Secure P2P Data Storage and Maintenance

Nouha Oualha, Melek Önen, and Yves Roudier
EURECOM, Sophia Antipolis
{oualha|onen|roudier}@eurecom.fr

Abstract

P2P data storage requires strong reliability and security assurances. Existing data storage solutions have been designed for centralized as well as distributed settings, yet they do not address the security and cooperation issues raised by self-organization. P2P systems also introduce new needs regarding data availability due to the dynamicity of the infrastructure, and which are unaddressed so far. This paper first discusses the approaches for tackling these problems. A solution is then introduced that relies on self-organizing security mechanisms in conjunction with a data rejuvenation scheme using erasure codes.

1. Introduction

P2P file sharing systems (e.g., KaZaA [7] and BitTorrent [8]) aim at fairly distributing storage services to the whole community of peers. P2P data storage solutions like Wuala [9], AllMyData [10], and UbiStorage [11] have recently appeared with similar scalability objectives. Such approaches unfortunately do not take the selfish nature of peers into account, even though free riding is a well-known fact in unregulated P2P file storage applications. Selfishness is however much more critical for data storage in that selfish peers may destroy the data they are supposed to store on a long term basis.

Techniques for remotely detecting data corruption or destruction have been analyzed in [1], [2], [3], [5], and [6], and dedicated cryptographic primitives have been proposed. These studies however do not consider the need for a self-organizing data restoration mechanism, which is however critical after data corruption has been detected in order to achieve storage reliability and availability on the long term.

The dynamic nature of P2P systems exacerbates these issues in data storage applications because peers can sometimes be offline while their data still need to be preserved within the network. Achieving scalability requirements in that context requires data storage and verification tasks to be delegated, which has not been addressed in existing systems.

This paper is structured as follows: the requirements for a secure P2P data storage and maintenance mechanism are first introduced and its five main phases are described. A threat model is then presented that outlines possible attacks that may compromise such a protocol. A new solution is then introduced for achieving self-organizing remote data integrity checking mechanisms in conjunction with an erasure code based data rejuvenation scheme. The security of this scheme is finally discussed together with data availability and reliability, the latter being analyzed using an analytic model of the maintenance process.

2. Problem statement

We consider in this paper a P2P storage system in which a peer, the data *owner*, can store its personal data at other network peers. These peers, named *holders*, should store the data until the owner requests its retrieval.

2.1. Requirements

In order to achieve both secure and efficient data storage and maintenance in a dynamic and autonomous system like a P2P network, the solution must ensure the following requirements:

Self-organization: Since current and future P2P storage systems rely on an autonomous communication design, peers take the role of both owners and holders. Therefore, the proposed solution should be designed for a self-

organizing environment whereby all operations including data maintenance are performed by simple peers and without the help of any authority. Since no security server or trusted server can be assumed to be present, security solutions such as peer authentication or data integrity using certificate based authentication mechanisms have to be revisited.

Cooperation: The performance of P2P storage strongly relies on the cooperation of peers. Still, peers are assumed to have limited resources, in particular in terms of storage space or bandwidth available. The scarcity of such resources inherently incites peers to behave selfishly if they are assumed to be rational, even though it should be noted that altruistic behaviors are frequently witnessed in P2P file sharing infrastructures for instance. Selfish behaviors can have a strong impact on the performance of the network. In order to reduce their effect, collaboration among peers must be encouraged.

Data availability: The main goal of a P2P storage system is of course to guarantee the potential retrieval of data. Since data are not stored in a centralized server and since P2P networks are assumed to be very dynamic, the stored data should be available even if peers may leave the network. Data availability can be increased with data redundancy techniques.

Data reliability: Data should not only be available at any time but also preserved on the long term. Data integrity thus has to be ensured and, in case of errors, a peer should be able to detect and repair the compromised data.

2.2. Threat model

The different attacks the P2P data storage and maintenance mechanism is exposed to are:

- *Data destruction:* the destruction of data stored at a holder must be detected as soon as possible. Destruction may be due to generic data corruption or to a faulty or dishonest holder. This threat may result from a voluntary and malicious action or from free riding. In the latter case, peers may not cooperate to the storage service yet claim that they do so. Such cheating holders should be discouraged through remote data integrity verification.
- *Man-in-the-Middle attacks:* the storage phase is exposed to man-in-the-middle attacks whereby a malicious holder *H'* asks another honest holder *H* to store the data for himself and to also generate the response to each challenge that could be generated by the verifier. This way, *H'* seems honest although it does not store any of the data.
- *Collusion attacks:* collusion attacks aim at taking unfair advantage of the storage application. There are two possible attacks. First, replica holders may collude so that only one of them stores data, thereby defeating the purpose of replication to their sole profit. Second, a dishonest verifier may collude with a data holder to help it answer honest verifiers' challenges without storing data.
- *Denial-of-Service (DoS) attacks:* DoS attacks aim at disrupting the storage application. The introduction of security and reliability mechanisms may bring up new ways to attack the storage service, in particular through denial of service attacks (DoS). For instance, the holder may either be flooded by verification requests from dishonest verifiers, or from attackers that have not been delegated by the owner. Malicious verifiers may also flood the network with useless repair messages. In order to prevent the latter type of attacks, a threshold number *t* of honest verifiers is defined: there should be at least *t* verifiers that detect such a problem in the verification phase before generating a new data replica. All such attacks should also be prevented, or at least mitigated.
- *Attacks against the repair phase:* during the repair phase, holders may cheat by performing a bogus data rejuvenation. Verifiers may also play a part in constructing such bogus data.

2.3. An overview of existing approaches

A storage mechanism consists in mainly two phases which are data storage, whereby the owner stores some data at one peer, and data verification, whereby it verifies that the data is actually stored. However, in order to discuss all the requirements described above and to address all above threats, we further refine the storage service into five sequential phases (see Figure 1): during the *selection phase*, potential holders of the data are elected by the data owner who later on stores its data at these holders during the *storage phase*. The owner then appoints verifiers for its remote data during the *delegation phase* and these verifiers periodically check the availability and integrity of the stored data during the *verification phase*. Whenever these verifiers detect any data destruction or corruption, the

repair phase is activated, in which the verifiers generate a new copy of the data with the help of the remaining holders. These phases are described below in more detail.

Selection: Selection is the process through which peers that are asked to store data are elected. The goal of the selection phase is to choose a set of peers that can maintain data availability while consuming minimal bandwidth. There are two possible techniques for holder selection. A *discriminatory selection* determines specific peers chosen such that they satisfy some constraint (for example, they exhibit a correct behavior as described in [14]) or such that they share similar characteristics with the owner like their on-line availability, or dedicated bandwidth (as illustrated in [15]). In contrast, *random selection* is generally used for its simplicity since it is less sophisticated and since it consumes less bandwidth per peer. TotalRecall [16] and the P2P storage system described in [17] rely on a distributed-hash-table (DHT) to randomly select data holders. The selection is realized by randomly choosing a value from the DHT address space and routing to that value. The authors of [17] claim that the random selection mitigates some type of pre-set collusion between these holders. Similarly, [18] analyzes peer selection strategies and proves the positive effects of randomization through the study of a stochastic model of a P2P system under churn. After holders have been selected, the owner can directly contact them for data storage. To mitigate the problem of peers having multiple identities as first described as a Sybil attack in [21], peers joining the system may pay with computational, bandwidth or storage abilities, such as for example crypto-puzzles in [23] (the reader may refer to [22] for an exhaustive survey of counter techniques to the Sybil attack).

Storage: Once peers which will store the data have been selected by the owner, the latter should send the data to these potential holders. Data availability can be ensured either by implementing some form of redundant storage, through either replication or erasure coding. With replication, a simple copy of the data is distributed to each selected peer. With erasure coding, a data is instead divided into several blocks and additional blocks are generated to ensure data reconstruction as soon as a given number of blocks are retrieved. Replication, which has been mostly used in DHTs, more seriously increases the storage overhead and maintenance bandwidth without a comparable increase in fault tolerance. In contrast, erasure codes offer a better balance between the storage overhead and fault tolerance achieved. Many storage systems like [9], [10], [16] and [27] rely on the latter. The overhead introduced by data redundancy can however be coped with. For instance, Wuala [9] reduces the remote storage space allocated to a peer in exchange of an equivalent local storage space based on its probability of being online: the unallocated storage space serves for trading space on other peers in order to achieve a redundant storage [15]. Erasure codes are more complex than replication and in particular, the maintenance of coded data blocks introduces additional computational costs since it requires performing the coding yet again. Communication costs are also needed to retrieve a minimum number of coded blocks from several holders. A tradeoff between storage requirement and data maintenance must be determined when considering the use of erasure codes or replication: [20] for instance describes how quantitative simulation might help in doing so. Moreover, in the case of replication, since the size of the data can be very large, holders may not cooperate and cheat on storing the data. They may even collude and ensure that only one holder is storing the data for all the other selected ones. Data personalization has been introduced as a solution to this threat: the owner generates a single and different replicate of the data for each holder and ensures that the response to the challenge during the verification phase is also different (e.g., [1], [19]). This type of collusion may also arise with erasure coding even though it becomes problematic only if the number of colluding holders exceeds the number of original data blocks.

Delegation: As previously described, the storage mechanism should ensure that the data is continuously available and that holders are rightly claiming to be storing the data assigned to them. The verification phase relies on specific challenge-response protocols that achieve remote data integrity verification. P2P networks being very dynamic, the owner cannot be assumed to be always online, in particular if the storage service is used for backup purposes. During periods where the owner is not present in the network, data verification should still be ensured by owner's delegates. We term such peers as *verifiers*. The owner provides verifiers with some information about the data that we call the security *metadata*, and which serves as a basis for remote data integrity checking. Verification is itself a self-organizing process that may entail peer selfishness, which is mitigated through the distribution of delegated verification to a set of verifiers that are also incented to cooperate. The distribution of that verification also improves performance through load balancing.

Verification: Some storage systems like [26] rely only on data redundancy to achieve data availability. The degree of data redundancy in the network is frequently adjusted in function of locally estimated network dynamic properties. However, such systems provide probabilistic guarantees for data availability that may be sufficient if the degree of data redundancy is considerably high. Otherwise, deterministic guarantees should be provided by regularly checking storage at data holders. P2P storage systems generally use timeouts/heartbeats to detect peer failures. For example, OceanStore [27] utilizes Tapestry (DHT) that uses periodic user datagram protocol (UDP) probes to gauge link conditions in order to detect servers' failures. When the redundancy level for some data has dropped below a critical level, servers owned by OceanStore service providers (OSPs) trigger the recreation and a new dissemination of lost data blocks. Actually, the redundancy level may fall below the acceptable level because of the maliciousness of storage servers without being detected. Probes to sense peer failures are not sufficient to detect storage default at untrusted holders. A new type of challenge-response protocol has been proposed to tackle the problem of remotely proving the integrity of some data stored by a holder (e.g., [1], [2], [3], [4], and [5]) as plain hash functions for instance are not enough. Even though these new cryptographic primitives prevent the generation of correct responses, a cheating holder may simply not reply to a verifier's challenge thereby pretending to be offline or crashed. Distinguishing between permanent failures, malicious or not, and transient ones is difficult. This is generally handled through the use of a *grace period* during which the verifier waits for challenges to be answered before declaring the holder as faulty.

Repair: Detecting that one of the holders has cheated and does not store the data anymore should trigger a data rejuvenation operation in order to ensure data availability. The verifier in charge should select another peer to perform the required operations to store the data and to generate the corresponding security metadata. Given the dynamicity of P2P networks, such operations should not rely on the presence of the owner. Additionally, the cooperative behavior of the peers participating in the repair operations should be stimulated. The recovery may be triggered almost immediately after the detection of a cheating or delayed holder. Simulation results of [16] demonstrate that delayed repair (lazy repair) is more efficient in terms of data availability and overhead costs tradeoff than immediate repair (eager repair) for a large data size and a highly dynamic system.

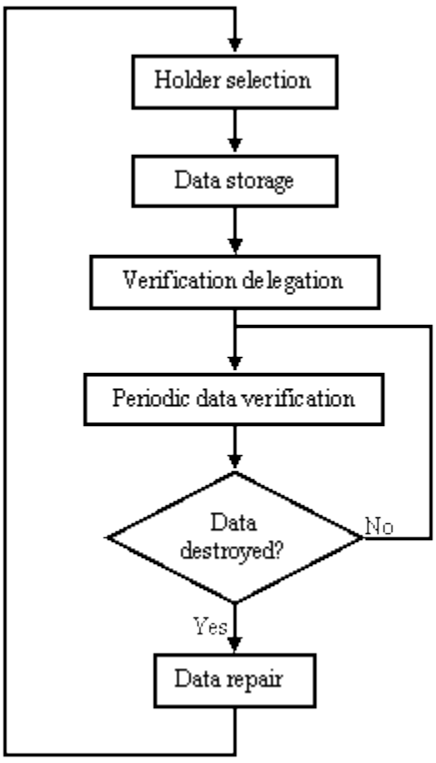

Figure 1. Data storage and maintenance phases

Since holder selection has been widely studied in the literature ([14], [15], [16], [17], and [18]), this paper focuses on storage, verification, and repair phases. We introduce a new data storage and maintenance scheme that does not rely on a centralized entity nor require the data owner to be online during data maintenance.

2.4. Contributions of this paper

The contributions of the paper cover several challenging issues that have not been addressed in previous work on P2P storage:

- In addition to peer-to-peer storage, which is not new in itself, the scheme proposed in this paper aims at realizing both the monitoring and maintenance services in a self-organized way. To our knowledge, the proposed scheme is the only one which ensures data integrity and offline maintenance at the same time. As opposed to existing reliability mechanisms [27], the new protocol does not require the presence of the data owner for maintenance.
- The security of the proposed scheme does not rely on the presence of trusted entities, but rather on a fault-tolerant and threshold-based approach to handling the maliciousness and selfishness of data holders. This paper in particular makes use of an epidemic model to validate the solution proposed. That model proves that the P2P storage system converges to a stable state that fulfills data reliability and availability requirements thanks to the proposed maintenance scheme is operated.

3. Erasure coding based storage and maintenance

This section introduces a new data storage and maintenance protocol for P2P storage systems. Like other systems, the proposed protocol uses erasure coding rather than replication for performance reasons (see Section 3.4). In contrast with these approaches however, the proposed protocol combines a data integrity check protocol with the offline maintenance. To our knowledge, the presented protocol is the only maintenance method proposed so far that takes into consideration the maliciousness of holders (compared to [27]) and that additionally does not require the intervention of the data owner (e.g., [26]) or a trusted server (e.g., [9]).

3.1. Cryptographic background

The security of the following protocol relies on a cryptographic function *f* that realizes a homomorphism. The function *f* is used to map each generated erasure coding block to some piece of metadata that is used for verifying the integrity of the block. For instance, for a coded block d_i, the corresponding metadata is $f(d_i)$ that should be as well a block digest to be stored at appointed verifiers.

The homomorphism is used to preserve linear combination i.e. for each set of erasure coding blocks $\{d_i\}_{1\leq i\leq k}$ and random coefficients $\{c_i\}_{1\leq i\leq k}$ we have:

$$f\left(\sum_{i=1}^{k} c_i \times d_i\right) = \sum_{i=1}^{k} c_i \times f(d_i)$$

There are several possible implementations of the homomorphic function *f* for data integrity verification purposes. It can be constructed based on the Discrete Logarithm problem like in [4] or [5]. It may also rely on elliptic curves as proposed in [1].

3.2. Scheme description

Only the four phases of the proposed protocol are described in this section. The selection phase can simply rely on a random selection as suggested in [17].

- *Storage:* As discussed in Section 2, data have to be stored at multiple peers in order to ensure data availability and reliability. Secure data storage with the simple replication technique has been proposed and evaluated in [1]. Since the use of erasure coding techniques provides the same level of reliability as replication but with much lower storage requirements at holders, a new storage mechanism based on erasure codes is proposed. At this

phase (Figure 2), the data D is first divided into k blocks $\{d_i\}_{1\leq i\leq k}$. These blocks are then encoded to produce $k+m$ coded blocks $\{b_i\}_{1\leq i\leq k+m}$ such that at least any of these k blocks enable the recovery of the original data. Blocks are coded using random linear erasure codes [12]. With such erasure codes, the generating matrix G is defined in $\mathbb{Z}$ as:

$$G = \begin{bmatrix} I_k \\ A \end{bmatrix}$$

where I_k denotes the $k\times k$ identity matrix and A denotes a $m\times k$ random matrix in $\mathbb{Z}$ (i.e., entries of A are chosen randomly). Each coded block b_i is generated using the following linear operations in $\mathbb{Z}$:

$$b_i = \sum_{j=1}^{k} \alpha_{i,j} \times d_j$$

where the $\alpha_{i,j}$ is an entry of G at the i^{th} row and j^{th} column. The coded block b_i is finally sent by the owner to one distinct holder that is named the i^{th} holder.

1) Owner: divide D into k blocks $\{d_i\}_{1\leq i\leq k}$
2) Owner: generate random numbers in $\mathbb{Z}$ $\{\alpha_{i,j}\}_{1\leq i\leq k+m, 1\leq j\leq k}$
3) for each $1\leq i\leq k+m$
   Owner: compute $b_i = \sum_{j=1}^{k} \alpha_{i,j} \times d_j$
4) for each $1\leq i\leq k+m$
   Owner $\rightarrow$ i^{th} holder: b_i
5) for each $1\leq i\leq k+m$
   i^{th} holder: keep b_i

Figure 2. Storage phase

- *Delegation:* The owner delegates (in Figure 3) the verification task of one coded block to a number v of verifiers (it is possible that a verifier may be responsible of several coded blocks). The verifier assigned to the i^{th} holder receives from the owner a metadata information T_i such that $T_i=f(b_i)$ where b_i is an integer that maps to the coded block stored at the holder (we call $(i, j)^{th}$ verifier, one verifier of the set of v verifiers assigned to the i^{th} holder).

1) for each $1\leq i\leq k+m$
   Owner: compute $T_i= f(b_i)$
2) for each $1\leq i\leq k+m$
   for each $1\leq j\leq v$
   Owner $\rightarrow$ $(i, j)^{th}$ verifier: T_i
3) for each $1\leq i\leq k+m$
   for each $1\leq j\leq v$
   $(i, j)^{th}$ verifier: keep T_i

Figure 3. Delegation phase

- *Verification:* Based on the stored metadata, the verifier is able to periodically check whether the holder stores the block b_i or not (in Figure 4). This verification only guarantees the storage of one block and therefore considered as partial. The verifier first sends a challenge message $Q=f(r)$ to the holder where r is a freshly generated random number. Upon reception of the challenge, the holder computes $R=b_i.Q$ and sends the product to the verifier. The verifier checks if the equality $R = r.T_i$ holds. Indeed, thanks to the homomorphism of the function f, holder's response can actually be written as:

$$R = b_i.Q = b_i.\, f(r) = f(b_i.r) = r.\, f(b_i) = r.T_i$$

If the latter equality is not met, the verifier detects that the block has been either corrupted or destroyed by the holder. A reaction to this event should result in the regeneration of new block to replace the lost one. This operation is performed in the next phase.

1) $(i, j)^{th}$ verifier: generate a random number r
2) $(i, j)^{th}$ verifier: compute $Q=f(r)$
3) $(i, j)^{th}$ verifier $\rightarrow$ i^{th} holder: Q
4) i^{th} holder: compute $R=b_i.Q$
5) i^{th} holder $\rightarrow$ $(i, j)^{th}$ verifier: R
6) $(i, j)^{th}$ verifier: check if $R=r.T_i$?
If $R\neq r.T_i$ launch a repair phase

Figure 4. Verification phase

- *Repair:* To activate this phase, a fraction of verifiers assigned to a given holder consisting of at least t peers detect the destruction of the block stored at the very holder. They first select a new holder randomly as explained in [17]. The new block is generated based on a coding operation over k blocks. The coding operation is executed by the new holder (Figure 5). This latter receives k verified blocks $\{b_{t_i}\}_{1\leq i\leq k}$ from randomly selected set of the remaining holders. The verifiers also agree on a seed s that will be sent to the new holder. The seed can be simply computed as a sum of random numbers each one of them chosen by each verifier. The seed allows to generate random coefficients $\{c_i\}_{1\leq i\leq k}$. The new holder then computes the new block b' in $\mathbb{Z}$ as follows:

$$b' = \sum_{l=1}^{k} c_l \times b_{t_l}$$

The new generated block can be written as a linear combination of the original data blocks. Indeed, since each block transmitted by the holders participating in the generating process can be written as a combination of the original data blocks, then:

$$b' = \sum_{l=1}^{k} c_l \times b_{t_l} = \sum_{l=1}^{k} c_l \times \left(\sum_{j=1}^{k} \alpha_{t_l,j} \times d_j \right)$$

Thus,

$$b' = \sum_{j=1}^{k} \left(\sum_{l=1}^{k} c_l \times \alpha_{t_l,j} \right) \times d_j$$

As a result, the generated block is coded based on the random linear erasure coding scheme. [12] demonstrates that any $k\times k$ sub-matrix of a random matrix is invertible with high probability for a large field size; thus the property of erasure coding is still provided by the new block.

1) Verifiers: generate a seed s
2) Verifiers: select k random holders
3) Verifiers $\rightarrow$ new holder: s, $\{b_{t_i}\}_{1\leq i\leq k}$
4) New holder: generate random coefficients $\{c_i\}_{1\leq i\leq k}$
5) New holder: compute $b' = \sum_{l=1}^{k} c_l \times b_{t_l}$
6) New holder: keep b'

Figure 5 Repair phase: construction of a new coded block.

Moreover, the new block is distinct from the lost block and the remaining blocks stored in the system as well. The indexes of the holders involved in the redundant block generation process along with the seed s must be kept stored by the verifiers before being handed out to the owner allowing this latter to update the generating matrix G of the erasure codes. If the block b_i has been destroyed, the update only affects the i^{th} row of the matrix: the new row is defined as $(\alpha'_{i,1}, \ldots, \alpha'_{i,k})$ where for each j in [1, k]:

$$\alpha'_{i,j} = \sum_{l=1}^{k} c_l \times \alpha_{t_l,j}$$

Each verifier assigned to the revoked holder keeps its role as a verifier for the new holder. However, it requires new metadata information *T*' for the new coded block (Figure 6) that is computed as a linear combination over the metadata information stored at other verifiers (responsible of the holders that have been involved in the block generation) and using the same set of coefficients:

$$T' = \sum_{l=1}^{k} c_l \times T_{t_l}$$

The new metadata corresponds to the new block *b*' stored at the new holder; this is realized owing to the homomorphism of *f*:

$$T' = \sum_{l=1}^{k} c_l \times T_{t_l} = \sum_{l=1}^{k} c_l \times f(b_{t_l}) = f\left(\sum_{l=1}^{k} c_l \times b_{t_l}\right) = f(b')$$

1) Verifiers→ new holder's verifiers: *s*, $\{T_{t_i}\}_{1\le i\le k}$
2) New holder's verifiers: generate random coefficients $\{c_i\}_{1\le i\le k}$
3) New holder's verifiers: compute $T' = \sum_{l=1}^{k} c_l \times T_{t_l}$
4) New holder's verifiers: keep *T*'

Figure 6. Repair phase: construction of new metadata.

The owner and verifiers that have been offline for a while must contact the rest of verifiers to update their list of holders and all related information whenever they reconnect to the network. This update process can be automated for instance if verifiers periodically exchange information with each another. To guarantee a consistent update process, a sufficient number of (at least *t*) verifiers should be contacted before the update is effective. Alternatively, a tracking server might be established by a trusted authority such as a service provider: that tracker would collect and maintain the location of blocks and address of data holders, similarly to P2P file sharing trackers [8] that point at content repositories. Upon request, the tracker might then provide such information to crashed data owners or to new verifiers. Solutions based on the establishment of a DHT to keep track of data holders might also be envisioned in order to implement such a holder repository in a self-organized way, although such an approach would certainly open the door to Sybil or pollution attacks similar to what P2P file sharing systems experience.

3.3. Security evaluation

This section analyses the security of the proposed data storage and maintenance protocol with respect to the attacks discussed in section 2.2.

Preventing data destruction: Each verifier checks the availability of one remote coded block. The destruction of any block is detected through the periodic verification. The verification protocol is a proof of knowledge protocol. Indeed, the protocol is complete thanks to the homomorphism of *f* as explained earlier i.e., the verifier always accepts the proof as valid if the holder follows the protocol. The protocol is sound i.e., the verifier will not accept the proof as valid if the holder destroys or corrupts the block. Because we assume that the homomorphism *f* is also one-way, the holder cannot infer *r* from the challenge *Q*=*f*(*r*) and should keep the whole block to correctly answer verifier's challenge. The verification protocol constructed using elliptic curves has been proved as being a proof of knowledge protocol in [1].

Collusion resistance: With erasure codes, the produced blocks that will be stored at holders inherently differ from each other; even though collusion between *k*+1 or more holders may succeed. This type of collusion is however less

likely because it requires the exchange of *k* blocks (comparable in size to the original data) to one of the holders to encode the destroyed block. This entails considerable bandwidth and computation costs for each verification operation.

A new generated block with the repair phase differs from the remaining stored blocks. This is guaranteed with the randomization added by the seed *s* that is chosen cooperatively by the verifiers to limit any collusion between them.

Preventing DoS attacks: A quota system can be introduced into the protocol to regulate the number of challenge messages the verifier is allowed to send to a given holder during a time frame. This allows mitigating a flooding attack against the holder launched by a malicious verifier.

The activation of a repair phase is made possible only with the association of at least *t* verifiers. The distribution of the work to independent verifiers allows mitigating the maliciousness of some of them that may flood the system with repair requests. The threshold value *t* is a tradeoff factor between these two considerations: prevention against verifier collusion and also handling peer intermittence.

Preventing attacks against the repair phase: A holder or verifier may send bogus information to the concerned peers. We argue that this problem can be easily thwarted by including in any information signature that provides proofs of origin and integrity for the recipient of such information. Each coded block or metadata is associated with some owner signature that attests its validity.

On the other hand, after a data repair phase, the new holder or the new verifier will keep track of a compilation of all necessary owner signatures that validates the new generated block or metadata. For example, a new holder can keep along with the freshly coded block *b*', the seed *s* used to generate coefficients for the new block and also the following set of information:

$$\left\{f(b_{t_i}),\ sign_{owner}(f(b_{t_i}))\right\}_{1\le i\le k}$$

having *b*' coded based on the blocks $\{b_{t_i}\}_{1\le i\le k}$ and $sign_{\text{owner}}(.)$ the signature generated by the owner. When the owner reconnects to the network and updates its holder related information (list of holders, generating matrix), it makes contact with the new holder, checks the validity of its signature compilation and replaces it with its corresponding signature:

$$sign_{owner}\left(f(b') = \sum_{l=1}^{k} c_l \times f\left(b_{t_l}\right)\right)$$

To simplify such process and particularly to reduce the signature overhead, the protocol may alternatively rely on a homomorphic signature (e.g., algebraic signature [24]) that provides the following property:

$$sign_{owner}\left(b' = \sum_{l=1}^{k} c_l \times b_{t_l}\right) = \sum_{l=1}^{k} c_l \times sign_{owner}(b_{t_l})$$

Thus with this type of signatures, the new holder or verifier can construct a valid signature for the new generated block or metadata based on the generating blocks or metadata' signatures and without having recourse to the owner.

Preventing man-in-the-middle attacks: The holders of a given data are selected randomly. As suggested in [17], a data owner cannot choose by itself the identities of its data holders. This means that the owner have necessarily a different key ID in the DHT distinct from the key IDs of its holders. These holders are then contacted directly for data storage and verification. To prevent a man-in-the-middle attack, the response of a holder storing the block b_i to a verifier's challenge may be constructed as a digest of the metadata $f(b_i)$ along with holder's identity *ID*: $R=hash(f(b_i), ID)$ having *hash* a pseudo-random one-way function. The peer's *ID* can correspond to the peer's IP address, which is forgeable but may still make it possible to establish a secure channel between two peers if we assume no attack on the routing protocol. With this construction of the response, an attacker cannot trick a verifier by pretending to be storing the block while holding at the same time an identity different from *ID*.

3.4. Performance evaluation

In the proposed protocol, the performance of both the delegation and verification phases in the particular case of elliptic curves has been already evaluated in [1]. Since metadata are only computed based on one block instead of the whole data, the performance in fact improves.

The proposed repair method requires the transmission of exactly *k* coded blocks (amounting to the file size) only once for the regeneration of one block and provides a personalized regenerated block to the new holder.

The bandwidth required for the repair is distributed between holders. Furthermore, the communication overhead of the proposed repair method can be optimized by relying on hierarchical codes as proposed by [13]. With such erasure codes, the required number of coded blocks to repair a block equals at least the number of children in the tree hierarchy and at most equals to *k*.

Any other communication overhead caused by verifier agreement and notification messages can be considered negligible owing to the fact that the data (or the block) is considerably larger.

The linear combination of blocks is operated in $\mathbb{Z}$ which may lead to an increase in the size of the produced block by at maximum *k* bits. We argue that this increase is insignificant given the original size of blocks.

4. An analytic model for P2P data storage and maintenance

This section introduces an analytic model describing the P2P storage system inspired from the epidemic models in [25] where peers are classified into groups depending on their state. The epidemic model allows to relate peer dynamics (e.g., peer leaves, disconnections) at the network population level to holders' behavior (data conservation or destruction) and data maintenance at peer level. Thanks to the use of such model, it is easy to prove whether the system finally converges to a stable state (i.e., equilibrium) or not. With such model, we endeavor to determine the right periodicity for data maintenance.

We consider an owner that replicates its data at *r* holders using a (*k*, *r*-*k*)-erasure coding. The original data can be generated from at least *k* coded blocks. The owner also delegates the verification of each coded block to *v* verifiers. These verifiers have the responsibility to periodically check the presence and integrity of the stored block at their assigned holder. Whenever at least *k*' verifiers detect the destruction or corruption of the block they decide to regenerate a new block and store it at a new holder. We assume that verifiers do not require to be replaced often during the data storage. The owner is supposed to connect to the P2P system from time to time in order to select new verifiers and to appoint them to the desired blocks.

4.1. Model of P2P data storage without data maintenance

Figure 7 depicts a state model of the uptime of holders in the P2P system. We consider three states: "connected", "disconnected", and "left". Initially, the owner appoints some connected peers to be holders of its data. These holders may afterwards disconnect or definitely leave the system. Peer disconnection and peer departure rates, which are respectively named λ and μ, are considered constant. Disconnected holders may reconnect at constant rate λ'. Additionally, holders do not just leave the system after a crash but may also pertain to state "left" if they destroy blocks they store.

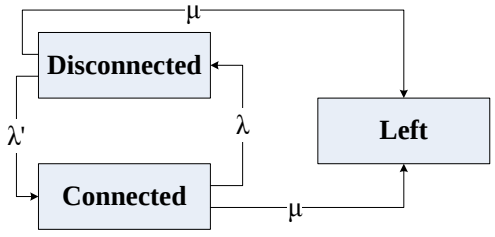

Figure 7. State model of data storage without maintenance

In this first model, we consider that holders that have destroyed blocks are not replaced (no data maintenance). We characterize each state by the number of holders at a given time *t* that happen to be in such state at *t*. For a given time *t*, such number has a random distribution of values but the value of its average is quantifiable based on the epidemic model. We designate the average number of holders in states "connected", "disconnected", and "left" at time *t* by respectively $n_c(t)$, $n_d(t)$, and $n_l(t)$, the total number of holders being:

$$n_c(t) + n_d(t) + n_l(t) = r$$

The model does not describe the system at a stable state but shows if there is convergence of the system to such equilibrium. Indeed, the average number of holders in each state varies with time according to the differential equations derived from the state model. There are holders that come into the "connected" state when they reconnect and holders that depart from that state when they disconnect, destroy data, or even leave definitively the network (equation 4.a.1):

$$\frac{dn_c(t)}{dt} = \lambda' n_d(\mathrm{t}) - (\mu + \lambda) n_c(t)$$

Holders that go offline go into the "disconnected" state and holders that go online or destroy their data or leave the network depart from that very state (equation 4.a.2):

$$\frac{dn_d(t)}{dt} = \lambda m_c(\mathrm{t}) - (\mu + \lambda') n_d(t)$$

Holders that leave or maliciously destroy data enter into the "left" state (equation 4.a.3):

$$\frac{dn_l(t)}{dt} = \mu(n_c(\mathrm{t}) + n_d(t))$$

The solution of these equations gives the average number of holders in each state at time *t*. Initially (at *t*=0), all holders are in the "connected" state:

$$n_c(0) = r,\ n_d(0) = 0,\ n_l(0) = 0$$

Thus, we obtain:

$$n_c(t) = \frac{\lambda' r}{\lambda + \lambda'} e^{-\mu t} + \frac{\lambda r}{\lambda + \lambda'} e^{-(\mu+\lambda+\lambda')t}$$
$$n_d(t) = \frac{\lambda r}{\lambda + \lambda'} e^{-\mu t} - \frac{\lambda r}{\lambda + \lambda'} e^{-(\mu+\lambda+\lambda')t}$$
$$n_l(t) = r(1 - e^{-\mu t})$$

The number of holders in the system $n_c(t)+n_d(t)$ converges to zero with time. This means that there is a certain time t_0 at which the owner's data is not available any more (i.e., t_0 is the time limit for data availability), and there is another time t_1 at which the owner cannot retrieve its data from the storage system (i.e., t_1 is the time limit for data reliability). The time limit t_0 is defined as:

$$n_c(t_0) = \frac{\lambda' r}{\lambda + \lambda'} e^{-\mu t_0} + \frac{\lambda r}{\lambda + \lambda'} e^{-(\mu+\lambda+\lambda')t_0} = k$$

To simplify the above equation, the exponential functions can be rewritten as infinite power series (in the form of Taylor series) that can be approximated to their first order. A solution is obtained for:

$$t_0 \sim \frac{1}{\mu + \lambda}(1 - k/r)$$

The time limit t_1 is obtained if:

$$n_c(t_1) + n_d(t_1) = re^{-\mu t_1} = k$$

This leads to:

$$t_1 = \left({}^{1}/_{\mu}\right) \log\left(\frac{r}{k}\right)$$

4.2. Model of P2P data storage with data maintenance

If we consider that destroyed blocks are detected and regenerated at other new holders, we obtain a new state model (depicted in Figure 8). This model describes a repair phase for destroyed blocks during which new holders are introduced in the model at a constant rate γ ($1/\gamma$ is also the verification time period). We assume that verifiers can distinguish transient failures (i.e., disconnection) from permanent failures (i.e., data destruction or leave). The number of new holders that are regularly added to recover from data destruction depend on the number of holders in "left" state which consists of holders that either have maliciously destroyed data or have definitively left the network. The frequency at which such new holders are added depend on the rate upon which "left" holders are detected i.e., the maintenance (verification) rate.

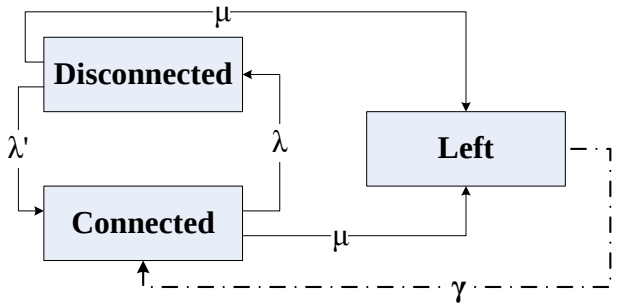

Figure 8. State model of data storage and maintenance

The number of holders in each state verifies the following differential equations derived from the model (equations 4.b):

$$\frac{dn_c(t)}{dt} = \lambda' n_d(\mathrm{t}) + \gamma(r - n_c(\mathrm{t}) - n_d(\mathrm{t})) - (\mu + \lambda) n_c(t)$$

$$\frac{dn_d(t)}{dt} = \lambda n_c(\mathrm{t}) - (\mu + \lambda') n_d(t)$$

$$\frac{dn_l(t)}{dt} = \mu\big(n_c(\mathrm{t}) + n_d(t)\big)$$

Compared to equation 4.a, the new equations 4.b takes into account the fact that new online holders with number depending on the number of blocks detected being lost are added to the system i.e., there are peers that depart from the "left" state and come into the "connected" state.

The number of the total considered holders increases with time since we regularly add new holders with data maintenance:

$$\frac{d(n_c(t) + n_d(t) + n_l(t))}{dt} = \gamma(r - n_c(\mathrm{t}) - n_d(\mathrm{t}))$$

The solution to these differential equations gives the number of holders in each state:

$$n_c(t) = \frac{r}{\gamma + \mu}\left[\frac{\gamma(\mu + \lambda')}{\mu + \lambda + \lambda'} + \frac{\mu(\lambda' - \gamma)}{\lambda + \lambda' - \gamma} e^{-(\gamma+\mu)t} + \lambda\left(\frac{\gamma}{\mu + \lambda + \lambda'} + \frac{\mu}{\lambda + \lambda' - \gamma}\right) e^{-(\mu+\lambda+\lambda')t}\right]$$

$$n_d(t) = \frac{\lambda r}{\gamma + \mu}\left[\frac{\gamma}{\mu + \lambda + \lambda'} + \frac{\mu}{\lambda + \lambda' - \gamma} e^{-(\gamma+\mu)t} - \left(\frac{\gamma}{\mu + \lambda + \lambda'} + \frac{\mu}{\lambda + \lambda' - \gamma}\right) e^{-(\mu+\lambda+\lambda')t}\right]$$

$$n_l(t) = \frac{\mu r}{\gamma + \mu}\left[\gamma t + \frac{\mu}{\gamma + \mu}(1 - e^{-(\gamma+\mu)t})\right]$$

We note that the equations 4.a match the above equations for $\gamma = 0$. To be able to perform the recovery of dropped blocks, we should have $n_c(t) \geq k$ for each $t>0$. This leads to the following inequality that must be met:

$$\gamma \geq \frac{\mu}{\frac{r}{k}\left(\frac{\mu+\lambda'}{\mu+\lambda+\lambda'}\right)-1};\ \frac{k}{r} < \frac{\mu+\lambda'}{\mu+\lambda+\lambda'}$$

The above equation gives a precise bound on the data maintenance period T_{max}:

$$T_{max} = \frac{\frac{r}{k}\left(\frac{\mu+\lambda'}{\mu+\lambda+\lambda'}\right)-1}{\mu}$$

4.3. Numerical simulation

We simulated the above model of a P2P data storage system in different scenarios based on the equations developed earlier. In the simulation, peers join the system for an average lifetime of 2 weeks. Each peer stays online for 1 hour and connects on average 6.4 times in a day. Additionally, holders are assumed to destroy their blocks one time per day on average.

Without data maintenance, the owner's data is not available after only 49 minutes and then cannot be recovered after less than 2 days. With maintenance however, the data is always available and retrievable. But data maintenance should be periodically performed 3.8 times per day.

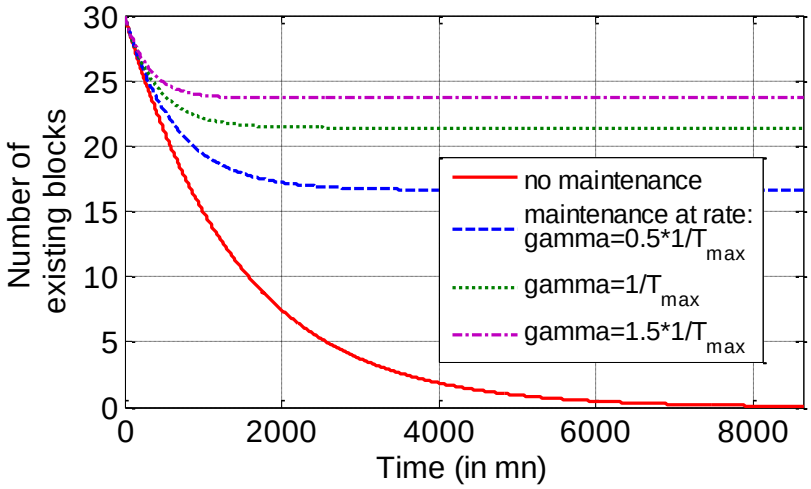

Figure 9. Number of holders. r=30, k=5, v=10, k'=7, μ=5×10^{-5}, λ=0.0167, λ'=0.0044 (rates per mn).

Figure 9 shows the average number of holders that are still storing data blocks with time in different settings. The figure illustrates the fact that without maintenance (γ=0) the number of holders decreases converging to zero. On the other hand, with maintenance ($\gamma \neq 0$), the number of holders converges to an equilibrium value that is not null. This value depends on the γ ratio: if $\gamma < 1/T_{max}$, then the value is lower than k; otherwise it is higher than k thus rendering the restoration of destroyed blocks possible. Our results prove that with the data maintenance mechanism, the P2P storage system is able to achieve survivability for stored data.

Figure 10 shows the number of online holders over time computed with and without a repair phase. Considering maintenance at a rate $\gamma \geq 1/T_{max}$ and with k chosen in function of system parameters, the system can ensure a high data availability. In the case without data maintenance however, blocks required to recover the data are not accessible anymore over time.

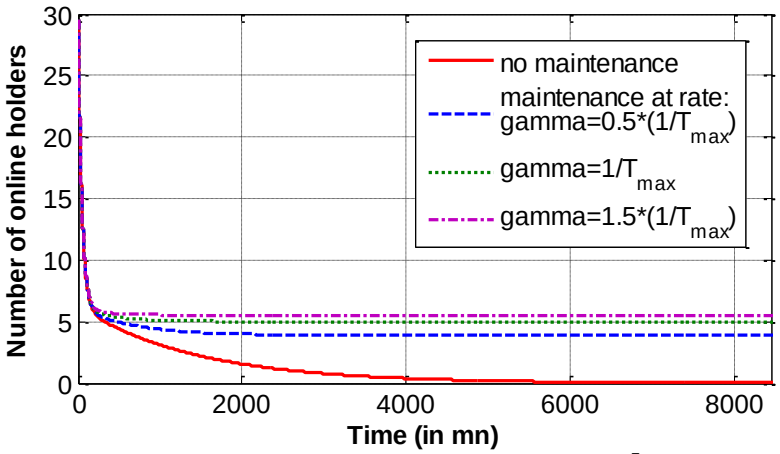

Figure 10. Number of online holders. *r*=30, *k*=5, *v*=10, *k'*=7, $\mu=5\times10^{-5}$, $\lambda=0.0167$, $\lambda'=0.0044$ (rates per mn).

5. Conclusion

This paper introduced a P2P data storage and maintenance protocol in which the detection of data corruption and data rejuvenation are self-organizing functions.

Data corruption detection in our approach relies on the combination of homomorphism based remote data integrity verification techniques with the generation of smaller blocks through erasure coding. Detecting data corruption is critical for preventing both selfish behaviors, which reduce the effectiveness of the storage service, and malicious attacks, which are aimed at ruining it. The techniques presented in this paper specifically make it possible to lessen verification costs through the delegation of verifications to various peers.

This protocol is the first to use peer cooperation not only for providing storage resources, but also for ensuring storage resilience. In particular, the main verification and maintenance functions of this protocol are distributed to multiple peers, which makes it easier to mitigate non-cooperative behaviors while coping with churn. This protocol also ensures data availability and reliability through the use of erasure codes, which serve as a basis to ensure both storage diversity and data rejuvenation while enabling to decide on the tradeoff between redundancy and storage overhead.

References

[1] Nouha Oualha, Melek Önen, and Yves Roudier. A Security Protocol for Self-Organizing Data Storage. 23rd International Information Security Conference (IFIP SEC 2008), Milan, Italy, September 2008.

[2] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson and Dawn Song. Provable data possession at untrusted stores. In Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, 598-609.

[3] Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. Remote Integrity Checking. In Proceedings of Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS), 2004.

[4] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, Report 2006/150, 2006. http://eprint.iacr.org

[5] Francesc Sebe, Josep Domingo-Ferrer, Antoni Martínez-Ballesté, Yves Deswarte, and Jean-Jacques Quisquater. Efficient Remote Data Possession Checking in Critical Information Infrastructures. IEEE Transactions on Knowledge and Data Engineering, 06 Aug 2007. IEEE Computer Society Digital Library. IEEE Computer Society, 6 December 2007 http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.190647

[6] Ari Juels and Burton S. Kaliski PORs: Proofs of retrievability for large files. Cryptology ePrint archive, June 2007. Report 2007/243.

[7] KaZaA. http://www.kazaa.com/

[8] BitTorrent. http://www.bittorrent.com/

[9] Wuala. http://wua.la/en/home.html

[10] AllMyData Tahoe. http://allmydata.org/

[11] UbiStorage. http://www.ubistorage.com/

[12] Szymon Acedański, Supratim Deb, Muriel Médard, and Ralf Koetter. How good is random linear coding based distributed networked storage?. In Proceeding of 1st Workshop on Network Coding, WiOpt 2005 Riva del Garda, Italy, April 2005.

[13] Alessandro Duminuco and Ernst W. Biersack. Hierarchical codes: how to make erasure codes attractive for peer-to-peer storage systems. The 8th IEEE International Conference on Peer-to-Peer Computing (P2P'08), September 8th-11th, 2008, Aachen, Germany.

[14] Roger R. Dingledine. The Free Haven project: Design and deployment of an anonymous secure data haven. Master's thesis, MIT, June 2000.

[15] Laszlo Toka and Pietro Michiardi. Analysis of user-driven peer selection in peer-to-peer backup and storage systems. GameComm 2008, 2nd ACM-Valuetools International Workshop on Game theory in Communication networks, October 20, 2008, Athens, Greece, pp 428.

[16] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, G. M. Voelker. TotalRecall: System Support for Automated Availability Management. ACM/USENIX NSDI, 2004.

[17] Nouha Oualha and Yves Roudier. Reputation and Audits for Self-Organizing Storage. In the 1st Workshop on Security in Opportunistic and SOCial Networks (SOSOC 2008), Istanbul, Turkey, September 2008.

[18] P. Brighten Godfrey, S. Shenker, I. Stoica. Minimizing churn in distributed systems. ACM SIGCOMM CCR, Vol. 36, N. 4, 2006.

[19] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Mike Burrows, and Michael 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.

[20] Hakim Weatherspoon, Byung-Gon Chun, Chiu Wah So, John Kubiatowicz. Long-term data maintenance in wide-area storage systems: A quantitative approach. Technical Report (2005) UCB/CSD-05-1404, EECS Department, University of California, Berkeley.

[21] John R. Douceur. The Sybil attack. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02). MIT Faculty Club, Cambridge, MA, 2002.

[22] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A Survey of Solutions to the Sybil Attack. Technical Report 2006-052, University of Massachusetts Amherst, Amherst, MA, October 2006.

[23] Vivek Vishnumurthy, Sangeeth Chandrakumar and Emin Gun Sirer. KARMA: A Secure Economic Framework for P2P Resource Sharing. In Proceedings of the Workshop on the Economics of Peer-to-Peer Systems, Berkeley, California, June 2003.

[24] Thomas Schwarz, and Ethan L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In Proceedings of the IEEE Int'l Conference on Distributed Computing Systems (ICDCS '06), July 2006.

[25] Douglas Samuel Jones and B. D. Sleeman. Differential Equations and Mathematical Biology. London: Allen & Unwin, 1983.

[26] Christof Leng, Wesley W. Terpstra, Bettina Kemme, Wilhelm Stannat and Alejandro P. Buchmann. Maintaining replicas in unstructured P2P systems. CoNEXT, page 19. ACM, 2008.

[27] Sean Rhea, Chris Wells, Patrick Eaton, Dennis Geels, Ben Zhao, Hakim Weatherspoon, and John Kubiatowicz. Maintenance-free global data storage. IEEE Internet Computing, pp. 40–49, September 2001.