WCAN 2007

Securing Ad Hoc Storage through Probabilistic Cooperation Assessment [1]

Nouha Oualha [2] Yves Roudier [3]

Eurécom Institute
Sophia Antipolis, France

Abstract

The trend towards self-organization of systems like peer-to-peer or ad hoc networks generates increasing needs for designing distributed storage schemes that should themselves be self-organized and cooperative. Unfortunately, in such systems, the data stored are exposed to new disruption attacks either because of the selfishness of participating nodes with respect to their resources or even because self-organization that leaves such systems much more subject to maliciousness. This setting, combined with the high churnout of nodes and with the high chances of network partition, particularly in mobile ad hoc systems, makes it quite uneasy to ensure the long-term availability of data stored in such fashion. This paper discusses a verification framework based on the probabilistic assessment of small cryptographic verifications in order to assess storage and to prevent data destruction. A protocol for determining the availability of data stored in an ad hoc fashion is detailed. The design of a reputation system based on this protocol and inciting nodes to cooperate towards storage is then discussed.

Keywords: self-organizing networks, ad hoc storage, storage availability, cooperation, selfishness.

1 Introduction

The development of ad hoc networks able to run in adverse environments like accident scenes or battlefields generates tremendous requirements in terms of architectural self-organization. Routing has for instance been vastly explored in the literature. The applications of ad hoc storage are spurring the development of appropriate schemes in this field. Storage applications, which are most prominent in the P2P world with file sharing providing a short-term storage service, typically allow ad hoc nodes to communicate even when coordination between nodes is hardly feasible: in particular, locally produced content may be distributed and may benefit from the storage space available in the immediate vicinity of a node. In unstable environments like ad hoc networks, distributed storage may contribute to achieving more dependable and secure applications through the provision of backup services

[1] This work was supported by the GET Initiative program on autonomic and spontaneous networks, the PACALAB project, and the ReSIST IST NoE.

[2] Email: oualha@eurecom.fr

[3] Email: roudier@eurecom.fr

[3] and [8], crash-tolerant blackboards inspired by desktop teleporting [1], or even contextual and location-aware services [10].

In comparison with P2P architectures, achieving secure and trusted ad hoc storage presents a particular challenge due to the open, autonomous, and highly dynamic nature of ad hoc networks. In particular, it should be anticipated that nodes will leave data behind them because their connection capabilities may be quite ephemeral or because the mobility model of the application is such that nodes are frequently joining and leaving the network; nevertheless, the applications discussed above necessitate the data stored in some "location" or distributed among a set of holder nodes to be protected from destruction as long as necessary. We argue that any effort to protect the storage system should ensure the following goals:

- **Confidentiality and integrity of data:** most of storage applications deal with personal (or group) data that is stored somewhere in the network at generally untrusted nodes. That's why data must be protected while it is transmitted between intermediary nodes and while it is stored at the destination node. Typically, confidentiality and integrity of stored data is ensured using common cryptographic means such as encryption methods and checksums.
- **Anonymity:** anonymity can refer to the data owner identity, the data holder identity, or the interaction details operated between them. Anonymity avoids attacks where the attacker targets all holders of a given data, thus extinguishing data from the whole system. Systems that seek to provide anonymity often employ infrastructures for providing anonymous connection layers, e.g., onion routing [7].
- **Identification:** within a distributed environment like P2P or mobile ad hoc networks, it is possible for the same physical entity to appear under different identities, particularly in systems with highly transient populations of nodes. This problem may lead to attacks called "Sybil attacks" [5], and may threaten as well mechanisms such as data replication that rely on the existence of independent peers with different identities. Solutions to these attacks are the deployment of a central certification authority or more likely in an ad hoc network, limiting the acceptable operations if the connection of too many ephemeral and untrustworthy identities is observed. This objective may limit anonymity.
- **Access control:** the lack of authentication can be overcome by the distribution of the keys necessary for accessing the stored data to a subset of privileged nodes. Access control lists can also be assigned to data by their original owners through the use of signed certificates.
- **Long-term data survivability:** the durability of storage in some applications like backup is very critical. The system must ensure that the data will be permanently conserved (until its retrieval by the owner node). Techniques such as data replication or erasure codes improve the durability of data conservation, but these techniques must be regularly adjusted to optimize the capacity of the system. Generally, the employed adaptation method is based on frequent checks over the stored data to test whether the very data is still held by the expected holder node. Moreover, cooperation incentive techniques must be used to encourage holder nodes to preserve the data they store as long as they are able to.

- **Data availability:** any storage system must ensure that the stored data is accessible and useable upon demand by an authorized node. Data checks at holder nodes allow the regular verification of this property, and at the same time inspecting the route (connectivity) between the owner node and the holder node.

We concentrate in this paper on the last two objectives above: high availability and long-term durability of data storage in the context of a mobile ad hoc environment. These two objectives are often ignored in P2P file sharing applications which rather follow best effort approaches. This paper suggests that performing periodic cryptographic verifications should enable the probabilistic evaluation of the availability of data stored in the system. Such storage evaluation enables the design of a cooperation enforcement framework; contrary to ad hoc routing cooperation where the result of packet forwarding can be observed immediately, such a system requires crafting a long-term evaluation scheme adapted to the peculiarities of the ad hoc network, which feature frequent connectivity disruptions.

This paper's contribution is threefold: an architecture for verifying ad hoc storage that overcomes the connectivity issues of ad hoc networks is first introduced. Second, a new verification protocol of data possession is described that fits the requirements of the framework. Finally, the use of storage verification for building a reputation system for discriminating well-behaved nodes from malicious ones is discussed.

2 An Architecture Enabling Ad Hoc Storage Verification

This section describes the architecture of the storage framework designed for ad hoc networks , and in particular how verifications can be implemented. A typical multi-hop mobile ad hoc network consists of mobile nodes which communicate with each other through multiple routes. The storage application allows a data owner node to select a given holder node in its vicinity and to send it the data to store once the holder has agreed to store the data. It is likely that the owner node will repeat this operation for the same data at several holders, with the aim of fostering data availability by replication. The application should permit the owner node to periodically verify if the holder node still possesses the data it has pledged to store, the verification consisting in a challenge-response protocol between the owner node and the holder node (see Section 3). Intermediary nodes on the route between the two nodes may as well verify the holder response, thereby making it possible for them to evaluate the behavior of the holder node, and to get a rough idea of the trustworthiness of the holder by themselves even if they do not store data at this node.

In addition to having constrained resources, mobile devices operate in an adverse environment where end-to-end connectivity is highly susceptible to various disruptions, especially so if the owner frequently joins and leaves the network. To cope with such issues, the owner can delegate storage verifications to other mobile nodes it trusts. While any trusted node may perform storage verifications, we suggest the use of dedicated nodes may make sense. The concept of "throwbox" [13] has been suggested to improve the connectivity of an ad hoc network in a DTN fashion.

Throwboxes normally relay data between nodes using a "store-carry-and-forward" paradigm. They have storage capability and they are easy to deploy in the field without access to any infrastructure. More complex versions of such relay nodes based on overlays have even been proposed [12]. We propose that throwboxes belonging to trusted authorities may carry out storage verifications on behalf of the owner node, thus improving the long-term availability of data storage in the region around a throwbox instead of restricting verifications to when the owner node is connected only. Throwboxes would not have to store data, but rather security metadata (who stores which owner data, and proofs of delegation) making it possible to perform periodic data storage verifications. They may also be in charge of replicating data as they would see fit based on the evaluation of security (see Fig.1 for examples). Thus, verification not only makes it possible to assess the cooperation of holders with respect to the storage application, but also to react to data destruction by increasing the number of replicas in the network. Throwboxes may also make it possible to cope with situations where the owner and holders are regularly, but not simultaneously reconnected to the network.

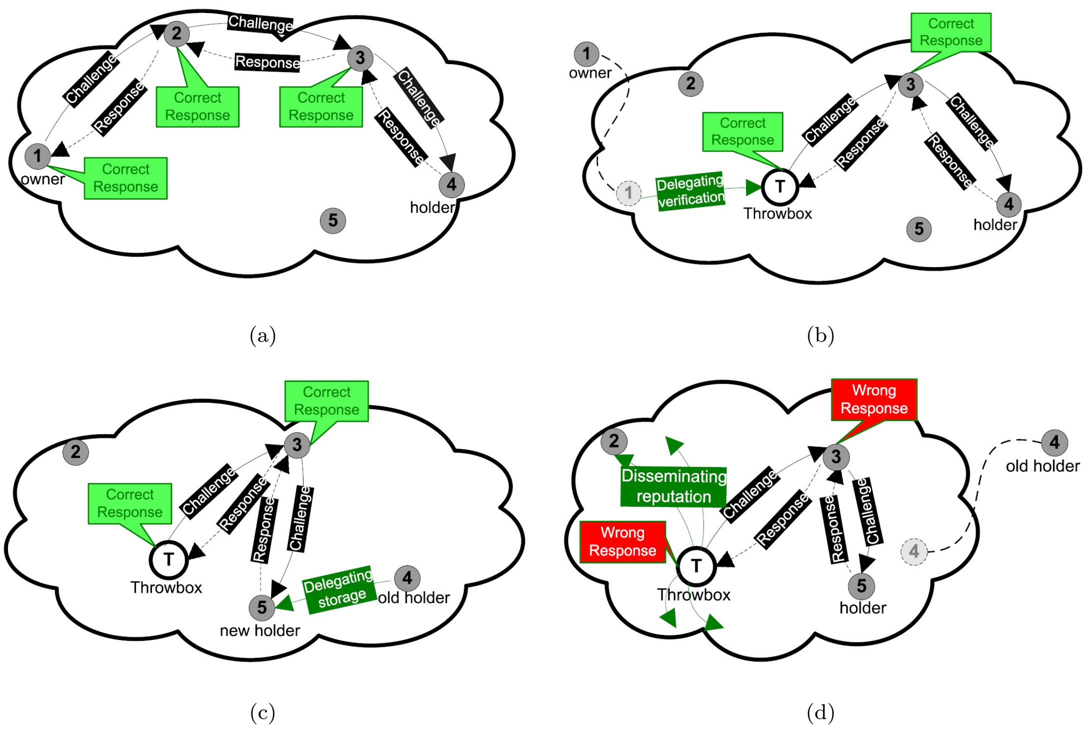

Fig. 1. Scenarios: (a) the owner node n°1 can communicate with the holder node n°4 to perform verifications, (b) the owner node cannot communicate with the holder node, the throwbox performs verification on behalf of the owner node, (c) the holder node delegates storage to another node n°5, and (d) the throwbox detects storage default at the new holder node, and informs neighboring nodes. It would then trigger a new replication as if it were the owner.

3 Verification Protocol

This section first discusses the properties that the protocol must satisfy to verify correctly and efficiently that a node still possesses some data at the time of challenge.

A new protocol that allows a node to probabilistically verify, using a key and based on challenge-response messages, whether a data holder node still possesses the data it agreed to store for the originator node is then introduced.

3.1 Properties

In the following we define the properties of the verification protocol that meets the expectations of the verification framework presented earlier:

- **Soundness:** if data is destroyed by the holder node, the latter cannot prove it is storing data to the verifier except with a negligible probability.
- **Completeness:** if both the verifier and the holder are honest and they correctly follow the proposed protocol, the verifier always accepts the proof of the holder as valid.
- **Eligibility:** a challenge should only be sent by the owner or the nodes that possess a certificate signed by the owner. The eligibility property is required to prevent denial of service (DoS) attacks, e.g., flooding of the holder node with challenges.
- **Verifiability:** any node that knows the challenge and its response can verify the correctness of the response. This property allows improving the way in which trustworthiness is assessed in the system, by taking advantage of the multi-hop routing architecture.
- **Efficiency:** the protocol impact should be as low as possible, especially so if devices have scarce resources.

Protocols for the verification of data possession have been mostly studied within P2P systems, in which data storage and backup have been addressed in a distributed and self-organized fashion. Two families of protocols should be distinguished: the first one (e.g., [9] and [2]) relies on the verification of some data sent by the holder against the original data kept by its original owner, while the second one (e.g., [4] and [6]) is based on the verification of a proof generated on demand (when receiving a challenge) from the data. The former category has been most used for backup applications, since data are preserved at the originator until a crash happens, while the latter one better addresses distributed storage applications in general. A list of existing verification protocols is presented in Table.1. The protocol proposed in this paper pertains to the second category yet it does not require the verifier to perform time-consuming computations to answer challenges because of the small size of the data chunks verified, contrary to [4] or [6]. The protocol also does not require the verifier to keep data nor pre-computed challenges.

3.2 Probabilistic verification Protocol

We assume that a node stores its data onto other nodes it encounters. In order to ensure that nodes preserve these data in their storage space, they are periodically challenged to prove they still have them at hand. The challenge periodicity should be fixed based on network and nodes specific performances, in particular in order to avoid attacks such as flooding by excessive challenging (see discussion below in

	Lillibridge et Al. [9]	Caronni and Waldvogel [2]	Deswarte et Al. [4]: Hash solution	Deswarte et Al. [4]: RSA solution	Filho and Barreto [6]
Soundness	Yes	Yes	Yes (limited by the number of challenges)	Yes	Yes
Completeness	Yes	Yes	Yes (limited by the number of challenges)	Yes	Yes
Eligibility	Yes	Yes	Yes	Yes	Yes
Verifiability	No	No	No	No	No
Efficiency	Data stored at verifier Simple comparison	Data stored at verifier MAC computation	Pre-computed challenges Hash computation	A hash of data is stored at verifier Data as an RSA exponent	A hash of data is stored at verifier Data as an RSA exponent

Table 1
Storage verification protocols

Section 3.3). The players in the system envisioned are denoted as follows:

- O denotes the data originator or owner.
- H denotes the data holder which stores O's data.
- V denotes the verifier: it can be O, or a delegate of O. We will concentrate on O as a verifier.
- P denotes the data prover, which answers challenges. The verification should prove that P is H or a registered delegate. We will address only the case $P = H$.

The verification protocol requires asymmetric pairs of cryptographic public/private keys (we name them verification keys) owned by participating nodes. Private keys are kept secret, while public keys are widely distributed. The protocol comprises the following two phases (see Fig.2):

Setup phase: O splits the data, termed M, into n segments, $\{m_i\}_{1 \le i \le n}$. O signs each segment, along with its index, using the verification keys. The result is the set $\{Sign_O(m_i, i, O)\}_{1 \le i \le n}$. ($Sign_U(m)$ means signature of message m by the private key of U). Then, O sends $\{m_i, Sign_O(m_i, i, O)\}_{1 \le i \le n}$ to H for storage.

Verification phase: V randomly chooses a value j in $[1, n]$. It generates a timestamp T, signs it, and then concatenates it to the index j (the timestamp prevents challenge denials of service using message replay). V sends the challenge message to P. P verifies the freshness of T and responds with the corresponding couple $(m_j, Sign_O(m_j, j, O))$ together with a signed T: the timestamp is used to prevent a man-in-the-middle attack replaying an old response and destined to denigrate H by pretending it is not storing the requested m_j. V verifies if the signatures in the returned response are all valid.

Setup phase

(i) $O \rightarrow H : m_i, Sign_O(m_i, i, O)_{1 \leq i \leq n}$

Verification phase

(i) V chooses a random value index j in $[1, n]$

(ii) $V \rightarrow P : O, j, T, Sign_V(O, j, T)$

(iii) V verifies $Sign_V(O, j, T)$

(iv) $P \rightarrow V : m_j, Sign_O(m_j, j, O), T, Sign_P(m_j, Sign_O(m_j, j, O), T)$

(v) V verifies $Sign_P(m_j, Sign_O(m_j, j, O), T)$ and $Sign_O(m_j, j, O)$

Fig. 2. Probabilistic verification protocol, with $V = O$ and $P = H$

In this protocol, P proves that it is keeping a data segment for O, and gives some evidence of its origin and its integrity. The verification process requires computational resources consumed at V, and additional storage space together with some computation at P. The extra storage at P is the price to pay for a verification process without data, or pre-computed information stored at V. Keys do not need to be stored by V if they can be generated based on a passphrase for instance. Such an approach, or the use of a token, would be required for a storage application in O may have completely crashed, thereby losing any secret stored there.

The proposed protocol verifies the completeness property because if the correct answer of P to a challenge suffices V to be convinced that H is keeping all data. Regarding the soundness property, if H destroys the data or a portion of the data, P cannot answer challenges sent by V with high probability (this is analyzed in the following subsection). H answer challenges for a specific data just only from eligible nodes: the data owner O and owner delegates including throwboxes, denoted V. In the protocol, the timestamp concatenated to the challenge message guarantees that the message is sent by O. For a delegate, the challenge message must comprise in addition to the signed timestamp a certificate proving the legitimacy of the verifier as a delegate in the form of $Cert_O(O, PK_V, validity)$, signed by O. Also considering the verifiability property, any node that has knowledge of the respective public keys of P and V, can verify the correctness of the response for a given challenge. Finally, the probabilistic aspect of the protocol allows to trade off some security with performance (small communication overhead). The protocol does not require storage at V and the verification process is much less expensive than verifying the whole data at once.

3.3 Security analysis

In this section, we study the security of the verification protocol: firstly by examining the probabilistic approach suggested by the protocol, and then by looking to some additional prominent attacks and their respective solutions.

3.3.1 Probabilistic assessment

In the proposed protocol, if the result of the verification performed at step (5) is true, then V is probabilistically assured that P still holds data. By each verification, V checks that P holds the segment m_j. Since j is chosen randomly, P has to keep all segments and their signatures $\{m_i, Sign_O(m_i, i, O)\}_{1 \leq i \leq n}$ to answer correctly to all challenges, unless it takes chances to revealed as an attacker at some point in time. This subsection discusses how secure is such a probabilistic verification. We are making the following assumptions:

- V's random selection of indexes is uniform, i.e., the probability to pick any segment is $1/n$.
- Index selections are independent events.
- P removes a data segment with a uniform probability d from its storage. The probability d is the same for all segments and it is referred to as the misbehavior rate of P.
- V performs c challenges where $1 \leq c \leq n$.

The probability that V detects P's misbehavior is given by $p_{detection}$:

$$p_{detection} = 1 - (1 - d)^c$$

For a given probability of detection of misbehavior, it is possible to probabilistically determine the average number of challenges that V should perform to attain this probability. The number of challenges c can be derived as follows:

$$c = \log_{1-d}(1 - p_{detection})$$

The required number of challenges to acquire a given probability of detection is most of the time not equal to 1 (see Fig.3). Therefore, the message challenge of V must consist of not only just one index, but c indexes. The appropriate value for c can be chosen based on the misbehavior rate of P estimated thanks to P's reputation (see Section 4). Also, the given probability of detection can be fine-tuned with respect to the criticality of the stored data: a very critical data (respectively an ordinary data) implies a high value (respectively a low value) for $p_{detection}$.

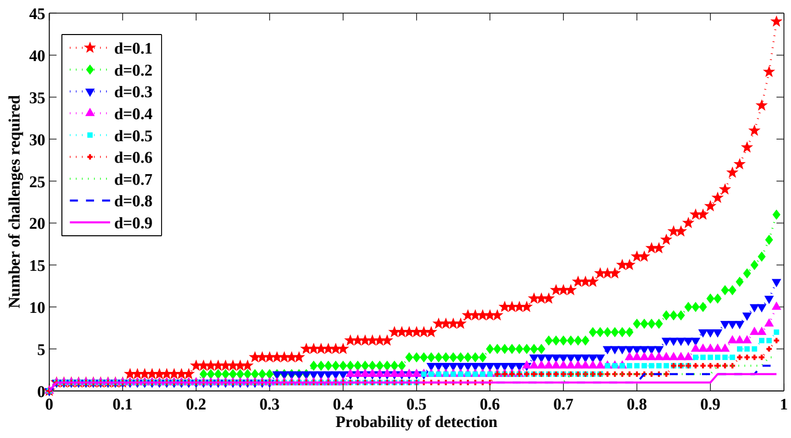

Fig. 3. Number of challenges required to achieve a probability of detection of P's misbehavior

3.3.2 Countering additional attacks

The described verification protocol permits to detect selfish holders that destroy the data they have promised to store. However, the protocol alone is not able to defend against other forms of misbehavior, such as denial of service attacks, proxying attacks, or colluding replica holders. A flooding attack can be launched by the verifier, by sending a large number of challenge messages to a victim data holder in order to slow it until it is unusable or crashes. Although this type of attack is unlikely to happen since the verifier performs computational operations for every challenge, it is possible to limit the number of challenges by imposing a quota on the frequency of challenges. This solution is proposed in [9]. Moreover, it is possible to force the verifier to pay fees for every challenge it requests, for instance using a micropayment scheme. An alternative approach is to reciprocate in storing data, thereby performing symmetric verifications between the two nodes, like in [2].

A holder can pretend to be storing data while in fact proxying in front of another data holder. Then, the attacker simply passes data back and forth between the originator and the holder, making the data originator believe that it is the data holder, and the data holder that it is the originator of the data. This problem can be addressed by having the index j for each challenge randomly chosen by both parties as suggested in the random-read protocol presented in [9] in which both parties randomly choose the offset of the block to be checked.

When using replication mechanisms to support the availability of data, replica holders may collude so that only one of them stores data, thereby defeating the purpose of replication to their sole profit. One way to counter this attack is to produce personalized replicas for each holder, as described in [2], by using an encryption key (used to encrypt the data) derived from the identity of the holder. Responses to a challenge are constructed such that $Sign_O(m_i, i, O, ID_P)$ for $1 \leq i \leq n$. When verification of data is managed solely by a throwbox, this throwbox can construct new personalized replicas based on its own signature (these replicas will comprise as well the previous signature of the owner node). Similar mechanisms should be crafted, for instance based on an identity strongly linked with the reputation and that may be a SPKI like cryptographic key provided by an offline trusted third party (TTP).

4 Cooperation Incentives

Participating nodes have distinct strategies, and may in particular try to save their energy or storage space. In particular, a node may try to profit from the storage system without contributing to it in a "free-riding" fashion. To counter such behavior, we follow the trend set by P2P file sharing applications, and suggest using cooperation incentives. Nodes in the system compute the reputation of other nodes based on the results of the verification process performed over the data stored by the holder node. This reputation is then used to decide on the storage policy towards assessed nodes. Reputation computation may be based on direct interaction or indirect observation of storage verifications by intermediary nodes that do not challenge the data holder. However in the latter case, nodes may collude in order to increase their reputation: a request may be sent over a multi-hop link for which the pretending holder has a predefined answer. Intermediate nodes should therefore give less weight to the interpretation of such results than of those resulting from their own interactions. For instance, they can assign a lower weight to nodes always recommended by the same owner node, or by nodes with a low reputation. Reputation should also slowly decrease if no verification has been witnessed by the computing node: this prevents nodes with good reputation from stopping their contribution to storage operations while enjoying the benefits of cooperative storage. Considering the assumption used above that a selfish holder node destroys data

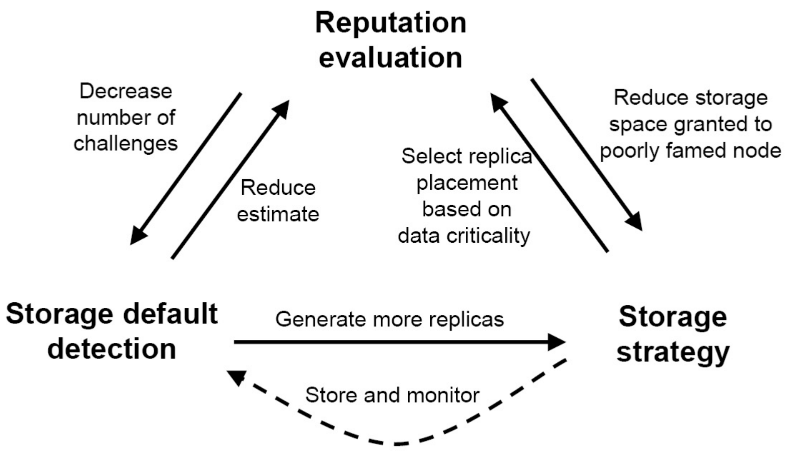
Reputation evaluation
Decrease number of challenges
Reduce estimate
Select replica placement based on data criticality
Reduce storage space granted to poorly famed node
Storage default detection
Generate more replicas
Storage strategy
Store and monitor

Fig. 4. Reacting to the detection of a storage default

with a probability d, the destruction of data will be detected on average after $1/d$ verification operations. Therefore after n verification operations that have produced correct responses, the verifier is probabilistically sure that the rate of destruction by the data holder is lower than $1/(n+1)$. For every verification operation performed against the holder, the verifier therefore can reevaluate the probability of destruction $d \approx 1/(n+1)$ which corresponds to reevaluating the reputation of the holder and the required number c of indexes per challenge round. Since they are considered as less reliable, poorly famed nodes do not require as many challenges to be deemed dependable according to the standards of their reputation. In contrast, well famed nodes should be further challenged up to a chosen threshold that determines the performance penalty the owner is ready to pay in terms of verification overhead. Whenever the verifier detects that the holder node has destroyed some data, it reduces the reputation of the latter to zero. This severe policy against selfish nodes may be unfair for holder nodes that have lost data because of a crash or faults. Therefore, we suggest the use of a grace period (e.g., one day) for the holder node to prove its conformance.

The reputation system also guides nodes in choosing the best strategy for storing their data. For instance, one's critical data should rather be stored at a good reputation node if one is available around. Verifying the data stored at some holder thus enables estimating its reputation which in turn helps choose the right holders. Fig.5 depicts the feedback loop taking place after data destruction is detected, and illustrates the correlation between storage verification, reputation estimation, and storage strategy. The use of the verification protocol as a building block for cooperation incentives has been validated in [11] using game theory (even though this work assumes a remuneration based instead of reputation based incentive scheme). With

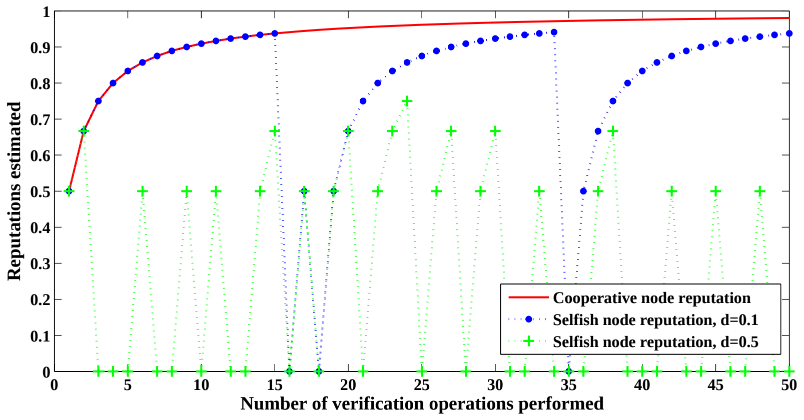

Fig. 5. Examples of reputation computation for different types of nodes submitted to periodic storage challenges

time, the reputation of a cooperative node increases; the reputation of selfish node fluctuates between 0 and $(1 - d)$. Every node in the system must have a history of some nodes whenever the reputation of these nodes nullifies, in order to detect these fluctuations and conclude that these nodes are selfish and must be blacklisted. The measured reputation is bounded between 0 and 1. We denote the reputation of the holder P at time t, $R_P(t)$. After a positive verification operation, the reputation is re-estimated as it follows:

$$[R_P(t)]_{new} = 1/(2 - [R_P(t)]_{old})$$

Fig.5 illustrates the reputations variations for three types of nodes: a cooperative node, a selfish node with probability of destruction $d = 0.1$, and a selfish node with $d = 0.5$. The cooperative node sees its reputation increasing asymptotically approximating a reputation equal to 1. The selfish node with $d = 0.1$ has a reputation fluctuating between 0 and 0.9.These fluctuations becomes more important for the selfish node with $d = 0.5$.

5 conclusion

We described how storage may be enabled through cooperation in ad hoc networks. Securing such an application requires at least enforcing cooperation, hence assessing it first. This paper details a cryptographic protocol that may be used to prove that some data are still in the possession of the node supposed to store it. This probabilistic challenge-response protocol aims at detecting misbehaving nodes attacking the cooperative storage scheme in a less expensive manner than cryptographic approaches in the literature. The protocol also does not require verifiers to store the original data, which are entirely kept by the prover, so that the verification process may be delegated to trusted nodes. We finally discuss how to build reputation incentives based on the results of storage verification for punishing ill behaved nodes and for increasing storage dependability and durability. We are currently developing alternative verification techniques and plan to investigate the performance requirements of our scheme in particular with respect to cryptographic verification primitives.

References

[1] F. Bennett, T. Richardson, and A. Harter. Teleporting - making applications mobile. In IEEE Computer Society Press, editor, *IEEE Workshop on Mobile Computing Systems and Applications*, pages 82–84, Santa Cruz, California, December 1994.

[2] Mark Caronni and Marcel Waldvogel. Establishing Trust in Distributed Storage Providers. In *Third IEEE P2P Conference, Linkoping, 20003*, 2003.

[3] Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Pastiche: making backup cheap and easy. *SIGOPS Oper. Syst. Rev.*, 36(SI):285–298, 2002.

[4] Yves Deswarte and Jean-Jacques Quisquater. Remote Integrity Checking. In Sushil Jajodia; Leon Strous, editor, *Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, pages 1–11, www.wkap.nl, 2004. Kluwer Academic Publishers.

[5] J. Douceur. The Sybil attack. In *The 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, MIT Faculty Club, Cambridge, MA, 2002.

[6] D. G. Filho and P. S. L. M. Barreto. Demonstrating data possession and uncheatable data transfer. In *IACR Cryptology ePrint Archive*, 2006.

[7] D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. In *Comm. ACM*, number 42 in 2, pages 39–41, 1999.

[8] Marc Olivier Killijian, Michel Banâtre, Yves Roudier, David Powell, and Paul Couderc. Collaborative backup for dependable mobile applications. In *MPAC'04, 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing, October 18th - 22nd, 2004, Toronto, Canada / Proceedings published in ACM International Conference Proceeding Series, ACM 2004, ISBN:1-58113-951-9*, Oct 2004.

[9] Mark Lillibridge, Sameh Elnikety, Andrew Birrel, Mike Burrows, and Michael Isard. A Cooperative Internet Backup Scheme. In *Usenix Annual Technical Conference (General Track), pp. 29-41*, Jun 2003.

[10] N. Marmasse and C. Schmandt. Location-aware information delivery with commotion. In Springer Verlag, editor, *Second International Symposium on Handheld and Ubiquitous Computing, HUC 2000*, pages 157–171, Bristol, UK, September 2000.

[11] N. Oualha and Y. Roudier. A game theoretic model of a protocol for data possession verification. In *The Third IEEE International Workshop on Trust, Security, and Privacy for Ubiquitous Computing*, Helsinki, Finland, June 2007.

[12] G. Yang, L. Chen, T. Sun, B. Zhou, and M. Gerla. Ad-hoc Storage Overlay System (ASOS): A delay-tolerant approach in MANETs. In *The Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'06)*, Vancouver, Canada, 2006.

[13] W. Zhao, Y. Chen, M. Ammar, M. Corner, B.N. Levine, and E. Zegura. Capacity enhancement using throwboxes in DTNs. In *IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS)*, Vancouver, Canada, October 2006.