Institut Eurécom
Corporate Communications Department
2229, route des Crêtes
B.P. 193
06904 Sophia Antipolis
FRANCE

Research Report RR-07-188

Probabilistically Secure Cooperative Distributed Storage

8 February 2007

Nouha Oualha and Yves Roudier[1]

Tel: (+33) 4 93 00 81 95
Fax: (+33) 4 93 00 82 00
Email: {oualha, roudier}@eurecom.fr

[1] Eurecom's research is partially supported by its industrial partners: BMW Group Research & Technology – BMW Group Company, Bouygues Telecom, Cisco Systems, France Telecom, Hitachi, SFR , Sharp, STMicroelectronics, Swisscom, Thales

Probabilistically Secure Cooperative Distributed Storage

Nouha Oualha and Yves Roudier

Abstract

The trend towards self-organization of systems like peer-to-peer or mobile ad hoc networks generates increasing needs for distributed data management, in particular data backup and distributed storage functions that will themselves be self-organized and cooperative. Unfortunately, in such systems, the data stored are exposed to the selfishness (e.g. data discarded because of lack of resources) or to the maliciousness (e.g. data voluntarily discarded for disruption of data storage functionality) of participating nodes. This setting, combined with the high churnout of nodes in such systems, makes it quite uneasy to determine the actual availability of some data stored cooperatively. We propose in this report the combination of probabilistic and cryptographic verifications to evaluate this availability and possibly react as it becomes less probable. After reviewing existent approaches to distributed data storage protection, we present a protocol for determining the availability of data stored in a distributed fashion with various degrees of criticality.

Probabilistically Secure Cooperative Distributed Storage

Nouha Oualha and Yves Roudier

1. Introduction

The trend towards increased self-organization, as illustrated by the interest for peer-to-peer (P2P) systems and mobile ad hoc networks, spurs a tremendous growth in the amount of data available and generated in the field. While the growing interest in peer-to-peer and ad hoc networks and applications makes it necessary to implement data backup and distributed storage in a cooperative and distributed form, self-organization in such systems exposes the data to new threats. Such misbehavior may be illustrated by selfishness whereby nodes may discard some data they promised to store or to back up for other nodes in order to optimize their resource usage, or by maliciousness whereby nodes aim at destroying data backups. Because of the dynamics and the scattered nature of nodes, especially in mobile ad-hoc systems, checking that a data has been backed up or stored somewhere is quite more complex than, and more importantly not as immediate as checking that a route has been established with another node in multi-hop MANETs for instance. All these problems contribute to the difficulty of properly determining the actual availability of stored data. Countermeasures that take into account the fact that users have full authority on their devices should be crafted to prevent them from cheating the system in order to maximize the benefit they can obtain out of the system cooperative function.

This report suggests probabilistically performing cryptographic verifications to allow the evaluation of the availability of data stored in a distributed fashion in the system, and to ensure that availability according to various levels of criticality. The approach suggested relies on a challenge-response based protocol that makes it possible for the data originator to determine whether data holders are still keeping the data they are assigned to preserve. The verification protocol proposed in this report yields a good tradeoff between the additional resource consumption of nodes due to security verifications and the confidence in the verification process.

2. Motivation

In P2P or mobile ad hoc networks, a data originator may store more or less critical data he just produced and without access to a dedicated, secure, and centralized storage infrastructure. Generally, the level of data criticality is associated with the rate of replication that is considered as a means for ensuring the availability of stored data. However, protecting data through replication techniques can be hampered by the presence of selfish and malicious replica holders. The following describes an example of a backup application, then the benefits that the verification of data possession brings to this application.

2.1. An opportunistic backup application

MoSAIC [1] addresses data backup in the particular context of one-hop mobile ad hoc networks, which may be supported by opportunistic and transient pairing and communication with Bluetooth or 802.11. The backup application addressed in this project aims at enabling a mobile node to back up its data by exploiting the storage space of nearby accessible nodes through a spontaneous interaction. It is also assumed that nodes may connect to an infrastructure from time to time, even though permanent connection would be too expensive for backing up non critical data for instance. Data could be transferred either immediately back to their owner if the mobile nodes are still in touch, or possibly through a trusted third party, for instance residing with the fixed infrastructure to which devices are regularly reconnected.

Backups rely on the mutual cooperation of these nodes with no prior trust relationships. To resist misbehavior, a trust management mechanism is being elaborated for MoSAIC using both a long-term reputation about the fulfillment of the cooperative task by one node, and a short-term reputation making it possible to evaluate the cooperation of a node.

The decision whether to cooperate with a node is based on its long-term reputation which is self-carried. This self-carried reputation is updated every time the node is connected to the infrastructure, where an authority (i.e., a trusted third party (TTP)) assigns new reputation ratings to every node after checking that it delivered the data it had promised to store. Evidences of some storage operation may be established through cryptographic proofs of every interaction performed or through the collection of equivalent traces of the nodes' history by an embedded tamper-resistant hardware.

2.2. Computing the short-term reputation

It is especially important to cope with the sporadic access of nodes to the infrastructure. Apart from the long-term reputation of a node, trust towards this node thus dynamically evolves based on the discovery of misbehaving nodes using direct verification. Backup or distributed storage systems perform such verification operations based on challenges sent on a regular basis in order to check whether data holders are still storing data. We claim that this assessment, which consists for every node in computing locally a short-term reputation based on its own interactions with others is an essential part of trust management in a MoSAIC-like application.

The verification of data possession operated on data holders does not straightforwardly protect the data stored at these holders; still, it represents a necessary primitive for reacting to their destruction. The detection of the misbehavior of a replica holder will first reduce the short-term reputation estimate of the data owner. The data originator may then decide to replicate its data again, and to store these new replicas at other nodes. This originator's response to the removal of data will depend on the criticality level of the data suppressed. The TTP will sanction the misbehaving holder node when it reconnects, which will finally be exposed, although in a delayed and more static manner to all other nodes by reducing the data holder self-carried long-term reputation. An example of short-term and long-term reputation variation is showed in Figure 1 where the reputation corresponds to a node that at a certain time stores the data of another node, an at a later time misbehaves and gets detected. This report presents an assessment method for short-term reputation and discusses its performances.

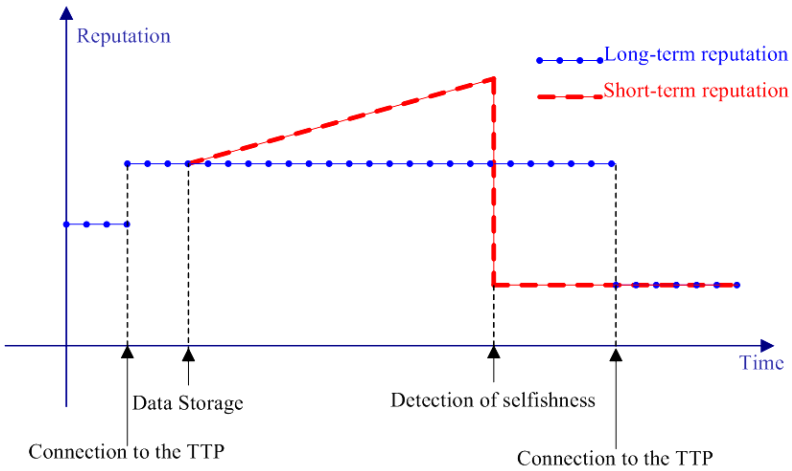

Figure 1. An example of short-term and long-term reputation variation

3. Related work

Protocols for the verification of data possession have been mostly studied within P2P systems, which have addressed the problem of storage and backup in a distributed and self-organized fashion. Two categories of protocols can be distinguished: the first one relies on the verification of some data sent by the holder against the original data kept by its original owner, while the second one 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, while the latter one better addresses distributed storage applications in general. The protocol proposed in this report pertains to the latter category.

3.1. Verification against original data

The first family of verification approaches regularly challenges the data holder to send back the original data: the verification of the holder's response is then compared with the original data.

In the cooperative Internet backup scheme described in [2], each peer periodically challenges its partners by requesting a block out of the stored data. The response is checked by comparing it with the valid block stored at the verifier's disk space. This work assumes that all blocks stored at an unreliable node are probably lost if one block cannot be recovered. It also assumes a P2P connectivity model, that is, most nodes are reachable. A mobile ad-hoc backup application makes essentially inverse assumptions.

In [3], the prover has to send the MAC of data as the response to the challenge message. The data originator sends a fresh nonce (a unique and randomly chosen value) as the key for the message authentication code: this is to prevent the holder node from storing only the result of the hashing of the data.

3.2. Verification using an on-demand proof

The second type of verification protocol answers a challenge by cryptographically combining a proof of the existence of the data at the holder together with some assurance regarding the freshness of this proof.

[4] offers two solutions: the first one requires pre-computed results of challenges to be stored at the verifier, where a challenge corresponds to the hashing of the data concatenated with a random number. The size of the pre-computed information required is smaller than the data size. Compared with [2] and [3], the protocol requires less storage at the verifier, yet it allows only a fixed number of challenges to be performed. The second solution uses an RSA-based proof. This solution requires little storage at the verifier side yet makes it possible to generate an unlimited number of challenges.

A similar RSA-based solution is described by Filho and Barreto in [5] that makes use of a key-based homomorphic hash function H. In each challenge of this solution, a nonce is generated by the verifier which the prover combines with the data using H to prove the freshness of the answer. The prover's response will be compared by the verifier with a value computed over *H(data)* only, since the secret key of the verifier allows the following operation (d for data, and r for nonce):

$$H(d+r)=H(d)\times H(r) \quad (1)$$

Both solutions in [4] and [5] however use exponential operations where the size of the exponent is approximately equal to the data size, which entails a poor performance. So as to boost the performance, the authors of [5] suggest the use of elliptic curves.

4. Verification protocol

We introduce a new protocol that allows a node to probabilistically verify, using a key and based on challenge-response messages, whether a data holder still possesses the data he agreed to store for the originator. This protocol does not require the verifier to keep data or pre-computed challenges, nor the prover to perform time-consuming computations to answer challenges.

4.1. Definitions

Given a mobile ad hoc context, 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[2] should be fixed based on network and nodes specific performances, in particular in order to avoid attacks such as flooding by excessive challenging.

The players in the system envisioned are denoted as follows:

- *O* denotes the data originator or owner. *O* has full rights over stored data. This entity may also be called the verifier *V* with respect to its role in the second phase of the protocol
- *P* denotes the data prover. *P* stores a replica of *O*'s data, and answers to challenge requests.

4.2. Protocol principle

The protocol requires two keys: the first one is used to encrypt data (encryption key), and the second one to check data possession (verification key).

The protocol manipulates a key-based function, denoted f_{Ko}, where K_O is the verification key and is only known by *O*, i.e., only *O* can compute for a given *x*, $f_{Ko}(x)$. For instance, f_{Ko} may be a symmetric encryption function like the Data Encryption Standard (DES) [6], or a keyed one-way hash function such as HMAC [7], or a one-way hash function with the message being concatenated with the key.

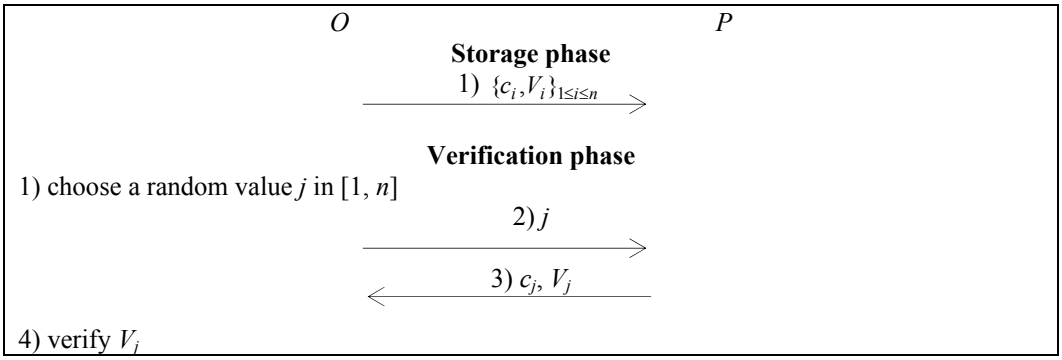

Figure 2. Probabilistic verification protocol

The protocol comprises the following two phases (see Figure 2):

Storage phase. *O* splits the data, termed *M*, into *n* segments, $\{m_i\}_{1 \le i \le n}$. *O* encrypts each segment with the encryption key, the result being the set of encrypted segments $\{c_i\}_{1 \le i \le n}$, and then computes for each

[2] This falls outside the scope of the report

encrypted segment (along with its index) its image with f_{Ko}. The result is the set $\{V_i = f_{K_O}(c_i,i)\}_{1\le i\le n}$. Then, O sends $\{c_i,V_i\}_{1\le i\le n}$ to P for storage.

Verification phase. O randomly chooses a value j in [1,n] and sends it to P. P responds with the corresponding couple (c_j, V_j). O verifies if this couple is a valid one, using its key K_O.

In this protocol, P proves that it is keeping a data segment for O, and provides an evidence that attests of its origin. The verification process requires computational resources consumed at the verifier, and additional storage space together with some computation at the prover. The extra storage at the prover is the price to pay for a verification process without data, or pre-computed information stored at the verifier. Keys do not need to be stored by the verifier 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 which the originator node may have completely crashed, thereby losing any secret stored there.

4.3. Probabilistic evaluation of security

In the protocol described in Figure 2, if the result of the verification performed at step (4) is true, then O is "probabilistically" assured that P still holds data. In reality, O only verifies that P holds the segment c_j. Since j is chosen randomly[3], P has to keep all segments and their images $\{c_i,V_i\}_{1\le i\le n}$ to answer correctly to challenges.

This section investigates how the probabilistic nature of the protocol makes it possible to enforce some security. We are making the following assumptions:

- O's random selection of indexes is uniform, i.e., for n segments, the probability to pick any segment is $1/n$
- Index selections are independent events
- P removes a portion k/n of messages from its storage; this portion is referred to as the misbehavior rate of P.
- O performs on average c challenges; $1\le c \le n$ (n is the number of segments)

The probability that P answers correctly to O's challenges all the time is described as:

$$p = (1 - k/n)^c \quad (2)$$

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

$$p_{detection} = 1 - p \quad (3)$$

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

$$c = \lceil \log_{1-k/n}(1 - p_{detection}) \rceil \quad (4)$$

The required number of challenges to acquire a given probability of detection is most of the time not equal to 1. O should therefore challenge P multiple times.

Figure 3 shows how the number of challenges c increases with the probability of detection. An appropriate value for c can be chosen based on the misbehavior rate of P estimated thanks to the node's long-term reputation together with its short-term reputation as computed from recent interactions. If P has bad reputation, fewer challenges are needed to likely detect P's misbehavior. The opposite will be observed if P has good reputation; however, O will likely store more critical data at P and challenge it more frequently thus compensating the higher number of challenges required to detect a misbehaving node (for a misbehavior rate of 0.1, the required number of challenges exceeds 50 to achieve a high probability of detection).

[3] The selection of an index j, for a challenge, has no impact on its probability to be picked another time for another challenge.

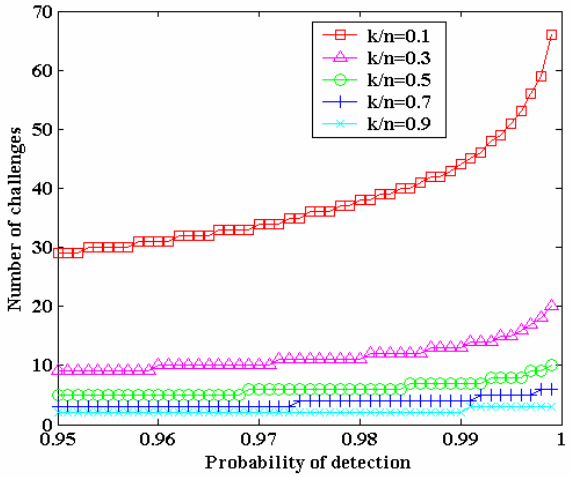

Figure 3. Number of challenges required to achieve a probability of detection of *P*'s misbehavior

In the proposed protocol, false negatives (i.e., verification is positive when *P* removes a given number of segments) are possible, yet their occurrence can be reduced by increasing the number of challenges *c*. False positives (i.e., verification is negative when *P* has got all segments) may occur as well, and are associated with communication losses. The latter issue can be thwarted by usual measures like the retransmission of packets after a timeout.

The data owner *O* needs to send *c* challenges regularly. For the first time, the number of challenges required *c* is computed based on the long-term reputation of the data holder *H*. If *H* answers correctly to the *c* challenges, *H*'s short-term reputation increases. The next time, *O* sends a higher number *c* of challenges since *H*'s reputation has increased, and so on. So, with time, the number of challenges required for every challenge regular period increases, and so do the cost of verification and the pressure (a lot of challenges) on the holder. Thus, the probability of detection of selfishness of holders increases with time.

While the use of a probabilistic approach might be seen as a weaker scheme, it should be noted for instance that multimedia data, like digital pictures or videos may support more degradation for some segments such as image details, than for segments with high-level description: these data, which promise to be one major area where in-the-field storage application will be required, may therefore tolerate less stringent protection mechanisms in exchange for more performance, as it will demonstrated in the following section.

5. Performance evaluation

We analyze in this section the statistical performance of the protocol in terms of communication, computation, and memory usage in comparison with a deterministic approach. We define the deterministic approach for verification as one challenge that carries on the verification of possession of the whole data set and that therefore requires retrieving the original data or their encrypted form (see Figure 4).

The following performance analysis shows that the probabilistic approach of the proposed protocol allows data possession verification to be less expensive for devices with limited resources. This approach indeed permits to trade some security, which can be measured probabilistically, in exchange of better performance.

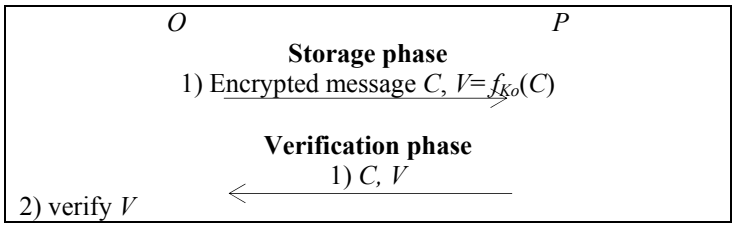

Figure 4. Deterministic verification approach

5.1. Communication overhead

The gain in bandwidth usage compared with the deterministic approach is given by the following expression:

$$\text{Communication\_ratio} = \frac{\text{data\_size} + \text{hash\_size}}{c \times (\text{segment\_size} + \text{hash\_size})} \quad (5)$$

Given expression (5), the increase of the data size implies an increase of the bandwidth needed in the deterministic case, while the decrease of the segment size (*n* increases) leads to a decrease of the used bandwidth in the probabilistic case. Meanwhile, the number of challenges *c* does not vary, because it is function of the probability of detection of misbehavior (that is fixed by *O*) and of the misbehavior rate *k/n* (see equations (2) & (3)). Figure 5 and Figure 6 show that the larger the data are, and the more segmented, the smaller the bandwidth usage for verification relatively gets.

Figure 7 shows that the bandwidth consumption of the protocol increases when the misbehavior rate of data holders rises. The protocol is much more efficient in terms of bandwidth consumption when the data holder has a worse reputation. This result is less significant if there is a higher restriction on the probability of detection.

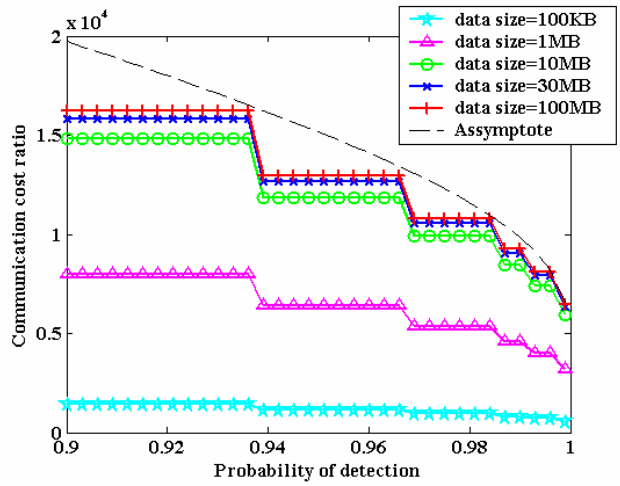

Figure 5. Communication cost ratio with varying data size. $n=2^{16}$, $k/n=0.5$, *hash_size*=128bits.

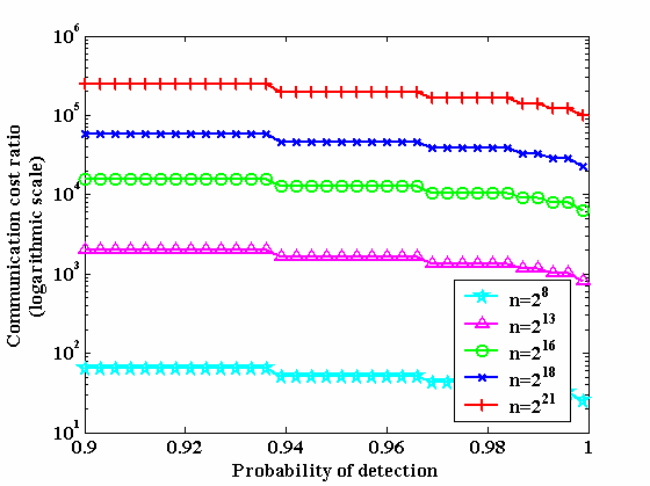

Figure 6. Communication cost ratio with varying number of segments. *Data_size* = 30MB, *k/n*=0.5, *hash_size*=128bits.

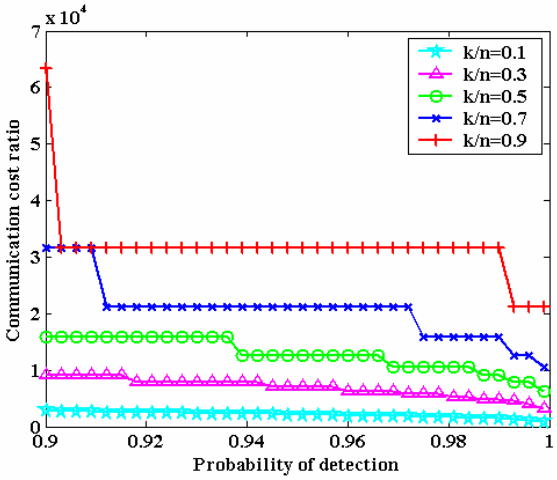

Figure 7. Communication cost ratio over probability of detection of misbehavior. *Data_size*=30MB, *hash_size*=128bits, $n=2^{16}$.

5.2. Computation overhead

The reduction of the verification time required is expressed by the number of computation operations the actual probabilistic protocol can perform compared with a deterministic approach:

$$\text{Computation\_ratio} = \frac{\text{Time\_to\_hash\_data}}{c \times \text{Time\_to\_hash\_segment}} \quad (6)$$

Expression (6) and Figure 8 illustrate the same conclusions regarding the computation overhead as for the consumed bandwidth. The protocol reduces the computation overhead by increasing the size of the data stored and the number of segments. Regarding the concern of energy costs for computation, [8] demonstrates that energy costs for hashing (or symmetric cryptography) varies with the size of data to hash, even though by a small amount: hashing a segment of data is less expensive than hashing the whole data.

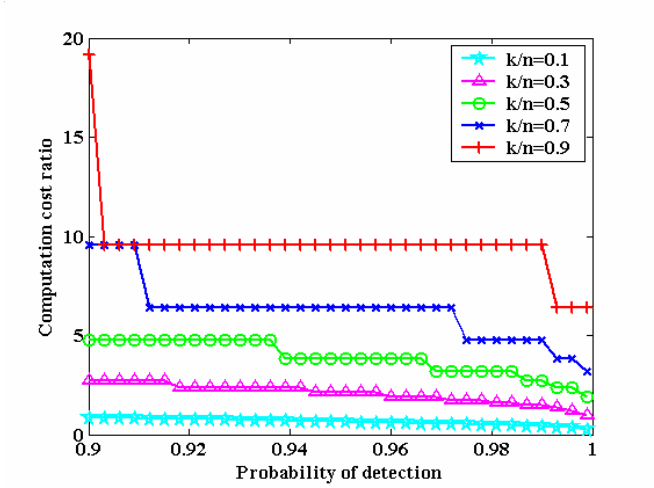

Figure 8. Computation cost ratio over probability of detection of misbehavior. *Data_size*=65536bits, *segment_size=128*bits, hash function: hmac-md5 [7]

5.3. Memory usage

The probabilistic protocol requires that P store a hash value for each segment ($\{V_i\}_{1\leq i\leq n}$). So, in comparison with a deterministic protocol, the storage overhead is equal to (n - 1) × *hash_size*. Depending on the size of a hash and the size of a segment, this storage overhead may represent the same size as the total data stored, if *hash_size* = *segment_size*. Generally, this does not present a problem since distributed storage in P2P or ad hoc networks generally requires the replication of data, which compensates verification storage needs. The protocol requires that O remember a secret key, i.e., the verifier needs the secret key K_O to verify if the data holder is still storing data.

6. Security analysis

In section 4, we described a protocol that guarantees data confidentiality and integrity, and that performs a verification of data possession immediately via challenges sent by the data originator (who owns the secret key). These challenges are a mean to defend against the selfishness of data holders, a form of misbehavior illustrated by the so-called free-riding attack, in which the attacker (called "free-rider") benefits from the system without contributing its fair share. However, the protocol is not able to defend against other forms of misbehavior, like denial of service attacks, man-in-the-middle attacks, or colluding replica holders.

6.1. Flooding attack

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 [2]. Moreover, it is possible to force the verifier to pay fees for every challenge it requests. [3] proposes to make the verifier pay for verification with CPU cycles, called "hash-cash". An alternative approach is to perform a reciprocal storage of data and thereby symmetric verifications between the two nodes, like in [3].

6.2. Man-in-the-middle attack

A man-in-the-middle attacker can appear to be storing data while in fact proxying in front of another data holder. Then, he simply passes data back and forth between the originator and the holder, making the data originator believe that he is the data holder, and the data holder that he is the originator of the data. This problem can be solved by having the index j for each challenge randomly chosen by both parties. This solution is inspired from the random-read protocol presented in [2] where both parties randomly choose the offset of the block to be checked. O and P choose two random values r_1 and r_2 in $[1,n]$. The index of a data segment is computed as: $j = r_1 + r_2 \bmod n$.

The man-in-the-middle attacker will have to perform on average $n/2$ challenges on the original data holder to correctly answer the original data owner challenge request. Using challenge quotas, this protocol will make the attacker quickly exhaust its quota leaving it unable to answer all challenges. Moreover, data holder is still unable to guess the value j in each challenge, since data owner contributes with randomness.

6.3. Colluding replica holders

Replication mechanisms support the availability of data, and are a common technique in current P2P networks. However, 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 [3], 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 $V_i = f_{K_O}(c_i, i, ID_P)$ for $1 \leq i \leq n$. Similar mechanisms should be crafted, for instance based on an identity strongly linked with the reputation and that may be a cryptographic key like in SPKI [9] provided by an offline TTP.

7. Verification protocol delegation

Instead of being performed by O, Self-organization suggests that the verification process might be delegated to another entity than the data originator (Figure 9). There is a potential interest in delegation method for verification, mainly in case O may be a pervasive device with not enough computing power to be able to undertake the verification of a challenge. The evaluation of P's behavior allows O together with O's delegate V to compute a short-term reputation for P. Also, delegates may take advantage of data storage performed by other nodes as a feedback to compute a better estimate of the short-term reputation of a node.

A delegate and data owner would most likely have bidirectional trust relationships. In this case, the delegate may possess the same secret key as the owner (or a data specific secret key generated by the owner) for verifying the data possession proof. Alternately, the proof might be generated using asymmetric cryptography, with the delegate knowing the public key of the owner. In this case, the basic protocol proposed in section 4 can be enhanced by replacing the symmetric encryption with an asymmetric one. A node that is aware that O holds data at P, and possesses the public key of O can perform the verification of data possession. Since data may be signed once and verified many times (depending on the number of challenges performed), it is therefore desirable to choose a signature with little verification time. For example, Table 1 shows that it is more appropriate to choose RSA over DSA.

Enabling such a mechanism may however strongly threaten user privacy (especially regarding user location) since nodes may detect who holds whose data, and therefore which nodes met, and since nodes who retrieve this information may also further distribute or correlate it, even after the interaction took place.

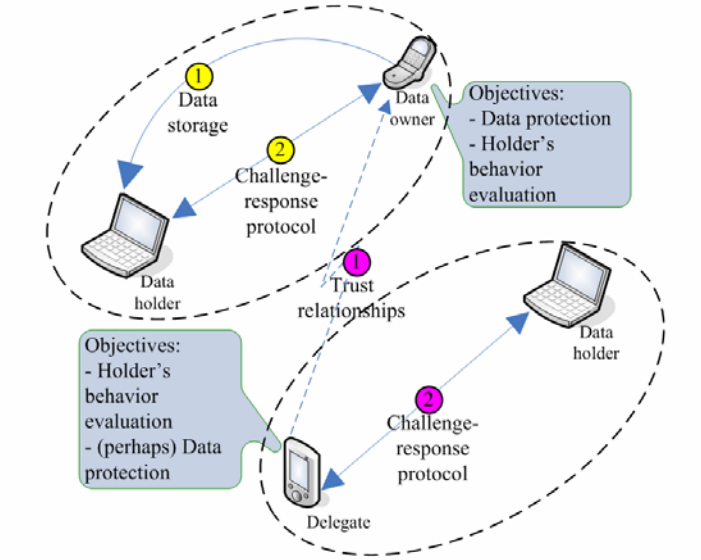

Figure 9: An example of a scenario making use of delegation to perform the data possession challenge-response protocol

Table 1. RSA and DSA signatures compared on an Intel Pentium 4 CPU 2.40GHz (done with OpenSSL [10])

	Key size	Sign	Verify
RSA	512 bits	0.0022s	0.0002s
	1024 bits	0.0130s	0.0007s
	2048 bits	0.0831s	0.0024s
DSA	512 bits	0.0021s	0.0025s
	1024 bits	0.0069s	0.0083s
	2048 bits	0.0235s	0.0294s

8. Conclusion

We presented a new protocol based on challenge-response messages aimed at detecting misbehaving nodes attacking a cooperative and distributed data storage application. This protocol does not require the verifier to store the original data, which are entirely kept by the prover. It relies on the probabilistic checking of the continuous possession of the stored data using computations less expensive than cryptographic approaches in the literature.

We also discussed how the delegation of verification might allow for a larger deployment of our trust management mechanism in mobile ad-hoc environments, where connectivity assumptions are quite different from P2P systems.

We plan to further analyze the performance requirements of our scheme in particular with cryptographic verification primitives, as well as to validate our cooperation based trust establishment mechanism, notably using game theory.

9. References

[1] M.O. Killijian, D. Powell, M. Banâtre, P. Couderc, Y. Roudier. Collaborative Backup for Dependable Mobile Applications. In *Proceedings of the 2nd International Workshop on Middleware*

for Pervasive and Ad-Hoc Computing, Middleware 2004, Toronto, Ontario, Canada, October 18th - 22nd, 2004, ACM.

[2] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, "A Cooperative Internet Backup Scheme", *In Proceedings of the 2003 Usenix Annual Technical Conference (General Track), pp. 29-41*, San Antonio, Texas, June 2003.

[3] G. Caronni and M. Waldvogel. Establishing Trust in Distributed Storage Providers. In *Third IEEE P2P Conference*, Linkoping 03, 2003.

[4] Y. Deswarte, J.-J. Quisquater, and A. Saïdane. Remote Integrity Checking. *In Proceedings of Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, 2004.

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

[6] National Bureau of Standards. Data Encryption Standard. *Federal Information Processing Standards Publication No. 46*, , January 15, 1977.

[7] M. Bellare, R. Canetti, and H. Krawczyk. HMAC: Keyed-Hashing for Message Authentication. *RFC 2104*, Internet Engineering Task Force, February 1997.

[8] N. R. Potlapally, S. Ravi, A. Raghunathan and N. K. Jha. A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols. In *IEEE Transactions in Mobile Computing*, vol. 5, no. 2, pp. 128-143, February 2006.

[9] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen. RFC 2693: SPKI Certificate Theory. *http://www.ietf.org/rfc/rfc2693.txt*

[10] OpenSSL Project, http://www.openssl.org/