modify shared data and to effectuate collective edition (e.g. during a process of program code writing by different software engineers). We propose a new resilient resource management scheme based on Kademlia routing scheme in order to overcome the limitations of other DHT algorithms. Moreover, we use meta-data storage mechanisms to cope with resilience problems caused by the dynamical nature of P2P environment. To improve the fault tolerance and security components of the proposed solution, a certificate-based authentication protocol for the lookup procedure and specific policies for work-groups engaged in different tasks have been introduced.

3. PREVIOUS WORK

Originally, the solution was meant for a network community represented by several groups of nodes with different levels (1, 2, 3…) of responsibilities and data access rights.

To effectively manage different data access privileges, we introduced a new identification system based on use of prefix identifiers (*prefix_IDs)* for both nodes and resources to handle read/write permissions in accordance with a certain enterprise hierarchical model.

We split the Kademlia 160-bit node identifier into two strings of bits, the former we call prefix_ID, and the latter - node_ID. The length of the prefix string is β bits, and the remaining $160 - \beta$ bits represent the node_ID. The prefix defines a level that a node/resource belongs to. Kademlia tree-like data organization is easily adaptable to this scheme of ID assignment.

To join the network a node contacts a bootstrap terminal, which recognizes the level of privileges the node can enjoy by examining its certificate. Then, the *prefix_ID* predefined for nodes of this level is assigned to the node. The *node_ID* is randomly generated by the bootstrap terminal, which also verifies that the obtained pair <*prefix_ID; node_ID*> doesn't match some already active node.

The assigned ID expires immediately when a node leaves the network.

The random attribution of node IDs by a trusted bootstrap terminal allows avoiding a situation when a malicious node gets a specific ID in order to access to confidential data (masquerade attack).

The same prefix-based approach is applied to the process of key assignment to shared resources. Similarly to the procedure of node IDs assignment, we divide a key identifier in two strings of bits that represent the prefix_ID (the first β bits) and the key_ID (the rest $160-\beta$ bits). The key_ID can be obtained applying a hash function to the file content, as it is implemented in Kademlia. The prefix_ID of a resource usually coincides with the prefix_ID of the node that produced it. So, the prefix indicates the level of "confidentiality" of a file's content, i.e. if it is for common use or only for limited use of certain work-groups.

Resource management in the previously proposed solution was implemented in accordance with the privilege management scheme and with the corresponding read/write permissions. According to this scheme, nodes of the first (highest) level can get, modify, store and cancel files produced by nodes of any level. Nodes of each lower level (starting with the 2nd level) are enabled to access and modify files created/hosted by peers of the same-name level and other inferior levels as well.

Finally, nodes of the lowest level n can access and modify data produced only by peers of the same level. At the same time, some types of data should be accessible and shared by all users (for example in enterprise environment such data could include administrative circulars, recommendations, instructions, etc).

According to these access rules a prefix-based node ID mechanism was introduced where β bits out of n bits of the ID are used for a group and permission-driven prefix. This leads to a strong correspondence between the level of data confidentiality (represented by the prefix ID) and their actual location (identified by the resource key):

$$dist(ID, key) < 2^{n-\beta}$$

This inequality imposes an upper bound to the distance between a key and possible target nodes. Likewise, if the identifiers of two nodes satisfy this condition, they reside at the same sub-tree and belong to the same work-group. We call such distribution of data access rights - strict scheme for hierarchical resource management (Figure 1). Such scheme is suitable for environments with a rigid hierarchy and strongly defined policies like enterprise networks described in our previous work [6].

In accordance with the strict scheme, nodes of different levels have at their disposal identifier spaces of different dimensions to implement data storage procedures: it looks like using different zones of the "global" DHT by different groups of users. In terms of Kademlia, the DHT zone for nodes that handle resources with the highest level of confidentiality is limited by the same-name sub-tree regarding publishing data.

To avoid such limitations we propose a new resilient resource management scheme that will be described in the next section.

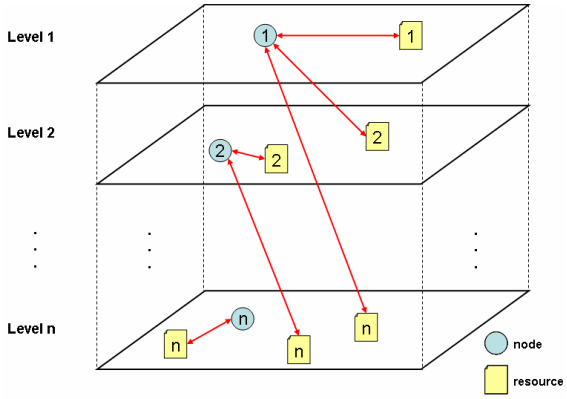

Figure 1. Strict scheme for hierarchical resource management

4. NEW RESILIENT RESOURCE MANAGEMENT SCHEME

Due to the constraints imposed by the confidentiality-based resource management scheme described above, in case of equal-distribution of the number of nodes and resources for different priority levels, the number of potential locations (target nodes) for resources of lower levels is significantly larger than for those of higher levels. Hence, nodes and resources of the highest level enjoy less redundancy. This fact can affect the system's fault tolerance in the case when a large percentage of the nodes of the highest level are off-line.

To enforce the fault tolerance and resilience of the system in such situations we propose a flexible scheme for group resource management (Figure 2), supported by a hierarchical and policy-based data access mechanism.

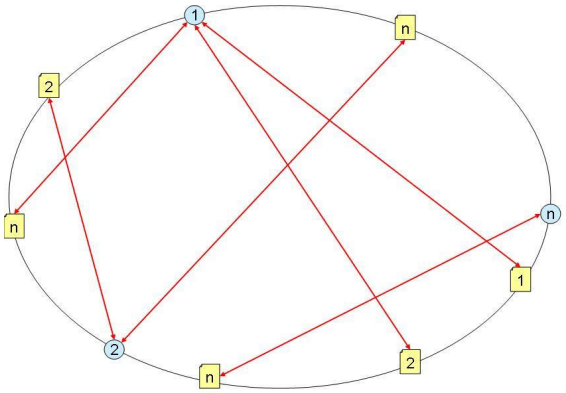

Figure 2. Flexible scheme for group resource management (unique DHT)

This scheme provides the possibility to exploit the entire DHT space by peers of any level for the publication of meta-data according to the original DHT principles. In Kademlia (as well as in other DHTs) a "value" associated with a certain "key" can contain either the resource itself (preferably small size data) or just some information about its location and some resource description (meta-data). Our proposal follows the latter approach that has the double advantage to limit the amount of data stored and managed by the DHT and to let network nodes to maintain DHT information (key-to-meta-data mapping) regardless the particular access level of the specific resource associated to such information. This in turn increases the number of nodes that collaborate on the maintenance of the DHT, and hence strengthens the DHT robustness. The absence of any strict rule for computing and assigning identifiers lets nodes and resources be arranged uniformly into a unique DHT, instead of forcing nodes or resources to be located at specific zones of the DHT space. Note that in this way the DHT is strictly used as distributed location registry rather than an actual storage system.

We propose to use the Kademlia DHT to store and retrieve meta-data information about the resources in preference to implementing a distributed file system, such as Oceanstore. This choice is due to a series of reasons.

First of all, storing actual data directly into the DHT implies data "migration" from node to node as the overlay reorganizes itself when nodes join and leave. The network traffic caused by data transfer can be significant if the churn rate is high and stored resources are numerous. Therefore, in order to avoid the network overload, sizes of data being passed from node to node should be small. Moreover, in a flexible scheme for distributed collaborative environment like the one we propose, it might happen that a resource belonging to some working group has been stored by a node that does not belong to that group. Therefore, a security issue arises, since the hosting node could potentially access the actual data even if it has not the right to access that resource. Using meta-data information in place of the actual resource avoids this issue. Location information contained by the meta-data are then used only by an authorized user to discover and access the actual resource.

Finally, meta-data can contain additional information about the resource, such as the publisher information, a last-update timestamp, and a signature (hash of the resource content signed with the publisher's private key, used for resource authentication and integrity check).

We chose to represent such meta-data information in XML format. An example of such XML-formatted data is the following:

```
<resource>
<key>a0b1c2d3e4…</key>
<name>http://www.mynet.org/docs/file.pdf</name>
<uri>http://80.10.1.2:8080/www/docs/file.pdf</uri>
<publisher>ff0123bcae…</publisher>
<access_rights>12</access_rights>
<expires>MON Oct 06 09:00:00 GMT 2008</expires>
<updated>MON Jun 08 09:00:00 GMT 2008</updated>
<signature>fabc234678…</signature>
</resource>
```

Therefore the DHT is used as a location registry that can be used to discover where the desired resource can be found. This information is specified in the `uri` tag which contains a routable URI containing in turn HostPort information (in order to by-pass any centralized DNS system) and other resource-related information useful for characterizing and/or accessing the resource.

Note that this approach, combined with the 'data:' URI scheme (RFC 2397) potentially allows the achievement of a persistent storage system. In fact, such URI scheme allows to store data directly inside the URI that in turn is included in the meta-data stored within the DHT and maintained by the DHT regardless the publishing node has left the overlay. The URI contains the data itself and not information on the location. Hence, the publishing node can leave the overlay while the data remains in the DHT. Obsolete data must be removed explicitly by the publisher or updated through successive `put()` operation. Obviously, this feature should be used only for small size data, that is, if the 'data:' URI is not excessively long, in order to limit the overall amount of traffic spent for the DHT management operations.

All DHT management operations are protected by some certificate-based security mechanisms described in the next section.

5. CERTIFICATE-BASED DHT PROTECTION

The proposed system architecture and all peer communications are protected through some security mechanisms based on certificate-based security architecture. Particularly, certificates are used to:

1. authenticate (signing) meta-data containing information about the location, the publisher, the access rights of a resource; this allows at any time a receiving peer to verify the resource's contents through the check of the given signature;
2. ensure resource data protection when the actual resource is accessed after the lookup mechanism has been succeeded;
3. control that meta-data are available only to nodes that are granted the appropriate access privileges;
4. protect the DHT against poison attacks, both for resources and for routing table entries;
5. optionally secure all DHT operations by enforcing cryptographic confidentiality.

DHT protection is achieved through some modifications of the original Kademlia protocol RPCs that will be next described. Such changes present some drawbacks such as a higher number of exchanges messages and a higher processing time to compute and verify the digests. However, the increased network bandwidth and computational power available nowadays make these problems trivial and totally acceptable in order to achieve a highly resilient and fault-tolerant system.

We assume that there is one central Certification Authority (CA) called DHT CA, and as many CAs as the number of work groups that belong to the DHT. The DHT CA is responsible for issuing certificates for the group CAs. User certificates are issued by group CAs.

Certificates are used for protecting i) the DHT, by authenticating DHT messages as described below, and ii) the resources, by authenticating and securing communications for the actual resource access (e.g. file transfer, copy, etc.).

User certificates are used during the bootstrapping phase and subsequently during each operation inside the DHT. User certificates are basically X.509 certificates with the addition of those extensions needed for group management.

We also assume that each group has some super-nodes that serve as bootstrap nodes, which are nodes with the following features:

- they are always on;
- they are active elements of the distributed system and therefore participate in the storage of resources.

A node receives its certificate and identifier from the CA of the group it belongs to. The bootstrap node first verifies the identity of the joining node. Identifiers are assigned by CAs. The process joining is basically divided into the following steps:

1. the bootstrap node verifies the certificate of the joining node and checks if it belongs to the same group;
2. the bootstrap node also verifies that the node ID is not already in use by performing a PING request for the ID, thus preventing Sybil attacks.

If a joining peer contacts a bootstrap node responsible for another group, the bootstrap node, after verifying the peer's credentials, it will redirect the request to the appropriate bootstrap node responsible for such group. Bootstrap nodes should then know how to contact other bootstrap nodes responsible for the other groups; this can be achieved through different mechanisms, such as pre-configured list of nodes, caching, node discovery service, etc.

Certificates are also used to protect (ensuring authentication and optionally confidentiality) all successive DHT requests. Encryption and decryption are performed by using the public-key RSA algorithm. In general, to ensure DHT protection, all non-idempotent procedures (procedures that cause changes in the DHT structure) should be protected. In Chord, for instance, all RPCs but *get()* should be protected. In Kademlia, all RPC are non-idempotent since all messages are used to update the k-buckets of the receiving peers. Therefore, the original Kademlia RPC protocol has been properly extended in order to protect also the following RPC requests: PING, FIND_NODE, FIND_VALUE (a.k.a. DHT GET), and STORE (DHT PUT). Mutual authentication is implemented in order to authenticate the server-side node as well (Figure 3).

Cryptographic confidentiality could be optionally provided in the DHT if the needed security level would require doing so.

Another security feature that can be optionally provided is a privilege-based access policy for DHT data access. Certificates are used to verify the identity of the requestor and to learn about the requestor's rights. Requests are fulfilled only if the authentication process succeeds and if the request pertains to a resource whose access rights are suitable for the user's credentials (access rights are defined in the meta-data stored in the DHT and associated to the various resources).

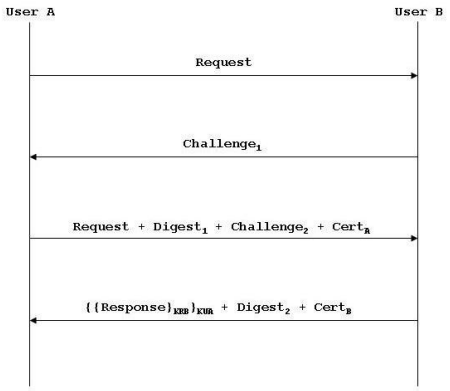

Figure 3. Authentication mechanism inside DHT requests

The first time a node wants to send a DHT request to another node (for example to issue a DHT GET request) it sends such request without any specific credential. When the target node receives such request it responds with an authorization request message including an authorization challenge together with information regarding the DHT realm. The requestor then resends the request together with its user certificate, group certificate, and the challenge response computed with the challenge, the DHT realm, the user and group IDs, and the user private key associated to the user certificate (it just encrypts with its private key the message digest of a combination of the challenge, DHT realm, user, and group IDs). It also adds to the request a client challenge used for server node authentication. When receiving such new request, the target node uses the DHT CA certificate to verify the group and user certificates, and then uses the user public key to verify the challenge response (it actually decrypts the challenge response and compare it with the message digest computed on the same combination of information). It then calculates the response to client challenge (in the same way of the requestor node) and includes it to the response message together with its user and group certificates and with a new "next" challenge. When the requestor receives such response, verifies the message, optionally stores the new "next" challenge, and keeps on with the DHT algorithm. The "next" challenge is provided to the requestor in order to speed up eventual successive request messages sent to the same target node: for successive requests the requestor may try to use such challenge for composing an authenticated request, thus reducing the total number of needed messages from four to two. Although such four (or two) messages authentication can be forced for each DHT request, in order to speed up the DHT operation a last hop authentication could be preferred, which means to challenge only the (last) request to the node that is actually responsible for the targeted resource (the meta-data associated to the resource). However, if this approach is used, some attacks like message hijacking or DoS might occur, since the verification is performed only at the end of the procedure and intermediate steps are blindly trusted. To avoid this, the challenge-digest verification should happen at each hop.

The mutual authentication mechanism proposed introduces an overhead in the execution time of each RPC. This overhead is due to the fact that each RPC consists of four messages instead of two and cryptographic operations must be performed on each node to perform authentication. We can split the execution time of each RPC into the following components: the round-trip time of message delivery (T_{RT}), the request processing time (T_P), the digest computation and verification time (T_{DC} and T_{DV} respectively). The original Kademlia RPCs take a total time:

$$T = T_{RT} + T_P$$

The new authenticated versions of the Kademlia RPCs take a total time:

$$T' = 2T_{RT} + T_P + 2T_{DC} + 2T_{DV} = 2T_{RT} + T_P + 2T_{RSA}$$

This means that the authentication mechanism introduces an overhead:

$$O = T' - T = T_{RT} + 2T_{RSA}$$

In our work, we made some simulations to measure the incidence of cryptographic operations. Our tests were made on a Pentium4 3 GHz with 1 GB RAM machine. Our simulations, with our Java-based implementation of RSA operations, showed that the aggregate time for RSA digest computation and verification is in average (on 1000 iterations) approximately 70 ms (T_{DC} = 69 ms, T_{DV} = 1 ms).

We can conclude that if the T_{RT} is prevalent (i.e. $T_{RT} >> T_{DC}$) then the overhead can be approximated by another T_{RT}, while in networks with little T_{RT}, such as wideband enterprise networks, the overhead is only due to the cryptographic operations, which are however quite fast. Therefore, the overhead introduced is absolutely acceptable if compared with the achieved resilience and security.

Note that if the authentication fails, then the request is not propagated nor iterated in the overlay and no additional time is spent by the DHT.

Confidentiality can be optionally introduced by performing key establishment through Diffie-Hellman algorithm secured through certificate-based signing and by encrypting data with such transaction key (ensuring both authentication and confidentiality). Certificate-based authentication prevents also from some typical DHT attacks [11, 12] leaded by unauthorized malicious users such as Sybil attacks. In fact, with the proposed operation the identity of a user is strictly related to the certificate and certificates are issued only to users that are granted to participate to the DHT.

6. ENFORCING APPLICATION-LAYER SECURITY

Authentication enables DHT protection, that is, requests are routed in the DHT only if the credentials of the requestor and target are certified and the corresponding messages are verified.

Although this mechanism prevents unauthorized users to request information about the location of resources for which they does not have proper rights, this does not prevent low level users to obtain location information about resources that they are maintaining meta-data information within the DHT according to the DHT algorithm. Moreover, since encryption of DHT RPC responses has not been made mandatory, eavesdropping could be still used to access to DHT data and hence to resource location information. For this reason a proper certificate-based security mechanism must be also introduced at application level in order to protect resources from illegal access. Although the definition of a specific authentication and encryption protocol used at application level is out of the scope of this document, in the next section is proposed a possible implementation in which both DHT RPC protocol and application resource access protocol use an extension of Session Initiation Protocol (SIP) [13] with a certificate-based authentication mechanism. The overall operation is summarized in Figure 4.

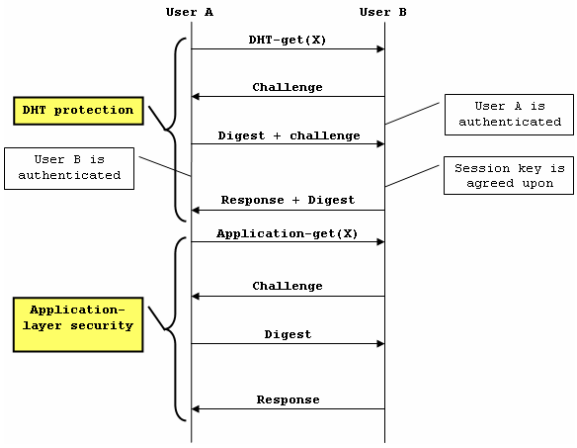

Figure 4. Application layer security

7. IMPLEMENTATION

The proposed distributed collaborative mechanism has been implemented in a demonstrative testbed. Such implementation comprises all mechanisms for allowing a network node to join a DHT, publish a resource in the DHT (DHT put), search for a specific resource (DHT get) for which the user has right access credentials, and access the selected resource by getting if from the location obtained by the DHT. Hence, the implementation includes both the protocol for DHT maintenance and the protocol for resource access according to the proposed security mechanisms.

Both protocols have been based on the SIP (Session Initiation Protocol) specified by the IETF [13]. SIP is an extensible application-layer UDP-based signalling protocol that permits to initiate, modify, and tearing down multimedia sessions in a peer-to-peer fashion. Particularly, for the DHT maintenance an extension of SIP based currently proposed within the IETF P2PSIP Working Group and called dSIP [14] has been used. Such protocol was designed in order to allow for compatibility with any DHT algorithm. Support for the Chord DHT was also defined. A specification for Kademlia has been defined in [15] and it is the protocol used in the implementation.

The resources are accessed via other protocols, opportunely extended with the proposed certificate-based authentication mechanism. Currently only a SIP based mechanism has been already implemented by using a new GET message (similar to the HTTP GET [16]) to fetch specific resources. The implementation has been based on Java technology. The DHT maintenance (dSIP [14]) and resource access protocol have been realized based on the SIP implementation provided by the MjSip open project [17]. Certificate issuing and authentication aspects are based on the SIP digest-authentication scheme proposed in the SIP RFC and currently being implemented and tested.

8. CONCLUSIONS

In this paper we presented a new resilient resource management scheme based on Kademlia lookup mechanisms for distributed collaborative environments. The proposed solution combines an efficient and robust lookup mechanism based on Kademlia DHT with a certificate-based authentication and authorization scheme for securing resource access in collaborative environments consisting of different work-groups. We also proposed to apply meta-data-based storage mechanisms to cope with resilience problems caused by the dynamical nature of P2P environment. The described approach significantly differs from that proposed in our previous work: in the new one network nodes exploit the entire DHT ID space for resource management instead of realizing storage/retrieval mechanisms only within the groups they belong to, in accordance with confidentiality principles. So, we tried to extend and generalize our solution in order to address some common problems regarding resilience and fault tolerance of distributed collaborative systems based on DHTs.

To enforce DHT protection, a robust two-way authentication (and optionally confidentiality) mechanisms have been introduced. Moreover, in order to avoid problems regarding unauthorized data accesses at the application-layer we proposed to apply specific certificate-based authentication and authorization mechanisms according to the rules specified for the DHT layer. Finally, the described mechanisms for collaborative resource management and data access have been implemented in a demonstrative testbed. These functionalities have been realized through a properly designed communication protocol based on SIP.

9. ACKNOWLEDGMENTS

This work has been partially supported by the Italian Ministry for University and Research (MIUR) within the project PROFILES under the PRIN 2006 research program.

10. REFERENCES

[1] Balakrishnan H. et al., “Looking Up Data in P2P Systems”, Communications of the ACM, Vol. 46, No. 2, pp.43-48, Feb. 2003.

[2] Stoica I., Morris R., Karger D., Kaashoek M.F., Balakrishnan H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proceedings of SIGCOMM-2001, San Diego, California, USA, August 2001.

[3] Maymounkov P., Mazières D., “Kademlia: A Peer-to-peer Information System Based on the XOR Metric”, In Proceedings of the 1st International Workshop on Peer-to-peer Systems, MIT, March 2002.

[4] Rowstron A., Druschel P. 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), March 2002

[5] Zhao B.Y., Huang L., Stribling J., Rhea S.C., Joseph A.D., Kubiatowicz J.D. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE J-SAC, 22(1), January 2003.

[6] Fedotova N., Fanti S., Veltri L. Kademlia for Data Storage and Retrieval in Enterprise Networks. In Proceedings of the Third International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom-2007), November 12 -15, 2007 – New York, USA.

[7] Rowstron A., Druschel P. Storage management and cashing in PAST, a large-scale, persistent peer-to-peer storage utility. In Proceedings of the 18th ACM Symposium on Operating systems Principles, October 2001

[8] Dabek F., Kaashoek M.F., Karger D., Morris R., Stoica I. Wide-area cooperative storage with CFS, In Proceedings of the 18th ACM Symposium on Operating systems Principles, October 2001

[9] Kubiatowicz J. et al. OceanStore: An Architecture for Global-Scale Persistent Storage”. In Proceedings of ASPLOS’2000, Cambridge, Massachusetts, USA, November, 2000

[10] Stribling J., Councill I.G., Li J., Kaashoek M.F., Karger D., Morris R., Shenker S. OverCite: A Cooperative Digital Research Library, In Proceedingd of the 4th International Workshop on P2P Systems (IPTPS05), February 2005

[11] Sit E., Morris R., “Security considerations for Peer-to-Peer Distributed Hash Tables”, in Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS’02), Cambridge, March 2002.

[12] Douceur J.R. The Sybil Attack, In Proceedings of IPTPS-2002, March 2002

[13] Rosenberg J., Schulzrinne H., Camarillo G., Johnston A., Peterson J., Sparks R., Handley M., Schooler E. "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[14] Bryan D. "dSIP: A P2P Approach to SIP Registration and Resource Location", IETF Internet Draft draft-bryan-p2psip-dsip-00, February 2007.

[15] Cirani S., Veltri L. “A Kademlia-based DHT for Resource Lookup in P2PSIP”, IETF Internet Draft draft-cirani-p2psip-dsip-dhtkademlia-00, October 2007

[16] Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, June 1999.

[17] MjSip project, http://www.mjsip.org.