Detecting and Excluding Misbehaving Nodes in a P2P Network

François Lesueur Ludovic Mé Valérie Viet Triem Tong*
SUPELEC, SSIR Group (EA 4039)
Avenue de la Boulaie - CS 47601 - 35576 Cesson-Sévigné cedex - France
E-mail: `firstname.lastname@supelec.fr`

Abstract

Given their fully distributed architecture, P2P networks allow the design of low cost and high availability systems but also pose new security problems. In these collaborative networks, security properties need to be ensured by the participants themselves. In this paper, we propose to detect and exclude misbehaving nodes to allow honest participants to enforce security properties. The proposed scheme is fully distributed.

Keywords: *P2P, Security, Behavior, Detection, Exclusion*

Introduction

P2P networks have been widely used for the last few years to design low cost and high availability systems. Indeed, P2P networks are fully distributed systems composed of many nodes (ordinary PCs) provided with replication mechanisms, allowing high availability. File sharing applications are well known (Gnutella [4]), but these networks can also provide distributed file systems (CFS [5]), publish/subscribe applications (Meghdoot [7]), multicast (SplitStream [3]) or Voice over IP (P2PSIP [2]).

There are two types of P2P networks: unstructured ones and structured ones. In unstructured P2P networks (Gnutella [4]), requests are broadcasted or routed through random walks. In structured ones (Chord [12]), requests are routed efficiently using generated routing tables. We consider here structured P2P networks security.

Structured P2P networks provide a virtual space called *overlay*. Each node (PC) is uniquely identified by a $nodeId \in \mathcal{KeyIds}$; in the same way, each *resource* (file, ...) is uniquely identified by a key identifier $keyId$, $keyId \in \mathcal{KeyIds}$ (for a file, the key is usually its SHA1 fingerprint). Nodes and resources thus share the same identifier space $\mathcal{KeyIds}$, which is finite but supposed large enough (often 2^{160} elements due to the size of SHA1 hashes). Each node is responsible for the management of a part of the resources. The overlay provides routing facilities with a logarithmic cost for a node to access a specific identifier.

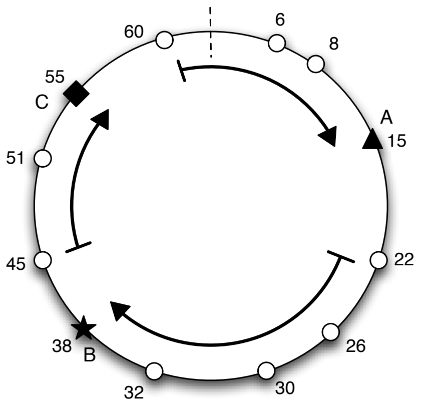

Figure 1. Simplified representation of Chord with identifiers going from 0 to $2^6 - 1 = 63$. ▲, ♦ and ★ are nodes (PC) and ○ are resources (files). Node A, which $nodeId$ is 15, is responsible for resources 60, 6 and 8.

In Chord for instance, the identifiers space goes from 0 to $2^{160} - 1$ and each node is responsible for all the resources which identifiers are between its identifier and the one of its preceding node in the overlay. This organization is illustrated in Figure 1.

Finally, a Distributed Hash Table (DHT) such as DHash [5] allows to *put* persistent resources in the P2P network. Each resource is replicated on some following successors of the responsible node to tolerate churn (nodes leaving or crashing) and any user can then *get* this value.

To the contrary of classical networks in which security is usually provided through a trusted centralized entity, P2P networks are collaborative networks in which each mem-

*The authors thank Orange Labs for partially funding this work and especially Hervé Debar for his involvement in the project and his advice and valuable comments.

ber is responsible for a part of the global system. To resolve security problems, impact of malicious nodes has to be minimized.

We propose thus a scheme to identify users and then detect and exclude misbehaving members in a P2P network. The first part, presented in [11], allows users to obtain a unique provable identity and protects against the sybil attack [6]. An accepted node obtains a unique membership certificate. The second part, presented in this paper, is a distributed misbehavior detection and node exclusion system, allowing to exclude misbehaving nodes from the network through the agreement of a static ratio of the nodes and without an administrative authority. Exclusion is materialized by the revocation of the membership certificate and the revocation process is handled using the distributed certification we presented in [10] (revocation is juste a special case of certification). In this system, the normal behavior is dynamically determined by each node using static specifications defining how to compare different nodes behaviors: a node is detected as misbehaving if it shows a minority behavior. We thus make the (reasonable) assumption that there is a majority of honest nodes in the network.

We present in Section 1 the related work. In Section 2, we present the distributed certification we use to revoke membership certificates. In Section 3, we overview the detection and exclusion of misbehaving nodes. In Section 4, we detail the detection part and in Section 5, we precise the exclusion part. In Section 6, we analyze experimental results and we finally conclude and suggest some future work.

1 Related Work

In this section, we present work related to the evaluation of nodes in P2P networks. We first present work on admission control to a peer group and we then introduce reputation systems which allow to assess other nodes.

1.1 Admission Control to a Group

Kim *et al.* proposed in [9] to limit the number of attackers in peer groups (P2P networks but also ad-hoc networks) through admission control to the network. Admission control is based on a *Group Charter* defining conditions to enter the network. A newcomer's access is then granted through a voting system among the members. Their scheme uses threshold cryptography to allow a fixed number of members accepting the newcomer to sign his membership certificate.

However, defining accepting conditions in P2P groups seems unrealistic since P2P systems are naturally open and large networks. Since open, everyone should be able to access the network; since large, we cannot rely on a significant part of the members knowing each newcomer. A newcomer can only be known by a very slight fraction of the members, and so relying on this very slight fraction of members to accept this newcomer would allow malicious users to easily join the network. Limiting the number of attackers in P2P networks can thus only be a reaction to misbehaving members, rejecting such members after bad behavior, and not a prevention.

1.2 Reputation Systems

Precisely, reputation systems [8] assess nodes behavior and then apply relevant rules. Each node attributes each other node a given reputation, computed from local observations and possibly recommendations. Reputation systems allow to favor good members over selfish ones.

However, even using recommendations, reputation systems are not tailored to exclude attackers after only one attack. If a honest node N detects an attacker A, N can stop communicating with A but other nodes will continue to communicate with A. In fact, other nodes cannot be sure that N is honest and so need other testimonies. Thus, an attacker attacking only a few nodes can be seen as honest by most of the nodes, whereas these few nodes might be in fact the replicas of an attacked resource. Some actions such as fake responses should lead directly to the attacker's exclusion, as soon as a single node detects it. Moreover, reputations being computed locally by each node, different nodes may locally have very different reputations for the same node: then, if this node owns a given resource and if some nodes decide to ignore this bad-reputation node, these different honest nodes may have an incoherent view of the P2P network. We propose thus in this paper a complementary system which allows to directly and globally exclude some types of attackers.

2 Distributed Certification

As stated in the introduction, we use the distributed certification scheme we proposed in [10] to revoke membership certificates (revocation is a special case of certification) and we thus briefly describe this scheme in this section. In this system, signing some data requires the collaboration of a fixed ratio t of the number of nodes. This ratio is enforced using a fully distributed scheme which tolerates misbehaving nodes and thus complies with the P2P basics. To our best knowledge, this is the only scheme based on the cooperation of a fixed ratio of nodes instead of a fixed number of nodes, which is mandatory in varying size P2P networks.

The network is characterized by an RSA public/secret key pair (P, S), $P = (d, m)$ being publicly known and $S = (e, m)$ being shared among the nodes (no node knows S entirely). This key pair is originally generated by founding members using a distributed algorithm as Boneh and

Franklin proposed in [1]. The network is decomposed in s *sharing groups*, each group knowing one share e_i of S such that the sum of all the different shares equals e. In this case, the RSA signature of some data d is

$$d^e[m] = d^{\sum_{i=1}^{s} e_i}[m] = \Big(\prod_{i=1}^{s} d^{e_i}[m]\Big)[m]$$

In other words, the RSA signature of d with S, which is $d^e[m]$, is equal to the product of the partial signatures with each share modulo m.

Signing a certificate requires then every share and thus the collaboration of one node of each sharing group. Given g_{min} and g_{max} the minimal and maximal sizes of sharing groups, the ratio t of nodes needed to sign a certificate verifies $\frac{1}{g_{max}} < t < \frac{1}{g_{min}}$ and thus, sharing groups can split (resp. merge) when nodes join (resp. leave) with only local knowledge to enforce this ratio t.

Each share is uniquely identified by a binary $shareId$ and is only known by nodes which identifiers are such that $nodeId = shareId*$ in binary form (i.e., $shareId$ is a binary prefix of $nodeId$). Splitting a group knowing e_i (resp. merging two groups knowing e_{i0} and e_{i1}) creates two groups knowing respectively e_{i0} and e_{i1} (resp. one group knowing e_i) such that $e_i = e_{i0} + e_{i1}$. This distribution is illustrated in Figure 2. If a share e_i is exposed, the group knowing e_i can *refresh* it: a random value Δ is added to e_i and subtracted from another randomly chosen share. After a refresh, the sum of all the shares is still the same (hence the certification still works) but the old value of e_i is useless (out-of-sync with the other shares).

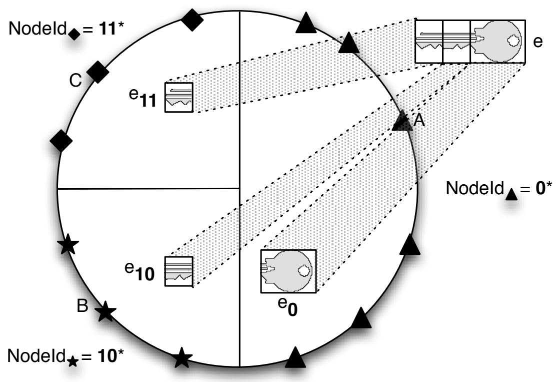

Figure 2. Distribution of three shares e_0,e_{10} and e_{11} with $e = e_0 + e_{10} + e_{11}$. Each node knows the list of members of its sharing group.

When a node A which identifier starts with 0 wants to sign some data d with $S = (e, m)$, A recursively partially signs d with (e_0, m) and asks a node B which identifier starts with 1 to partially sign it with (e_1, m). B then partially signs it with (e_{10}, m) and recursively asks C to partially sign it with (e_{11}, m). A node stops the recursion when it is asked for its own share, in which case it returns $d^{e_i}[m]$; otherwise, the node returns the product of the two recursive calls. A finally gets the product of the partial signatures with all shares which is the signature with S.

In this scheme, colluding attackers may eventually obtain every share of S or control every node in a sharing group. The probability for a ratio k of colluding attackers to obtain every share of S is bounded by $\prod_{i=1}^{s} 1-(1-k)^{g_{max}}$ and the probability to control every node in any sharing group is bounded by $1 - \prod_{i=1}^{s} 1 - k^{g_{min}}$, s being the number of sharing groups. In a $10,000$ nodes network with $g_{min} = 20$ and $g_{max} = 40$, the probability to obtain every share is infinitesimal for $k < 0.2$ and the probability to control every node in a sharing group is infinitesimal for $k < 0.6$.

3 Detection and Exclusion

We propose to identify users and then to detect and exclude misbehaving members. Each node is first accepted in the network through a sybil-protection mechanism, as the one we proposed in [11]. An accepted node obtains a unique certificate proving its membership. Then, misbehaving nodes are automatically excluded to limit their number. Nodes are thus monitored and those showing a bad behavior are excluded. We observe here the behavior of nodes through sent messages. In the following, we present the detection and exclusion of misbehaving nodes.

The detection of misbehaviors is based on Observable Behavior Specifications (OBS) (see Section 4). We assume that the majority of the nodes are well-behaving and so misbehaviors are detected by difference from other nodes. Since misbehaving nodes are only a minority, they have a different behavior from the majority. Normal behaviors are thus deduced dynamically according to observed behaviors and proposed OBS.

We propose then to prove misbehaviors to lead to node exclusion. When a node detects a misbehavior, it generates a proof of this misbehavior. Such a proof must be verifiable by any other node in the network and should thus not contain local unverifiable information. Moreover, a misbehavior proof leading directly to node exclusion, an attacker should not be able to forge a proof. In the proposed system, proofs are based on exchanged messages.

This proof is then used to globally exclude the misbehaving node through the collaboration of a given ratio of the nodes (see Section 5). The certificate of the node, which is its membership proof, is revoked using the distributed certification algorithm presented in Section 2.

4 Detecting Misbehaviors

In this section, we present the detection of misbehaving members. We first detail the precautions needed to prove misbehaviors. Then, we define the Observable Behavior Specifications (OBS) used to compare different nodes behaviors and we illustrate them on three proposed attacks. Finally, we present the detection architecture.

4.1 Precautions to Consider

In our system, misbehaviors are proved and since a proof directly leads to the exclusion of the concerned node, an attacker should not be able to forge proofs. First, messages are signed in order to be unrepudiable and unforgeable. Then, we present in this section precautions needed to be able to judge the validity of a proof. We present *timeframes* allowing to check that observed behaviors are comparable and *decorellation* preventing an attacker from forging proofs with the help of accomplices.

4.1.1 Timeframes

Since P2P networks are dynamic, responses to the same request may be different over time. For instance, after the refresh of a share identified as $shareId$, the partial signature of given data with $shareId$ changes.

To distinguish such legitimate changes from misbehaviors, nodes are provided with a common clock. The needed precision depends on the resources dynamics, but if changes are not very frequent (refreshes are not so frequent for instance), a precision of a few minutes is sufficient. Nodes can initially synchronize their local clocks using NTP servers, an already deployed clock synchronization mechanism, and then rely solely on these local clocks.

Each response contains then two dates: the date since when this response has been valid (for instance the date when the share was last refreshed) and the date when the response has been sent. Dates are here used in the wide sense and contain days but also hours and minutes. The recipient accepts the message if and only if the emission date is the current date, tolerating a shift of a few minutes. Dates allow then to compare messages which should be the same.

4.1.2 Decorellation

Since nodes detect misbehaviors by the comparison of several responses to the same request, an attacker could forge a misbehavior proof by requesting some accomplices and one honest node: this honest node would exhibit a minority behavior and so would be detected as misbehaving.

To prevent this attack, the set of compared nodes is constrained: nodes used to generate a misbehavior proof must have uniformly distributed node identifiers among all the nodes eligible for the request. Since attackers have truly random identifiers (otherwise, this would be a sybil attack and the identification of users to enter the network prevents the sybil attack), the ratio of attackers in the constrained set of nodes is the same as in the network, hence a minority.

4.2 Format of OBS

The behavior of a node is reduced to its observable part which is the messages it sends. Moreover, the misbehaviors are detected by the comparison to other nodes behaviors. Observable Behavior Specifications define thus how to compare messages of several nodes to detect misbehaviors.

We consider here that messages are composed of the $nodeId$ of the sender node, an operation, a payload and a timeframe. The payload itself is composed of several fields depending on the type of message and is independent of the sender (all honest nodes reply with the same payload for a given request). The proposed scheme compares thus the payloads of the messages to detect misbehaviors. The timeframe is composed of two dates defining since when this response has been valid and when this response has been sent. This format of message is illustrated in Figure 3 which represents more precisely an attack against signature (see Section 4.3.1).

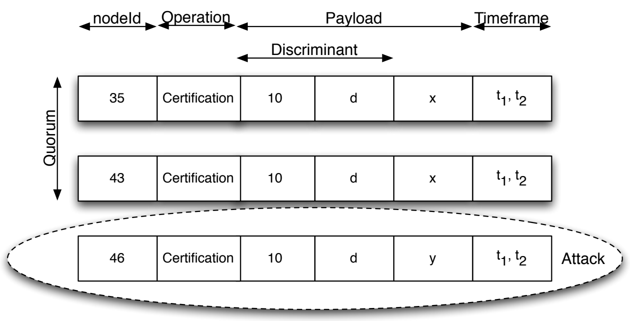

Figure 3. Three messages showing an attack against signature by the node 46. The two messages from 35 and 43 have the same payload and form the quorum. The message from 46 has the same discriminant and hence should have the same payload if 46 was honest; however, the payload is different and 46 is detected as an attacker.

An OBS first specifies the attacked *operation*. In Figure 3, the three messages concern a Certification operation.

An OBS then describes the *direction* of attack which defines if the attack is in the request for an operation or in the response.

	Possible values	Description
Operation	Certification | Obtention ...	Attacked operation
Direction	Request | Response	Attack is in the request or in the response
Discriminant	Field names	Equal fields among compared messages
Quorum	$\mathbb{N}$	Number of messages to deduce valid result
Scope	Network | Replicas | Prefix	Scope of the operation

(a) Format of OBS

Attack	Operation	Direction	Discriminant	Quorum	Scope
Signature	Certification	Response	$shareId, data$	$\frac{nbAsks+1}{2}$	Asked $shareId$
Newcomers	Obtention	Response	$\emptyset$	$\frac{2}{3}g_{min}$	Sharing Group
Availability	Merge|Split|Refresh	Request	$\emptyset$	$\frac{2}{3}g_{min}$	Sharing Group

(b) Defined OBS

Figure 4. Observable Behavior Specifications

The *discriminant* defines the fields of the payloads which must be equal to consider that the payloads should be the same, if all the nodes were honest. The discriminant is composed of fields on which attackers cannot cheat, for instance representing the request they are answering: attacks consist thus in biasing the other fields of the payload. In Figure 3, the discriminant is composed of two fields defining the partial signature which was asked and the three presented messages have the same discriminant: these three messages should be the same if all nodes were honest.

The *quorum* defines the number of identical payloads needed to consider that the contained value is the valid one. In Figure 3, if we consider an OBS with a quorum of 2 and the proposed discriminant, then the messages from nodes 35 and 43 prove that the valid payload is composed of 10, d and x. Then, the different payload sent by 46 is considered as an attack (same discriminant but different payload).

Finally, the *scope* defines the set of eligible nodes for the request. This set can be the whole network, a set of nodes (for instance k replicas) or the nodes sharing a given prefix (for a partial signature in distributed certification). This *scope* allows to define the set of identifiers in which the compared nodes representing the *quorum* have to be uniformly distributed and enforces then the decorellation.

These fields are summed up in Figure 4(a).

4.3 Detecting Some Proposed Attacks

We plan on detecting attacks on any application. However, we start by detecting attacks against the distributed certification since the exclusion mechanism presented in this paper relies itself on this distributed certification. We propose then in future work to detect attacks against other P2P services such as DHT.

We describe thus here three different attacks against the distributed certification and present their associated OBS in Figure 4(b).

4.3.1 Attack Against Signature

In the distributed certification presented in Section 2, an attacker can return a wrong partial signature when asked to sign with its own share or return a wrong product otherwise. To prevent this attack, each node asks $nbAsks$ nodes instead of one and uses the majority result; however, it is crucial to limit these attacks to improve the performance (asking several nodes is expensive) and to prevent attackers from becoming a majority, in which case the certification would not work anymore. The payload of a response contains three fields: the share identifier which was asked ($shareId$), the data which has been signed ($data$) and the partial signature returned ($psig$). This attack is the one illustrated in Figure 3 with $shareId = 10$, $data = d$ and $psig = x$ or y.

To detect the attack against signature, we propose to compare the $nbAsks$ responses ($nbAsks = 3$ in the figure). This attack consists in sending a fake response to a partial signature request and so the *operation* is Certification and the *direction* is response. This attack resides in replying a fake $psig$ (y instead of x in the figure) for given $shareId$ and $data$ and so the fields which have to be equal among compared payloads (*discriminant*) are $shareId$ and $data$. The *quorum* is fixed to $\frac{nbAsks+1}{2}$ which means that a payload is considered valid as soon as the majority of the nodes returned it. The eligible nodes for a request are all the nodes which identifiers are prefixed by the share identifier asked and the *scope* is thus all these nodes.

When some attackers have been excluded, honest nodes automatically detect that all the partial signatures are valid and can thus dynamically reduce the number of nodes asked for each partial signature, in order to balance the correctness and the efficiency of the requests.

4.3.2 Attack Against Newcomers

When a newcomer joins the network, it has to obtain the part of the network secret key owned by its sharing group. If attackers send a fake share to this newcomer, it is then unable to participate in distributed certifications (it will even be detected as an attacker, since it will return wrong partial signatures). To protect from this attack, each newcomer asks g_{min} nodes for its share (remember that sharing groups are composed of g_{min} to g_{max} members). As for the Certification Attack, it is important to detect such attacks to prevent attackers from becoming a majority. The payload of a response contains three fields: the share identifier of the group ($shareId$), the value of this share ($share$) and the list of the members of the sharing group ($members$).

To detect the attack against newcomers, we propose to compare the g_{min} responses. This attack consists in sending a fake response to an obtention request and so the *operation* is Obtention and the *direction* is response. This attack resides in sending a wrong share identifier, a wrong value for the share or a wrong list of members of the group. Indeed, a newcomer has no information on the sharing group it should belong to, so an attacker can lie on these three fields and all the obtention payloads are thus comparable (*discriminant*= $\emptyset$). The *quorum* is fixed to $\frac{2}{3}g_{min}$ which means that a payload is considered valid as soon as $\frac{2}{3}g_{min}$ of the nodes returned it. On the one hand, since sharing groups are composed of g_{min} to g_{max} members, the quorum needs to be low enough to allow a group of g_{min} members to prove the misbehavior of one of them; on the other hand, the quorum needs to be high enough to prevent attackers from forming a quorum in a group of g_{max} members. A quorum of $\frac{2}{3}g_{min}$ seems a good tradeoff between these two extreme cases. The eligible nodes for a request are all the nodes of the sharing group and the *scope* is thus a single sharing group.

4.3.3 Attack Against Availability

An attacker can try to break the distributed certification by rendering a share unavailable, for instance by launching a Denial-of-Service attack against all the nodes of a sharing group. To achieve this, an attacker can launch unnecessary maintenance operations in the sharing group (requests to merge, split or refresh). These operations would not be accepted by the other members but would provoke agreement protocols which are expensive operations. The payload of a maintenance operation is a boolean $agree$ indicating if this node thinks that this operation is legitimate and should be launched.

To detect the attack against availability, we propose to compare the responses to a maintenance request. This attack consists in requesting a maintenance operation and so the *operation* is either Merge, Split or Refresh and the *direction* is request. This attack resides in asking for an illegitimate maintenance operation and an attacker lies thus on the $agree$ field: all the payloads for a given operation are thus comparable (*discriminant*= $\emptyset$). For the same reasons of sharing groups sizes as for the obtention attack, the *quorum* is fixed to $\frac{2}{3}g_{min}$ which means that a payload is considered valid as soon as $\frac{2}{3}g_{min}$ of the nodes returned it. The eligible nodes for a request are all the nodes of the sharing group and the *scope* is thus a single sharing group.

4.4 Detection Architecture

First, we overview the detection mechanism which is composed of the *classifier* and of the *checker*. Then, we detail the classifier and the checker.

4.4.1 Overview

To detect misbehaviors, each node monitors all the messages it sends and receives. This observer is placed between the applicative layer, represented here by the distributed certification mechanism, and the P2P overlay. The detection of misbehaviors is split in two parts: the *classifier* and the *checker* (Figure 5).

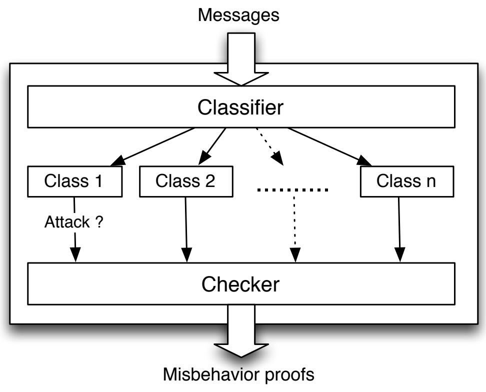

Figure 5. Detection and Exclusion Architecture.

First, the *classifier* (see Section 4.4.2) associates each sent or received message to other related messages in a *class*, according notably to discriminants. For instance, Figure 3 corresponds to a class of messages. Behaviors are then checked among messages of the same class.

When a message is added to a class, the *checker* (see Section 4.4.3) verifies if this class corresponds to a misbehavior. If a misbehavior is detected, a misbehavior proof accusing one or several nodes is created from this class.

This misbehavior proof is then used to exclude the misbehaving node (see Section 5).

4.4.2 Classifier

The *classifier* associates in a *class* messages which should be compared to detect misbehaviors. Comparable messages are the ones corresponding to a given OBS, having the same discriminant and sent during the same timeframe. Such messages should be identical and so a different message in a class indicates a misbehavior, which is detected by the *checker*.

Each new message is labeled by a set of triplets $\{(OBS, Discriminant, Timeframe)^*\}$ which is initially empty. First, a triplet is added to this set for each OBS defined on the operation of this message. Then, the discriminant of each of these triplets is set to the values found in the message, w.r.t. the discriminant of the related OBS. Finally, the timeframes are set to the timeframe of the message. For instance, the three messages of Figure 3 are labeled by the set $\{(Signature\ OBS, (10, d), (t_1, t_2))\}$.

Each new message is then associated in a class to every other message having a common triplet in its label. Two timeframes are considered common as soon as their intersection exists, i.e., they have a common period of validity. The three messages of Figure 3 form thus a class and should be identical if all the nodes were honest.

Each class is finally erased after a fixed amount of inactivity time.

4.4.3 Checker

When a message is added to a class, the *checker* applies the OBS to detect attacks in this class, attacks which correspond to differences in the payloads. If a misbehavior is detected, the checker creates the proof accusing one or several nodes.

A misbehavior proof is a minimal class proving the misbehavior. A proof contains the minimal number of messages representing the normal behavior, i.e., the quorum of the OBS triggered, plus the message(s) of the misbehaving nodes.

However, a misbehaving node may try to fool the system by sending a fake response at a date t and announcing this response is valid since $t - \epsilon$, ϵ being a short time. Given the imprecise clock used, it is not possible without further investigations to detect whether this is an attack or whether the valid response has just changed. In such a case, the checker waits a bit and then re-asks the nodes composing the quorum. If the suspected node is honest then the novel responses also take into account the change of the valid value; if the suspected node is misbehaving then the novel responses include the date t in their validity period and the misbehavior proof includes thus these new messages as well as the old message of the attacker.

The node which has detected the misbehavior uses then the proof to exclude the misbehaving node through the mechanism presented in the following section.

5 Excluding Attackers

The exclusion of an attacker is materialized by the revocation of its certificate. We first present how a certificate is revoked. Then, we explain the publication of the revocation and we finally detail what happens after an exclusion.

5.1 Certificate Revocation

A certificate revocation is of the form $\{Cert, revoked\}_S$, where $Cert$ is the membership certificate to revoke and $revoked$ is a flag indicating the revocation, all that being signed with the network secret key S. Certificate revocation uses then the same algorithm as distributed certification to obtain the signature with S. The node detecting the misbehavior provides the proof of the misbehavior to each node involved in the revocation.

Each involved node locally validates the proof. Each node checks the proof as if it was a local class of messages and also verifies that the quorum is constituted of nodes uniformly distributed in the scope of the operation (otherwise, an attacker would be able to ask some accomplices to bias the normal behavior). However, these nodes cannot be truly uniformly distributed since node identifiers represent only a small part of the identifiers space: chosen nodes are in fact the ones responsible for the identifiers which are uniformly distributed. During the revocation, each node has thus to check whether the proposed nodes are really responsible for these resources.

To be able to decide if a given node is close enough to a resource to be responsible for it, each node calculates the density of nodes in its neighborhood. Node identifiers being uniformly distributed using a hash function, this density is roughly the same in the whole overlay and a node can thus estimate the density of nodes in the overlay. Even if not so precise, this evaluation allows to constrain the set of nodes used in a proof.

If the proof is valid, this node proceeds with the distributed certification and the initiator node finally obtains the signed revocation if a given ratio of the nodes validated the proof. It distributes then this revocation to other nodes.

5.2 Revocation Publication

To effectively exclude the misbehaving node, the revocation must be notified to all nodes. To limit the communications, revocations are stored in the DHT and each node checks for their presence before communicating with an unknown node. If the identifier of the attacker is $nodeId$ then this revocation is put at $h(nodeId)$ (putting it at $nodeId$ would transiently put this revocation under the responsibility of the attacker). Given the replication mechanisms of a DHT, this revocation is finally stored on some successors of

$h(nodeId)$, which guarantees its availability even if one of the replicas is malicious.

Revocations are also directly sent to nodes already communicating with the attacker. These nodes are the members of the attacker's sharing group, the nodes which have the attacker in their routing table, the nodes which are in the attacker's routing table and possibly some nodes connected at the applicative level. Members of the sharing group of the attacker are easily found since the prefix of their identifiers is known (the attacker has the same prefix). The way to find nodes which have the attacker in their routing tables and nodes which are in the attacker's routing table depends on the overlay used. In Chord for instance, there are 2^{160} identifiers and the attacker identified by $nodeId$ has in its routing table the nodes succeeding the identifiers $nodeId + 2^{159}, nodeId + 2^{158}, etc.$ Nodes having the attacker in their routing table can be found in the same way. Finally, finding nodes connected to the attacker at the applicative level depends on the application. The distribution of revocations could also be handled through a *publish/subscribe* mechanism, each node interested in the hypothetical revocation of a given node subscribing to this feed and the revocation being published on this feed. All that being done, the attacker cannot communicate anymore with any node in the overlay.

5.3 Actions after an Exclusion

The exclusion of a node leads to modifications of the P2P overlay and to maintenance of the shared secret key of the network S. Modifications of the overlay are the same than when a node leaves the network: the attacker is replaced in routing tables and another node takes control of the resources it was responsible for.

Then, the share of the secret key which was known by this attacker must be invalidated. Indeed, this share being known by an attacker, it is considered exposed. Moreover, in the case of an Obtention Attack, the misbehavior proof used to exclude this attacker contains itself this share. This share is thus *refreshed* (see Section 2), operation in which this share is mixed with another one rendering old shares useless. Finally, since the group of the attacker may become composed of less than g_{min} members after the exclusion, this group may trigger a merge operation with the adjacent group.

6 Simulations

In this section, we present experimental results. These results are presented in the case of the attack against signature which seems the most dangerous. Indeed, this attack quickly decreases the performance of the certification system, certification system which is used to revoke misbehaving members. In the first part, we study the reliability of the detection system. In the second part, we show the impact of node exclusion in the lifetime of the system.

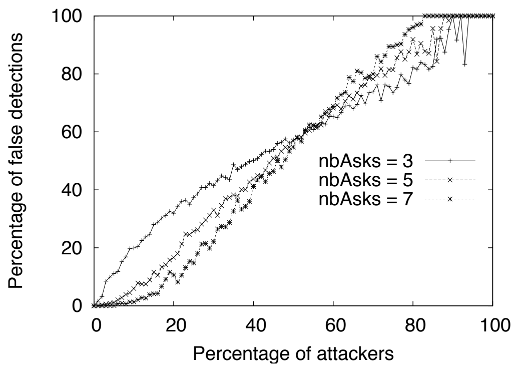

Figure 6. Percentage of false detections in function of the percentage of attackers. Each experiment contains 5000 nodes.

6.1 Detection System Reliability

Figure 6 shows the percentage of false detections (honest nodes detected as misbehaving) in function of the percentage of attackers. This figure shows a symmetry which corresponds to the change of the normal behavior. In the first half of the curve, attackers are a minority and so the normal behavior is the honest one; in the second half, attackers are a majority and so the normal behavior is the misbehaving one, yielding the detection of honest nodes as misbehaving. However, we consider that a ratio of 1 attacker for 10 nodes is already quite high. In such a case, the percentage of false detections is less than 5% when each node asks each partial signature to 5 nodes ($nbAsks = 5$). Irregularities with more than 90% of attackers are explained by the fact that attackers collude and never participate in the detection of one of them, whereas honest nodes might testify among themselves.

6.2 Impact of Exclusion

Figure 7 shows the percentage of the honest nodes which are wrongly excluded with $nbAsks = 5$. Honest nodes can be wrongly excluded if attackers are enough to return a (wrong) majority response. Even if some misbehaviors are not detected the first time, all attackers are finally excluded in the simulated configurations and the number of honest nodes which have been excluded is calculated when all attackers have been excluded. The first curve is drawn for networks composed of 5000 nodes with a percentage of attackers varying from 0 to 25%; the second curve is drawn

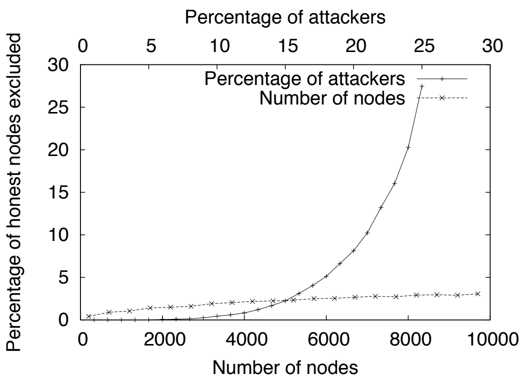

Figure 7. Percentage of the honest nodes which are wrongly excluded in function (top axis) of the percentage of attackers in a 5000 nodes network or (bottom axis) of the number of nodes with 15% of attackers. Each partial signature is asked to 5 nodes.

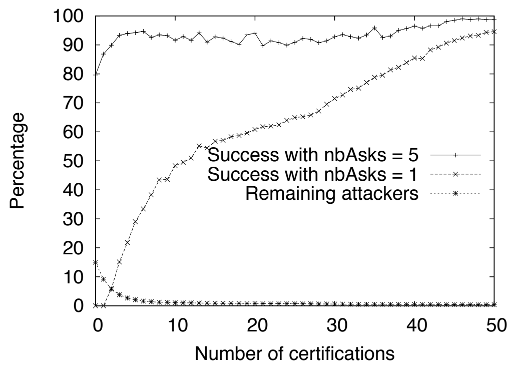

Figure 8. Percentage of success of the certification algorithm in function of the number of already done certifications. Each experiment starts with 5000 nodes and 15% of attackers. The curve with $nbAsks = 1$ corresponds to the certifications succeeding on the first call and the one with $nbAsks = 5$ to the certification succeeding with the fallback algorithm. The remaining percentage of attackers is also drawn.

for networks composed of 0 to 10000 nodes with 15% of attackers. The number of honest nodes excluded is low since only 2% of them are wrongly excluded in a 5000 nodes network with initially 15% of attackers.

In Figure 8, we illustrate the impact of exclusion on the success of certification operations. Each node asks first only 1 node and then 5 if the obtained signature is invalid. This figure shows the success of successive certifications and shows that with initially 15% of attackers in a 5000 nodes network, the algorithm with $nbAsks = 1$ is quickly usable after having used $nbAsks = 5$ a few times, this last value allowing to exclude misbehaving nodes. Success with $nbAsks = 1$ is nearly 90% after only 40 certifications. This figure also shows the remaining percentage of misbehaving nodes in the network after each certification. The percentage of misbehaving nodes drops from 15% to 1% in 10 certifications.

Conclusion

In this paper, we proposed to detect and exclude misbehaving nodes in a P2P network. The detection is based on the comparison of nodes behavior and the exclusion is handled through fully distributed algorithms.

Based on three use-cases, we precisely described the mechanisms to detect misbehaviors, to prove misbehaviors to other nodes and finally to exclude misbehaving nodes through revoking their certificates. The mechanism is extensible to detect attacks on other applications.

We finally experimentally studied the performance of the proposed system. These experimentations show that our system performs well in networks composed of up to 10% of attackers, which is a high percentage since these attackers are progressively excluded from the network thanks to our proposition.

We are now interested in evaluating the benefits of this security mechanism on a P2P application. In a DHT for instance, an attacker can return a wrong value for an asked resource. Since resources are replicated on several nodes, an expensive countermeasure is to ask several nodes, through different routes, for the same resource. With our detection and exclusion system, it is possible to detect the attackers which returned a wrong value, exclude them, and finally reduce the number of nodes asked for the same resource without sacrificing reliability.

References

[1] Boneh and Franklin. Efficient generation of shared RSA keys. In *Proceedings of the 17th Annual International Cryptology Conference (CRYPTO)*, volume 1294 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[2] D. A. Bryan, B. B. Lowekamp, and C. Jennings. SOSIMPLE: A serverless, standards-based, P2P SIP communication system. In *Proceedings of the International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA)*, 2005.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*

(SOSP), volume 37, 5 of *Operating Systems Review*, pages 298–313. ACM Press, 2003.

[4] Clip2. The gnutella protocol specification v0.4. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`, 2000.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-Area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, volume 35, 5 of *Operating Systems Review*, pages 202–215, New York, 2001. ACM Press.

[6] J. R. Douceur. The sybil attack. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer-Verlag, 2002.

[7] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-based publish/subscribe over P2P networks. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware)*, volume 3231 of *Lecture Notes in Computer Science*, pages 254–273. Springer-Verlag, 2004.

[8] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. In *Decision Support Systems*, 2005.

[9] Y. Kim, D. Mazzochi, and G. Tsudik. Admission control in peer groups. In *Proceedings of 2nd IEEE International Symposium on Network Computing and Applications (NCA)*, pages 131–139. IEEE Computer Society, 2003.

[10] F. Lesueur, L. Mé, and V. V. T. Tong. A Distributed certification system for structured P2P networks. In *Proceedings of the 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, Lecture Notes in Computer Science, Bremen, Germany, 2008. Springer-Verlag.

[11] F. Lesueur, L. Mé, and V. V. T. Tong. A sybilproof distributed identity management for P2P networks. In *Proceedings of the 13th IEEE Symposium on Computers and Communications (ISCC)*, Marrakech, Morocco, 2008. IEEE Computer Society.

[12] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*, Computer Communication Review, pages 149–160. ACM Press, 2001.