Off-line incentive mechanism for long-term P2P backup storage

Marco Gramaglia[a,b,*], Manuel Urueña[b], Isaias Martinez-Yelmo[c]

[a] *Institute IMDEA Networks. Av. Mar Mediterraneo 22. E-28912 Leganés (Madrid). Spain*
[b] *Universidad Carlos III de Madrid. Av. Universidad 30. E-28911 Leganés (Madrid). Spain*
[c] *Universidad de Alcalá, Escuela Politécnica Superior. Campus Universitario, N-II Km 33,6. E-28871 Alcalá de Henares (Madrid). Spain*

Abstract

This paper presents a micro-payment-based incentive mechanism for long-term peer-to-peer storage systems. The main novelty of the proposed incentive mechanism is to allow users to be off-line for extended periods of time without updating or renewing their information by themselves. This feature is enabled through a digital cheque, issued by the user, which is later employed by the peers to get a gratification for storing the user's information when the user is off-line. The proposed P2P backup system also includes a secure and lightweight data verification mechanism. Moreover, the proposed incentive also contributes to improve the availability of the stored information and the scalability of the whole system. The paper details the verification and cheque-based incentive mechanisms in the context of a P2P backup service and analyzes its scalability and security properties. The system is furthermore validated by means of simulation, proving the effectiveness of the proposed incentive.

Keywords: Peer-to-Peer (P2P), Long-term Storage, Incentive Mechanism, Micro-Payment, Digital Cheque, P2P Backup

1. Introduction

The increasing number of consumer devices that can generate all kinds of digital media (e.g. audio, video, photos) has worsened the old problem of safely storing all these space-consuming data. To relieve users from the laborious and eventually expensive task of maintaining their own dedicated storage hardware, in the past few years many on-line storage services made their appearance on the market, ranging from the most basic ones like *Dropbox*[1] to more complex and professional-oriented ones like *Amazon S3*[2]. The growing success of new networking paradigms such as peer-to-peer

*Corresponding author. Address: Avda. Mar Mediterraneo, 22. 28918 Leganés, Madrid (Spain). Tel: +34 91 4816210 . Fax: +34 91 4816965.

Email addresses: marco.gramaglia@imdea.org (Marco Gramaglia), muruenya@it.uc3m.es (Manuel Urueña), isaias.martinezy@uah.es (Isaias Martinez-Yelmo)

[1] http://www.dropbox.com

[2] http://aws.amazon.com/s3/

or, more recently, cloud computing, is offering more tools to tackle this storage problem.

To use a peer-to-peer (P2P) paradigm radically changes the nature of the solutions to this problem, offering some advantages (i.e. utilization of unexploited space in users' hard disks, tunable level of reliability, low cost, etc.), but posing other challenges regarding security, privacy and a fair use of the network. Some P2P-based solutions are already present in the market: LaCie's Wuala[3] or Fiabee[4] exploit this paradigm to offer on-line storage services at a reduced price.

One of the storage services that may benefit most from a distributed P2P architecture is backup, because data is replicated and stored in the hard disks of many different users, distributed worldwide. Thus, it can be hardly affected by a single failure or even a set of failures that may otherwise wipe out a local backup or even a whole data center. Of course, any existing P2P distributed file system could be employed as the basis of a P2P backup service. However, there are two specific characteristics that play a major role in P2P backup: the presence of a local copy, and looser access-time constraints. In a distributed file system, usually, the information is just saved in remote hard disks in order to offload the local one and to better balance their utilization. In a network-based backup solution this assumption is no longer true: the user always has a local copy of the data in order to continue working and updating it. The network backup will only be used in case that some failure happens to the local one. The second key difference of a P2P backup system are the access-time requirements. P2P-based file systems impose hard time constraints (in order to guarantee the performance of input/output operations), while in a P2P backup system these timing constraints are much less strict. A user could tolerate some extra time as long as the backup is completely restored in a reasonable period of time.

In this paper we present an incentive mechanism based on micro-payments and digital cheques for long-term P2P storage systems, such as a P2P backup service. In our proposal, a user pays other peers to store its backup data, whereas charges other (possibly different) peers for using their local hard disk. This kind of monetary incentive approach has already been proposed for different P2P applications [1, 2, 3, 4, 5, 6] and, even if it is a hidden market, for a P2P backup service [7]. We extend this micro-payment incentive framework by means of digital cheques to motivate peers to keep storing backup data even when the owner (user) is off-line for an extended period of time. Albeit the presented schemes also mentioned the problem of long-term availability, they were just focused on employing redundancy techniques to minimize the impact of a lost chunk due to a failure of an, otherwise well-behaved, peer. To the best of our knowledge, this is the first proposal that tackles the problem of long-term availability in a P2P backup system with selfish peers. These selfish peers can deliberately erase a chunk when the owner (user) goes off-line to free their local resources. Our proposal introduces the possibility to keep on charging users even if they are off-line (e.g. due to a hardware problem), providing an incentive to not erase their backup when it is even more necessary. This last point is crucial for any P2P backup service.

[3] `http://www.wuala.com`
[4] `http://www.fiabee.com`

The paper is structured as follows: after studying the related works in section 2, section 3 introduces an overview of the proposed P2P backup system and defines the incentive mechanism that governs it. Later, section 4 presents a detailed discussion about the design of our proposal, emphasizing on the behavior of the system when a user is on-line or off-line. The different incentive mechanism is evaluated by means of simulation in section 5. In section 6, we analyze the possible threats that could affect our proposal and some security mechanisms to prevent them. Finally, we summarize the main conclusions of this paper in section 7.

2. Related Work

The idea of taking advantage of unused space in remote hard disks that are part of a P2P network was first studied at the beginning of the past decade. One of the earliest proposals was OceanStore [8] by Kubiatowicz et al., which provided solutions to many of the issues caused by relying on an untrusted infrastructure for data storage. Also Farsite [9] by Adya et al. was focused on the problems concerning the fault-tolerance and reliability of the stored data. However, these initial proposals considered a distributed file system in a heterogeneous, yet cooperative and trusted scenario. The first proposal of a backup-oriented solution was done by Batten et.al. in [10]. It features file encryption, version control, and provides reliability in case of multiple nodes failures. In this first stage, the research was more focused on how to achieve scalability, reliability and fault-tolerance in such a system [11, 12].

More recently, researchers have started considering additional aspects of P2P storage and backup systems. In such a distributed environments, peers are service users and providers at the same time: they want to store their data in the system, but to do so, they should also share part of their unused capacity with other peers. This peculiarity raises obvious fairness issues, especially when peers can behave selfishly. The so-called *free-riding* problem of P2P systems is well-known since Adar and Huberman showed in [13] that the 70% of peers were not sharing any files in the Gnutella network. Since then, a plethora of works [14, 15, 16, 17] have been proposed to try and mitigate this problem in P2P networks. They usually exploit some physical constraint of the system or they are based on fundamental principles taken from economics or game theory, modeling the problem as the "Tragedy of the commons" dilemma. In fact, most of the P2P systems currently deployed implement some kind of incentive mechanism like BitTorrent's "Tit-For-Tat" [18].

However, the *free-riding* problem in a P2P backup (or file storage) system is quite different from the one that can be found in a file-sharing system. A first difference is the potential "audience" of the resources being stored by the peers. For example, let us consider one file being uploaded with some file-sharing software. This file is "public" and could be potentially downloaded by any user of the network. In fact, in most P2P file-sharing systems, peers store a file because they are actually interested in it. On the other hand, in long-term storage services and especially in P2P backup services, each piece of information is usually encrypted and, thus, belongs to a single user (or to a restricted group of users). This consideration makes the problem even more complicated since there is not any implicit incentive for peers to store (useless) file chunks from other peers. Thus, how to reward a peer that is sharing an amount of

disk space much greater than the one it is asking for its data? Or the opposite case, how to incentive a peer to be more generous with the system, if it is using more space than the one is sharing? Furthermore, how can such storage quotas be enforced in a fully distributed system? A possible solution was proposed by Cox et.al. in [19, 20], using a framework for limiting the amount of data that a user is allowed to store into the network. This leads to a symmetrical behavior that, although keeps the system in a stable state, limits its flexibility. More recently, other studies have tried to tackle this problem without forcing the users to share a fixed amount of disk space. In [7], Seuken et al. have proposed to solve the problem by introducing a virtual market where a central system computes the exact amount of resources that users have to share (including uplink and downlink bandwidth) by following a trade mechanism that, under certain conditions, leads the system towards an equilibrium. However, this proposal still implies that peers follow a fair share, and it does not define any verification or penalty mechanism for selfish peers. Using monetary incentives in P2P systems was also studied in [1, 2, 3, 4, 5, 6], but these works are mostly focused on the security aspects about coining digital currency.

Furthermore, the incentive mechanisms for P2P file systems cannot be directly applied to P2P backup systems due to their specific characteristics. Whereas in a standard file system the saved data is frequently read and written, this is usually not true in a backup system. Instead, the data are commonly stored only once. Read operations are not frequent at all and, hopefully, null unless the user's data is lost from the local storage. Moreover, whereas in a distributed file-system the most important feature is access performance, the long-term durability of stored data is paramount in a backup service. The terms data durability and availability are often used in the literature regarding P2P storage systems [21]. The durability is the property that guarantees the fact that data stored in a peer will last for a time ideally infinite. This property is valid even when the peer is off-line. The availability property is more restrictive: it is valid when the data stored in a peer is correctly saved and available for downloading. Therefore, if we consider that the users of a P2P network behave selfishly, we cannot just rely on replication mechanisms and consider that the data stored in a working peer is "safe" without a secure control mechanism that continuously verifies this. If both peers are on-line, the problem could easily be solved by performing periodical checks of the data availability as proposed by Toka et al. in [22, 23]. Michiardi et al. in [24] presented an analytical model based on game theory for the detection of selfish peer and a similar solution was provided previously by Pamies-Juarez et al. in [25]. They propose a proactive monitoring system that checks the availability of the peers and assigns different quotas of the system to the users according to their obtained score. This architecture is further researched by the same authors in [26] where they found a relation between the system health (in terms of data availability) and the peer selection algorithm. Selecting the best peers increases the efficiency of the P2P system and provides better results when trying to retrieve the stored chunks. The solution proposed by Oualha et al. in [27] also identifies the data reliability as one of the biggest issues in the field of P2P storage. They propose a distributed system to find out malicious peers, either passive (which do not allow other peers to use their resources) or active ones (which deliberately free their resources after having stored a chunk). However, their proposal is focused on how to recognize whether a peer misbehaves and not on how to reward peers for guaranteeing

the maximum durability of the information.

However, even by periodically checking the presence of stored data, the system will not be in a truly safe state without a proper incentive mechanism. Peers are still storing data that is not of their interest. Hence, even non-malicious peers can potentially decide to remove it at any moment in order to free their local storage resources. This problem is even worse when users go off-line (e.g. vacation time periods or due to hardware failures) and cannot perform the verification or payment procedures associated to their remote stored data since there is no guarantee that they will come back on-line again to pay for their consumed resources. Therefore, peers may decide to remove the data of users that have been off-line for a while, which is exactly when backup data may be more important, since the user may desperately need it when coming back on-line (e.g. the laptop was stolen during the vacations or the hard disk crashed the night before).

Therefore the main objectives of our P2P backup proposal are twofold: 1) to design a secure verification mechanism that ensures that the backup information is safely stored, and 2) to devise an incentive mechanism based on micro-payments that motivates peers to store backup information even when the user remains off-line for an extended period of time (e.g. 2 months or more).

3. Overview of the proposed P2P backup system

This section briefly summarizes the proposed P2P backup system, including the secure data verification mechanism and the incentive system based on micro-payments. It also explains the desired behaviors that are encouraged by such incentive mechanism. All the technical details of the proposed mechanisms are defined in the following section. But first, let us provide a general overview of the proposed P2P backup system (see Figure 1).

Let us consider a *user* that wants to back up the information that she is working with. However, instead of employing the local hard disk, which has some free space but may suffer some hardware problem that wipes out both, the original information and its backup, this user prefers to store the backup in a P2P system where the information is replicated into multiple *peers* distributed all over the world. In exchange, other users are allowed to store their information in the user's hard disk, although the quota dedicated to the P2P backup application can be reduced if additional space for personal files is needed.

Thus, the user's P2P application makes a complete backup of the local information and keeps updating it with incremental changes whenever the user modifies local files. Since handling such large backup file is quite inconvenient, and it may not fit in the space shared by a remote peer, backup files are split in smaller pieces (e.g. 1 MB long), called *chunks*, which also simplifies updating the backup when only part of the data changes. These chunks are encrypted to guarantee the confidentiality and privacy of the backup data, and then uploaded to multiple peers of the P2P backup system. For resiliency reasons, each chunk is replicated and stored in different peers, and the (small) index file that specifies in which peers the chunks have been stored is placed at a known and safe place. How these peers are found or how many replicas are necessary to guarantee a given resilience level is out of the scope of this paper, since these topics have been already investigated in previous P2P storage works [28].

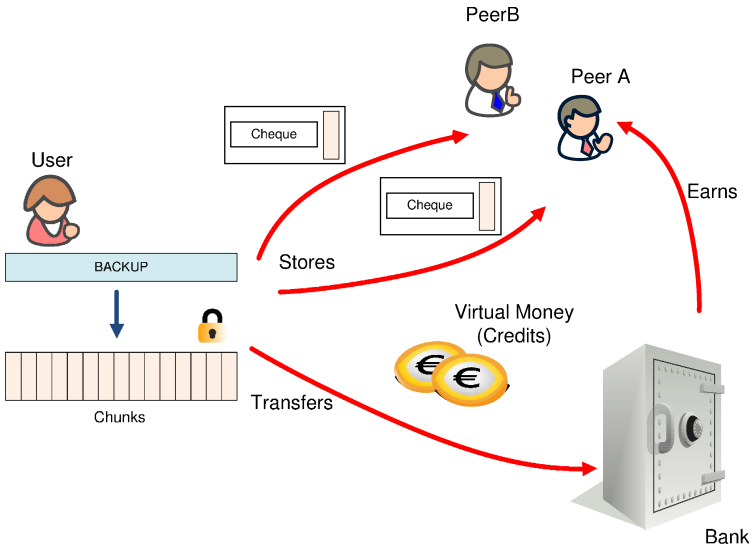


Figure 1: Overview of the proposed P2P backup system

Another implementation issue is whether the P2P application stores a complete copy of the backup locally or it may just depend on the remote copies stored in the P2P network. The only requirement is that any given chunk can be regenerated locally, either from a local backup copy or from the user's working file set.

In order to minimize the bandwidth employed by the P2P application, once the backup chunks are stored, they are never changed unless the user's information is updated. Thus, only the affected chunks have to be uploaded again. Furthermore, if some of the peers are not reliable enough (i.e. spend too much time off-line, have corrupted or lost chunks), the chunks that they store are moved to other, more reliable, peers. Additionally, in order to guarantee that the backup information is safely stored, the user's application periodically checks the chunks by means of a secure verification mechanism that does not require downloading the chunk itself. Instead, the user just sends a hash-based challenge to the peer, so it can only reply if it has really stored the chunk and has not modified it, either intentionally or due to some hardware or software glitch. Therefore, this mechanism is able to identify unreliable peers, and usual offenders may be placed in a local black list in order to avoid trusting them in the future. More complex data verification architectures can also be applied. A comprehensive work about "data possession" was presented by Ateniese et al. in [29].

However, we consider that this verification mechanism alone is not enough to ensure the correct behavior of the system in a real world scenario, with selfish peers that may delete chunks at any time, or somehow waste the resources of the system (e.g. storing too many copies of the backup or trying to verify their chunks too often). For this purpose some incentive mechanism should be employed to reward well-behaved peers. In this case, we propose an incentive mechanism based on micro-payments, where peers must pay for the resources of the system (i.e. bandwidth and storage space) that

they are consuming. That means that users must not only pay the peers for storing their chunks, but also for any operation that might be requested, for instance the verification of data. Therefore, peers will get more or less revenue depending on the space they are willing to share. Moreover, since this revenue can be employed to pay for storing your own chunks, this mechanism naturally couples the resources consumed by a user with the resources shared with other peers (which may be completely different to the ones where the user's chunks are stored), without trying to impose artificial altruism or symmetric behaviors. Furthermore, paying for every requested operation will lead to more efficient applications that avoid superfluous operations. In addition, peers are encouraged to stay on-line in order to be seen as reliable ones by other users, and thus to get the fees related with the different operations that are necessary to store chunks (i.e. upload chunks, verify and update them, etc.).

There are many micro-payment proposals in the literature that could be employed to perform the payments required by this incentive mechanism. These systems are characterized by the trade-off between its security and the overhead required to generate and verify the virtual currency that is used in the transactions. For simplicity and performance reasons we have chosen a centralized micro-payment mechanism based on a trusted third party, simply called *bank* in this paper, since it resembles a real-world bank. Actually, there could be multiple banks that perform virtual money transactions among each other, but for the sake of simplicity, we will describe the system considering a single bank. All users/peers will have an account in the bank that records the virtual money owned by each peer. Peers may start with a predefined amount of virtual money, and the bank may also accept real money in exchange of virtual one. This allows users to participate in the P2P system, saving their backup without sharing their own resources. The payment of some amount of virtual money from one peer to another is then performed by asking the bank to transfer money between the two peers' accounts. Since too many payments could stress the bank's infrastructure, payments are delayed and performed as a batch. Our proposed incentive mechanism also enforces this behavior, since users must pay the bank for each transfer operation that they request. Moreover, the bank could employ the collected fees to fund its infrastructure, either directly, if the virtual money can be exchanged for real one, or indirectly, if the bank employs its virtual money to gain storage space that can be later rented to other parties.

However, both the verification and incentive mechanisms have a common problem: the user must be on-line to perform the verification and payment procedures. This requirement could be seen just as slightly inconvenient, but in the case of a P2P backup system it could be fatal. If users suffer a hardware failure (e.g. hard disk crash) and the recovery takes some time, when they come back on-line their, now vital, backup may have been gone, deleted by unpaid peers. Therefore, the basic incentive and verification mechanisms have been extended to support users staying off-line for extended periods of time. Off-line payments are enabled by means of a *digital cheque*, issued by the user, which allows the entitled peer to request to the bank the payment for the user's chunks being stored. But how is the bank able to verify that the peer is really storing those chunks if it does not have a copy of the user's data? The proposed solution is that the user provides to the bank a list of challenges and their valid responses, called *nonce list*, which could be later employed to verify the chunks pertaining to the digital cheque

being cashed by a peer. To avoid overloading the bank with the nonce lists of all users, the nonce list is stored alongside the chunk in the peer itself, but encrypted so only the bank can read it. Therefore, the bank only has to retrieve and cache the nonce lists of the users being off-line too much time due to the peers are able by themselves to cash their cheques. The fact that the off-line verification mechanism is much more expensive for the user than the on-line one, it incentives users to remain on-line if possible.

The following sections fully specify the P2P backup system operations, including the proposed verification and incentive mechanisms, in both cases, when the user is on-line or off-line.

4. P2P backup system design

Before going into the technical details of our proposal, it is necessary to define the different roles that are necessary in this architecture:

User: We employ the term "user" to refer to a peer that is storing its backup information in other peers. Because of performance and resiliency issues, user's information is split into several *chunks* that are later encrypted, replicated and stored in different peers. There should be redundant copies of the same data to cope with peers leaving the system or remaining too much time off-line. In that case, the owner of the information has to create another chunk replica and store it in a different peer to replace the lost one.

Peer: It is a node that cooperates sharing some of its resources to compose the proposed P2P backup system. An incentive mechanism based on micro-payments is employed to ensure that bandwidth and storage resources are not wasted, and to make sure that well-behaved peers are properly rewarded. In particular, users must pay peers for storing their chunks and performing the requested operations. The earned money may be then employed by peers to store their own chunks, or even to exchange virtual credits for real money.

Bank: The proposed micro-payment mechanism requires a trusted entity, which performs the role of a real-world bank, but handling virtual currency, called *credits* in this paper, instead of real-world money. The bank[5] accounts the virtual money owned by all users/peers of the system, thus payments between peers are performed just by asking the bank to transfer virtual money from the payer's account to the payee's.

4.1. P2P Backup System operations

All transactions related to the P2P backup service are charged with a fixed amount of virtual money. Therefore, it is necessary to pay at least 1 credit for each transaction, charged to the initiator of the operation. Our long-term storage service employs five basic operations:

[5]This architecture supports any number of banks that trust each other, as in the real world. However for simplicity reasons we constrain our description to a single bank.

PUT: It uploads and stores a chunk of data in a remote peer. The cost of this operation must be much higher (e.g. 10 times) than the cost of renewing a chunk, which does not require uploading it, as in this case.

VERIFY: It checks that a remote peer is truly storing the desired chunk of information, for instance before being renewed. The cost of this operation is 1 credit, so users are encouraged to not check continuously their chunks.

UPDATE: It renews a previously uploaded chunk for a certain period of time (i.e. 1 day) without uploading it again. The cost of this operation is 1 credit plus D credits per day (or any other arbitrary time unit) elapsed since the last UPDATE operation.

GET: It downloads a chunk of data from a peer. The cost of this operation should be similar to the PUT one since it also requires moving data between peers.

DELETE: It removes a chunk from a peer. Its cost is 1 credit and it is necessary to avoid being blacklisted by a peer storing an old chunk. That is, when a user moves one or more chunks from one peer to another (e.g. because the new one is more reliable), it should delete the chunks from the old one.

The nominal value of the fee associated to each operation could be calculated using the relative costs of performing these operations, however, for simplicity in this paper we will assume that there is a common, fixed price to store (e.g. 100 credits) and update (e.g. D=10 credits/day) one chunk.

It is important to define how the transactions of virtual money are performed after each operation. There are several ways to perform micro-payments. In our system, we use delayed bank transfers, where a user delays all payments to perform them together in a specific moment of time (e.g. once per day after updating all the chunks). We think that this solution is a good trade-off between computational load and complexity with respect to other solutions (e.g. instant bank transfers or direct micro-payment exchanges between peers [1, 2, 3, 4]). The proposed solution of delayed bank transfers reduces the consumption of bandwidth and computational resources to process the transactions of all peers, although the bank could still become a bottleneck. However, this architecture can support multiple bank entities, and there are secure mechanisms [2] that even allow each bank to define bank assistants to help them in their duties if necessary. Furthermore, the bank also charges each money transfer batch with a small fee (e.g. 5 credits) in order to fund its operations, as well as to reduce its load, because the peers will try to contact the bank as less as possible.

However, delayed bank transfers introduce a time gap between the peer performing an operation and when it receives the payment from the user. Therefore, a peer should allow some debt from its users until the payment comes. For this purpose, each peer also maintains local *debt accounts* per user, where the debt accumulated by each user is annotated. That is, when a peer performs some operation for a user, like updating a chunk, the peer adds its cost to the user's debt account. Conversely, the user employs its own peer's debt account to known how much money it owes to each peer. Then, when the peer receives some payment from the user, the peer subtracts it from its own

local debt account, which ideally should go back to zero. Otherwise, if the debt of a user grows above certain threshold, the user is blacklisted and all its chunks are erased. This debt threshold should be relatively low in order to prevent free riders, and thus this mechanism alone should only tolerate peers being off-line during short periods, let say a weekend.

4.2. On-line system behavior

In this subsection, we define how our proposal of long-term peer-to-peer backup works when both the user and its peers are on-line. This subsection presents the basic design of our proposal to later understand how it supports off-line users, which is the main objective of this work. The system is governed by the incentive of obtaining a certain amount of virtual money per transaction. The exchange of chunks and the transfer of virtual money among users, peers and banks is based on a simple client-server protocol. Each user operation is acknowledged or responded by the remote peer. Figure 2 presents an example of the signaling among peers, when all of them are on-line, to explain precisely the proposed design. This example includes the initial storage, verification and update of user's information at two remote peers, and exemplifies the mechanism of virtual money transfers associated with each operation.

Let us assume that the user wants to store one chunk of encrypted information but, for resiliency purposes, places it in two peers: peer A and peer B. To do so, the user performs one *PUT* operation per peer (messages 1 and 3), which includes the full chunk data (*ChunkData*), as well as a timestamped *Cheque* issued by the user to each particular peer, and a list of nonces (*NonceList*) employed for off-line verification purposes (this mechanism will be fully detailed in subsection 4.3). Each *PUT* operation is confirmed with a *PUT ACK* acknowledgment (messages 2 and 4). Each chunk in the system is uniquely identified by the globally-unique user identifier (*UserID*) plus an individual chunk identifier (*ChunkID*), which can be independently assigned by each user. A *PUT* transaction is only employed once per chunk and peer, unless the information inside the chunk is modified; thus, it has to be uploaded again.

When the user is on-line, chunks should be updated periodically, and thus pay peers for storing them. However, before performing any payment to a peer, the user should check whether the peer is actually storing those chunks by means of a *VERIFY REQUEST* operation (messages 5 and 9). To avoid downloading the chunks for local verification, the user sends a *nonce* (i.e. a random number) as a challenge to the peer. The remote peer should then apply a predefined hash operation over the chunk data concatenated with the nonce provided by the user. The result of this operation is sent back to the user in a *VERIFY RESPONSE* operation (messages 6 and 10). The user could then check the challenge response using its own copy of the chunk and the nonce. Therefore, the peer can obtain the proper solution to the user's challenge only if it is actually storing the chunk. Users should include peers with several unsuccessful *VERIFY RESPONSE* operations in their local black lists, since they are not reliable peers to store information.

If the verification of a chunk of data succeeds, an *UPDATE* operation is performed afterwards. The *UPDATE* operation renews the cheque of the peer and optionally the chunk's nonce list (messages 7 and 11), although in most cases the latter is not necessary because the nonce list can be reused until it is exhausted because of off-line

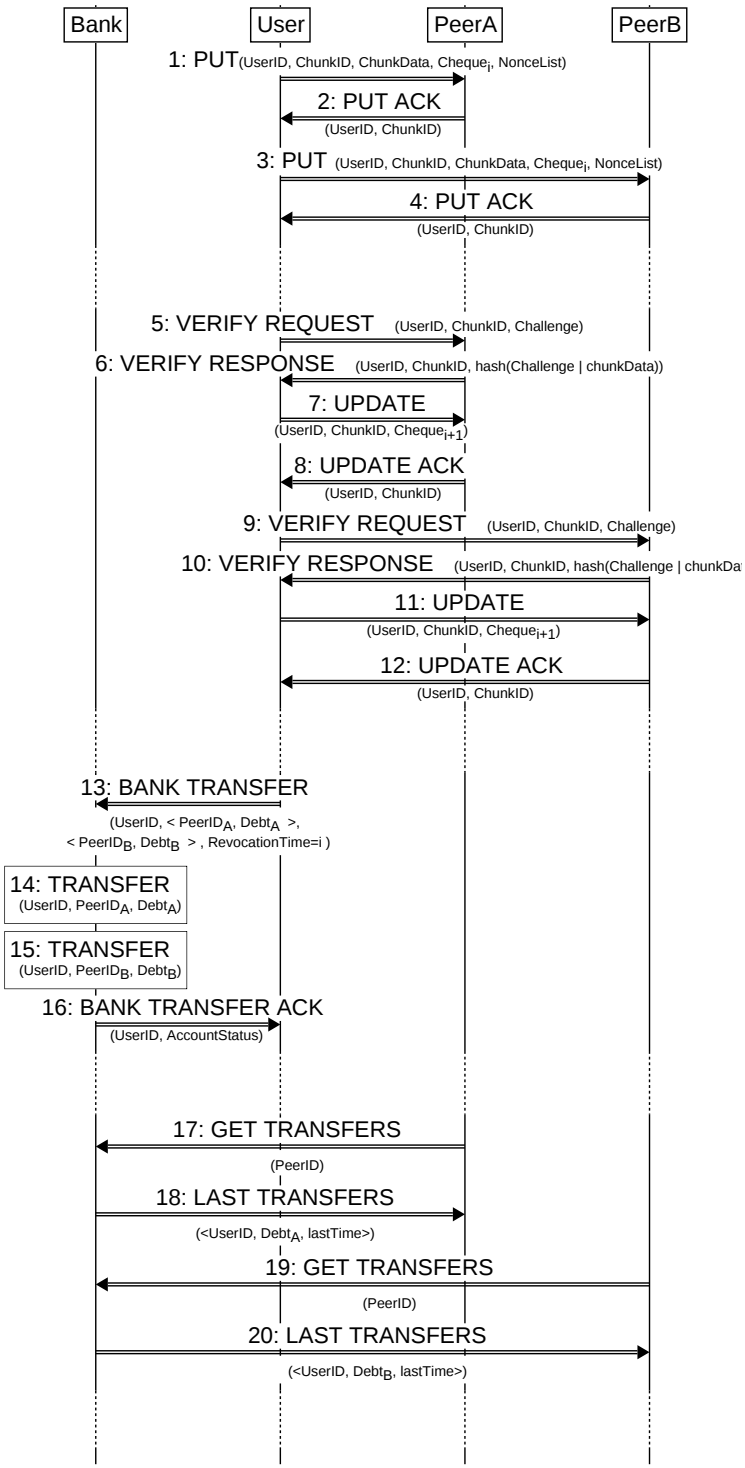


Figure 2: Example scenario of long-term P2P Backup with one on-line user and two peers.

verifications. The peer confirms the *UPDATE* operation with an *UPDATE ACK* response (messages 8 and 12)[6].

Therefore, the total cost of updating the chunks stored at a peer for an on-line user is: 1 credit for the (optional) *VERIFY* operation, plus 1 credit for the *UPDATE* operation plus D, the cost of storing a chunk per day, multiplied by C the number of chunks and t, the number of elapsed days since the last update time. That is:

$$U_{on}(C, t) = 2 + D \cdot C \cdot t \tag{1}$$

However, until this moment, no payment has been performed yet because, thanks to the long-term relationship between a user and the peers, the system could employ delayed bank transfers to reduce the load. Meanwhile, all peers locally maintain the accumulated debt with other peers. That is, in the above example the user will subtract U_{on} credits from the local peer's debt account, whereas the peer will add U_{on} credits to the user's debt account. Finally, after the user has performed all the daily update operations, a single payment transaction is performed for all the accumulated debt with its peers. This is only necessary for positive debt balances, thus if two peers store the same number of chunks into each other and update them daily, the mutual debts cancel each other out and no payment is necessary. Otherwise, these peer payments are requested to the Bank through a *BANK TRANSFER* operation (message 13), which specifies the accumulated amount of virtual money that should be transferred from the user's bank account to each peer. The user also employs this operation to revoke all the cheques created before the specified date. The bank only needs to remember the last revocation date of each user. This way peers with old, not renewed, cheques cannot get cash from them. The bank also charges a small fee (e.g. 5 credits) to the user to cover the cost of this transaction. After processing the money transfers (operations 14 and 15), the bank sends back a *BANK TRANSFER ACK* response (message 16) confirming the payments and detailing the balance of the bank account to the user.

Peers are not instantly notified of these bank transfers, instead peers should periodically request, via *GET TRANSFERS* requests (messages 17 and 19), the balance of their own accounts and the last received transfers. The bank replies with a *LAST TRANSFERS* response (messages 18 and 20) containing the accumulated virtual money obtained from each user, identified by the *UserID*, since the specified time. This money is then subtracted from the local user's debt account, thus ideally it should be zero after all operations and payments have been performed. Therefore, the peer could check whether the users are actually paying for the stored chunks, either with on-line or off-line paying mechanisms. If this is not the case, and the accumulated debt of a user exceeds a predefined threshold, all the user's chunks are removed, and the *UserID* is added to a local black list to avoid trusting that user in the future.

This mechanism enables the management of remote information if both peers and users are on-line. However, the big challenge is how to incentive peers to store the information from users when they are off-line, and thus they do not update/pay their

[6]For simplicity purposes, the VERIFY and UPDATE operations shown in Figure 2 are independent and refer to a single chunk. However, a real implementation could optimize this and perform both operations with a single message exchange for all the chunks of the user stored in the peer.

chunks. This fundamental issue is addressed in the next section.

4.3. Off-line system behavior

Off-line system behavior is crucial for long-term storage services like P2P backup, since it must guarantee the storage of information even when users go off-line for extended periods of time. Thus, the aim of this design is to provide a mechanism that assures peers storing information from off-line users to keep earning virtual money for their service. Our proposal defines a secure *digital cheque* that enables these off-line transactions with the help of a trusted third party, in this case the user's bank.

By using the cheque issued by the user, a peer can keep earning virtual money for storing chunks of off-line users. In this section, we detail the off-line behavior of our proposal. Later, in section 6, we detail how this cheque is secured to avoid selfish and misbehaving nodes.

A cheque is composed by the following fields[7]:

Bank-ID: This field identifies the bank of the user that issues this cheque. A peer should reject the cheques from a bank that it does not trust.

User-ID: It is the identifier of the user that generated this cheque.

Peer-ID: It is the identifier of the peer receiving the cheque for storing one or more chunks from the user.

Chunk-IDs List: This array field contains the identifiers of all the chunks from the user stored by the peer.

Nonce Lists Hashes: Each chunk has an associated *nonce list* for verification purposes. This array contains the hashes of the nonce list of each chunk that appears in the above *Chunk-IDs List* field.

Nonce Lists Key: This field contains the symmetric key used to encrypt the chunks' nonce lists. The key itself is encrypted in such a way that only the bank has access to it. Further details can be found in section 6.

Creation date: It specifies the date when the cheque was created. The bank compares the last revocation date provided by the user with this creation date to check whether the cheque is still valid, or it has already been revoked.

Validity date: This field defines the first day (i.e. 7 days after the creation date) when the cheque can be employed to withdraw money from the user's bank account, although it is off-line.

[7] Some of the cheque's fields have fixed, well-known values, and thus it is actually not necessary to include them in the cheque exchanged by the P2P backup protocol. However for completeness, and in order to better resemble a real-world cheque, we explicitly list them all.

Face value: It defines the quantity of virtual money from the user that the peer can obtain per chunk and day. This amount should include the cost of storing one chunk during one day, plus the extra cost of cashing the cheque at the bank. This is an incentive to not overload the bank with cheques since peers can obtain higher net incomes (they can keep the fees for themselves) if they choose to reduce the frequency with which they cash the cheques.

User's signature: The cheque must be digitally signed by the user in order to ensure its legitimacy, as well as to protect its contents. Notice that users independently issue cheques, without involving the bank.

With the previous definitions, we can now explain an example of an off-line scenario that allows peers that are storing chunks from off-line users to keep earning virtual money. This example scenario is shown in Figure 3.

When a user goes off-line, it is necessary to keep updating the chunks of information that are already stored in remote peers. We make use of the digital cheques to perform this process. When the *Validity date* specified in a cheque arrives and the chunks have not been updated yet, peers can request a *CASH CHEQUE* operation to the user's bank for the amount specified by the cheque's face value (messages 1 and 10). As in the on-line case, no payment is performed until it has been verified that the peer is actually storing the claimed chunks. However, the bank does not have access to the user's chunks to generate and validate a challenge, as it is done in the on-line case. Instead of this, the bank employs the nonce list previously stored at the peers by the user. The nonce list associated with a chunk is generated by the user and contains a number of challenges (e.g. 60 nonces) and their associated responses. This list is encrypted so only the bank can access to the different challenge responses.

The bank asks for this list to the peer trying to cash a cheque with a *NONCE LIST REQUEST* and its corresponding *NONCE LIST RESPONSE* (messages 2 and 3 respectively) and then uses one of the nonces [8] to challenge the peer on behalf of the user with a *VERIFY REQUEST* (messages 4 and 11), which should trigger a *VERIFY RESPONSE* (messages 5 and 12). If the operation is successful, an *UPDATE* operation is issued to the peer, and the bank transfers the required virtual money from the user's account to the peer's one. This quantity is C (the number of chunks that have been verified), t (the time elapsed since last update operation) times the cheque's face value. The fee charged by the bank is also added to this quantity.

Since a chunk could be replicated in several peers, the bank could store the nonce lists in a cache so it does not need to download it again from the replica peers. This fact explains why after the *CASH CHEQUE* operation in message 10 the nonce list is not requested again to peer B, since it was previously obtained from peer A (message 2).

When the user comes back on-line, it will ask the bank for the payment operations performed during its off-line period (messages 17 and 18). With this information the

[8] The bank could just choose the nonce whose index equals to the number of days elapsed since the cheque's validity date. This ensures that the nonce has not been employed yet with that peer, without requiring the bank to remember which nonces have been consumed already.

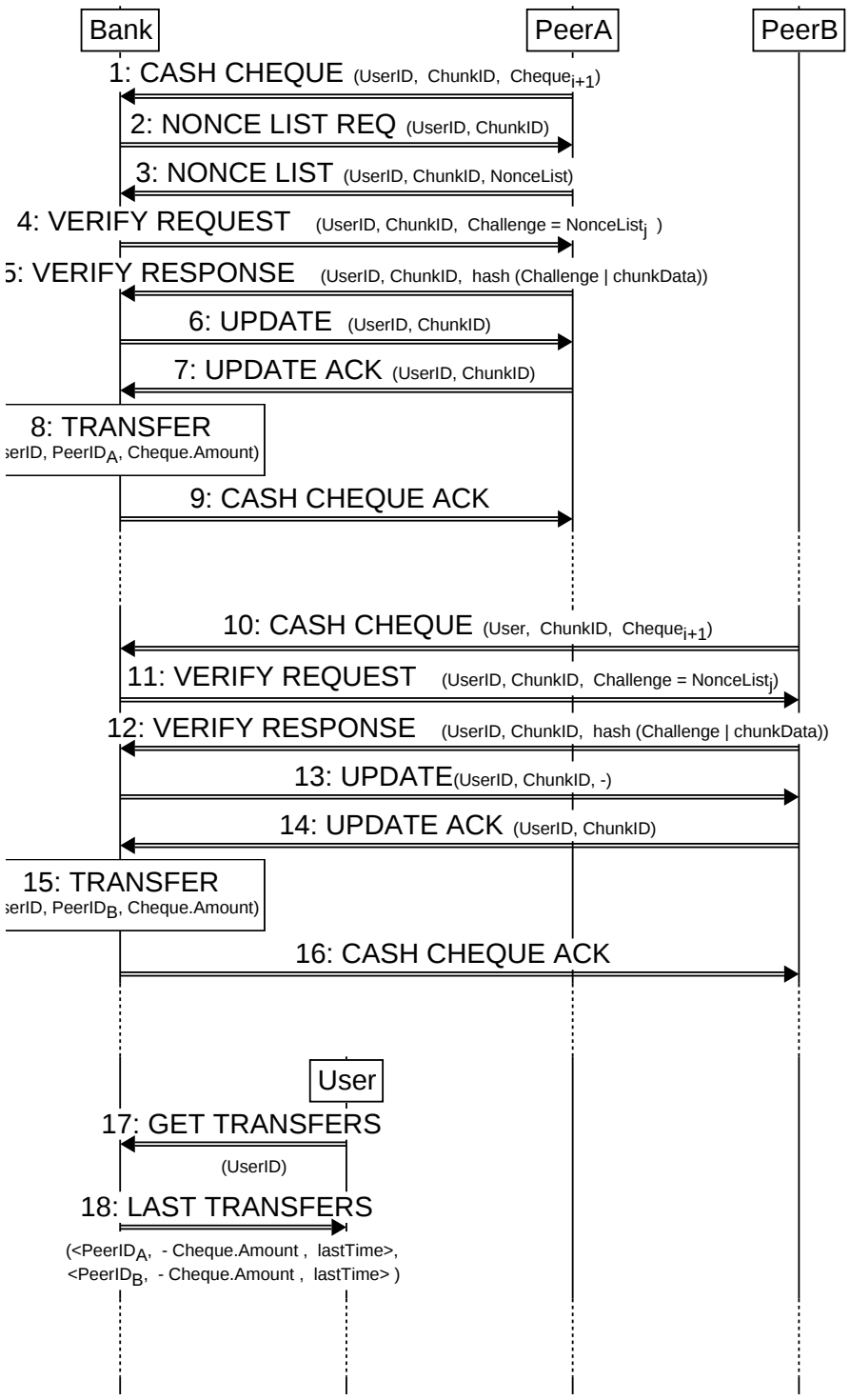


Figure 3: Example scenario of long-term P2P Backup with an off-line user

user knows the last update time of its peers, and keep updating the chunks in on-line mode (after paying for the days since the last update operation performed by the bank). The user should also renew the cheques, revoke the old ones and create new nonce lists to avoid repeating the same challenges to the peers.

Since the peer's *CASH CHEQUE* operation requires the bank to perform additional operations, this transaction must have an extra cost X to cover the expenses of the (optional) *NONCE LIST REQUEST*, *VERIFY* and *UPDATE* operations issued by the bank in name of the user. This extra cost is later compensated by the cheque's face value (Y). Therefore the final benefit for the peer is:

$$U_{off}(t) = 3 + Y \cdot C \cdot t - (1 + X) = Y \cdot C \cdot t - X + 2 \tag{2}$$

It is important to carefully set the cheque's face value for the off-line incentive mechanism to work. In particular two conditions should hold:

1. Peers should get more virtual money per day and chunk than in the on-line case.
2. Peers should cash their cheques as less as possible in order to not overload the bank. Therefore they should obtain more money performing a single *CASH CHEQUE* operation after n days than doing it every day.

Translating these restrictions into equations:

$$U_{off}(C,1) > U_{on}(C,1) \implies Y - X + 2 > 2 + D \implies Y > D + X \tag{3}$$

$$\begin{aligned} U_{off}(C,n) > n \cdot U_{off}(C,1) \quad \forall n > 0 &\implies Y \cdot C \cdot n - X + 2 > n \cdot (Y \cdot C - X + 2) \\ &\implies X \cdot (n-1) > 2 \cdot (n-1) \implies X > 2 \end{aligned} \tag{4}$$

And thus, the cost of *CASH CHEQUE* operation should be $X > 2$ (i.e. 3) and the Cheque face value must be $Y > D + X$ (i.e. 15). Thus, this mechanism gives the appropriate incentives to peers to keep storing information from off-line users. The maximum duration of the off-line period for a user is only limited by two elements: the virtual money left at the user's bank account and the length of the nonce list. Moreover, due to the bank fees, it is more costly for the users to update their chunks in off-line mode than in on-line mode, thus users are also encouraged to be on-line. Furthermore, peers have also to pay an upfront fee to the bank for cashing cheques. Therefore, by increasing the interval between consecutive cashes of the same cheque, they can obtain higher revenues, meaning that peers will only contact the bank sporadically to cash their cheques. This also helps to improve the scalability of the off-line incentive mechanism.

5. Evaluation of the Incentive Mechanism

In order to evaluate the different aspects of the incentive mechanism and how they complement each other, we have implemented the proposed P2P backup system in a custom-made simulator developed in Java. The simulator is based in cycles, on each

Table 1: Simulation parameters

The values $^{(a)}$, $^{(b)}$ and $^{(c)}$ are D, Y and X respectively, as defined in equations 2, 3, 4

Simulation time	365 *cycles*
Total number of peers	10,000 *peers*
Backup size	10 GB
Chunk size	1 MB
Number of replicas	3 *replicas*
Chunks stored per peer	1,000 *chunks*
Initial free storage space	75 GB
Initial bank balance	200,000 *credits*
Cheque validity date	7 *days*
Cost of PUT operation	100 $credits/chunk$
Cost of VERIFY operation	1 $credit/chunk$
Cost of UPDATE operation	$1 + 10^{(a)}$ $credits/chunk \cdot day$
Cost of DELETE operation	1 $credit/chunk$
Cheque face value	$15^{(b)}$ $credits/chunk \cdot day$
Bank fee of CASH CHEQUE operation	$5^{(c)}$ *credits*

cycle (i.e. one day) all the on-line users/peers put/update their chunks in other on-line peers and, in order to improve the availability and durability of their backups, move all the chunks from the peer with the lowest measured availability[9] to a new, randomly selected peer. For simplicity, all peers/users share the same storage space (75 GB) and have exactly the same backup size: a 10 GB backup split in 10,000 x 1 MB chunks, which are then replicated 3 times, leading to 30,000 chunks per user. Moreover, the number of chunks stored in a peer is set to 1,000, in order to prevent the failure of a single peer wiping out a significant portion of the backup, as well as to reduce the number of peers that have to be contacted each day (i.e. 30 peers). The full list of simulation parameters is shown in Table 1.

In order to study the effects of the different aspects of the incentive mechanisms, we have implemented different peer behavior classes (i.e. on-line/off-line periods, rate of update operations, etc.), instantiate an equal share of peers (from a total of 10,000 peers) featuring the desired behavior class, and simulate the P2P backup system with those mixed peer behaviors for 365 cycles (i.e. one year).

Let us start with the main objective of the P2P backup system: peers should stay on-line and offer their free storage space to other users. In order to verify this, the first simulation has four peer classes (with 2,500 peers each), all of them offering the same space, but on each cycle they randomly choose to stay on-line or off-line with a different probability (P_{online}). In the first class, the peers are always on-line (P_{online}=1.00),

[9] Peer availability is measured locally by each user by trying to contact each day with the peers storing the chunks, and checking whether are on-line or off-line. The availability ratio of new peers is only computed after 5 measurements (i.e. five cycles).

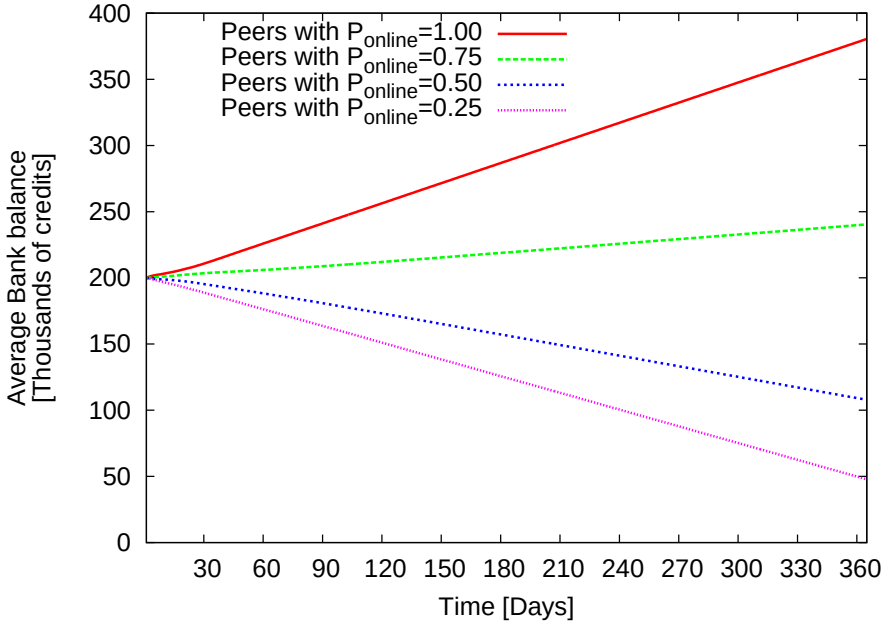
the peers of the second class are on-line the 75% percent of time (P_{online}=0.75), a 50% in the third class (P_{online}=0.50), and finally the peers of the last class are only on-line during the 25% of the simulation cycles (P_{online}=0.25).

Figure 4(a) shows the average bank balance of each class of peers. Since all peers have exactly the same backup size (10 GB), and thus roughly the same cost per day, the differences among classes come from the different revenues that peers obtain by being on-line and offering their free storage space. Clearly, the peers staying more time on-line outperform the ones that are off-line more often, since a longer time on-line means greater revenue. In fact the last two classes have deficit and may be expelled from the system (i.e. their backups will be deleted) shortly after one year, unless they change their behavior or pay the bank real money to get additional credits. On the other hand, the first two classes have surplus, meaning that they must not pay any additional money to support the operation costs of the bank, and actually they could store a larger backup, replicate it more times, share less space or even earn some real-world money.

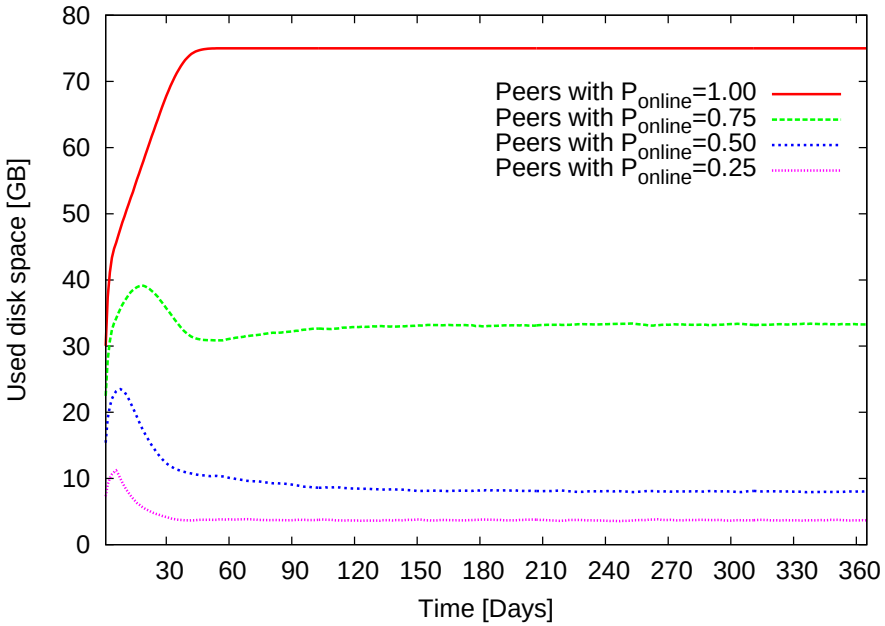
This aggregated behavior naturally evolves from the fact that users prefer on-line peers to off-line ones, thus each day on-line users try to move the chunks from their peer with the worst availability ratio to a new, randomly chosen peer that is both on-line and has free space left (on steady state this requires, on average, 5,500 DELETE + PUT operations per day). Thus, once a chunk is stored in an always on-line peer, it is never removed from it, whereas the chunks stored in a peer with an on-line ratio of 25% will be moved before the ones stored in the other classes of peers. This leads to the distribution of stored data shown in Figure 4(b). Class 1 peers (100%) have their disk completely full, whereas the remaining classes still have free space, due to the time they stay off-line. The small surplus of the 75% class comes from the fact that, on average, those peers store more than 30 GB, which is the size of the replicated backup for all peers, while the last two classes are below this break-even point and thus run on deficit.

Figure 5 shows the effect of the two additional incentives introduced into the system to lower the load. In particular Figure 5(a) shows the average bank balance of a simulation with two peer classes (5,000 peers each), which also have the same backup size and storage space, but they now both stay on-line 50% of time. However in this case on-line/off-line periods are not random, but deterministic. The peers of the first class are one day on-line and off-line the next one, while, in the latter class, peers are 8 days on-line, followed by other 8 off-line days. Therefore in this case the differences between them do not come from preferring peers with a greater availability as explained before, but from the fact that the users of the second class remain off-line longer than the cheque validity date (7 days) and thus their peers will cash their cheques. Therefore, the extra cost of cheques make class 2 peers to spend more money than class 1 ones, and thus providing them with an incentive to stay off-line less than 7 days to prevent other peers cashing their costly cheques. Notice that the balances of both classes do not compensate (i.e. do not sum up to 400,000), because the bank gets a small fee (5 credits) for each cheque transaction in order to pay its operational costs.

Similarly, in order to lower the total load of the system, users are also incentivized to minimize the number of operations they request to their peers. The strength of this incentive can be tuned by changing the fixed cost per message. Figure 5(b) shows the average bank balance of two peer classes that are both 100% on-line, but take

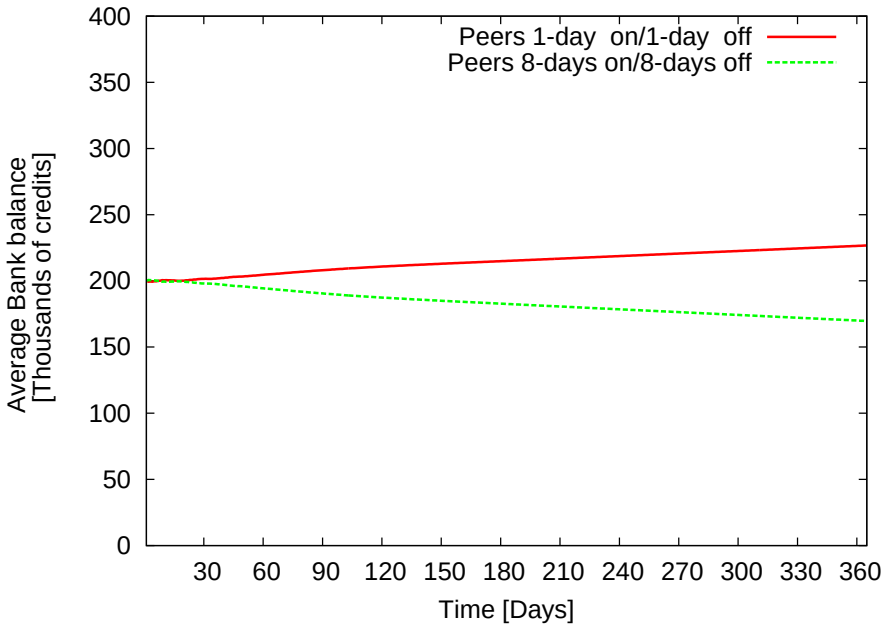


(a) Average bank balance of peers

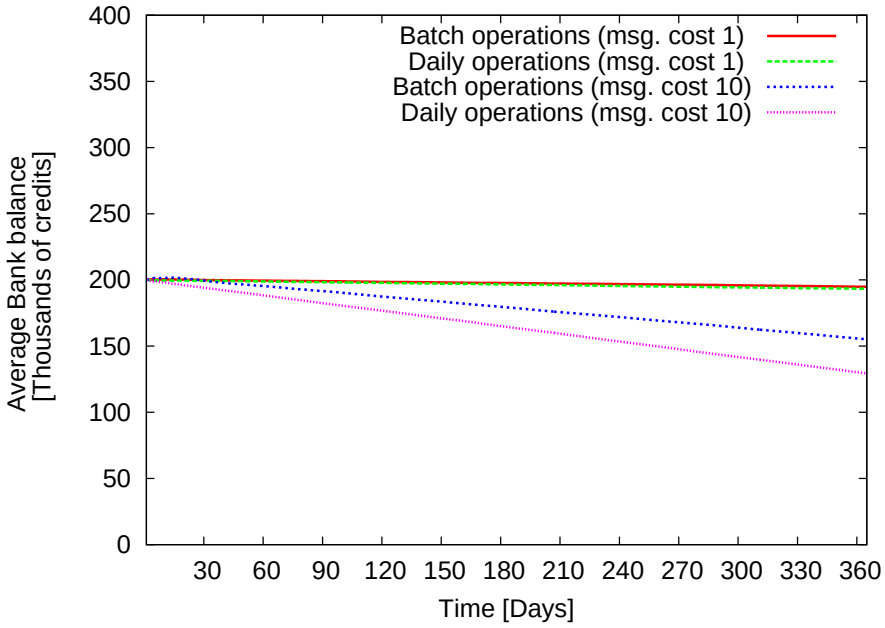


(b) Peers used disk space (75 GB Max.)

Figure 4: Evaluation of the incentive to stay on-line and share storage resources



(a) Minimizing cheques: Average bank balance of peers



(b) Minimizing operations: Average bank balance of peers

Figure 5: Evaluation of the incentive to minimize operations and cheques

long vacations (30 days) with a small probability (3%). The only difference between these two classes is that, in the first one all operations (i.e. requesting verify and update operations, making money transfers and cashing cheques) are performed daily, while in the second one operations are delayed and executed as a batch after 5 days, effectively reducing almost five times the total number of operations (e.g. 61,753 vs. 11,972 UPDATE messages on average per day). This simulation is run twice, first with a fixed cost of 1 credit per message, and then with a message cost of 10 credits. It can be clearly seen that the best peer strategy is to reduce the number of operations as much as possible. The differences between both peer classes broadens by increasing the cost per message, thus allowing the system designer to explicitly set the reward to users for reducing their load in the system.

Therefore, with the proposed incentive mechanism, the P2P backup system benefits well-behaved users and penalizes free-riders or malicious users, effectively enabling peers to store the backup of off-line users. However, the incentive mechanisms alone do not prevent the possibility of lost backup chunks due to some kind of physical failure in the peers storing them, and in this case an off-line user is not able to regenerate the lost chunks. Therefore users must rely on the chunk replication mechanisms to keep their backups alive while they are off-line during extended periods of time. If we assume that p is the probability of a peer failure, then the probability $P_{ok}(t, R, M)$ that at least one of the R chunk replicas survives during t days, for all $M = B/C$ peers storing the backup file (being B the total number of backup chunks and C the number of chunks stored per peer), is:

$$P_{ok}(t, R, M) = (1 - (1 - (1 - p)^t)^R)^M \tag{5}$$

This means that with 3 replicas, a peer reliability of a 99.9% (i.e. that means a probability of failure p equal to 0.001), and with

$$M = \frac{10GB/1MB\,chunks}{1000\,chunks/peer} = 10\,peers, \tag{6}$$

the probability that the full backup survives during 60 days with no user intervention is P_{ok}=0.99802, or P_{ok}=0.99974 for 30 off-line days, and, even in the improbable case of some backup replication loss, only a subset of the user's chunks will be affected.

6. Security of the Incentive and Verification Mechanisms

Any incentive mechanism, and specially the ones based on virtual money, should be secure, otherwise they become useless, or even worst, benefit mischievous users instead of well-behaving ones.

Probably the most obvious attack to the proposed mechanism is impersonation, where an attacker tries to convince the bank or other peers that it is a different user to request payments from the targeted bank account or to be able to store chunks with a different *UserID*. Therefore, first of all, it is necessary to avoid all kinds of impersonation or man-in-the-middle attacks in the system. Each entity participating in the system (i.e. banks and users/peers) should have a X.509 digital certificate [30] that links its *BankID*/*UserID* with a RSA public key [31], as it is shown in Figure 6. Therefore a user could easily assess its identity by means of a digital signature or by using a

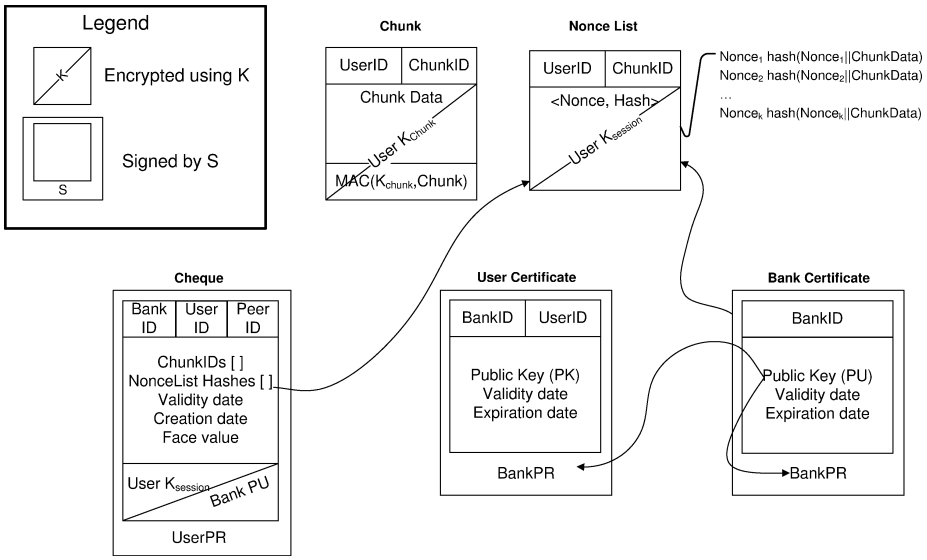


Figure 6: Security relationships between the different elements of the P2P backup system

secure protocol that exchanges certificates like TLS [32]. The bank of the user or other trusted third party could issue these user certificates. The banks acting as Certification Authorities (CA) has the additional benefit that there is a strong relation between users and their banks, and the peers could just reject transactions with users from unknown banks that are not in the peer's trusted CA repository.

Even if certificates do not allow an attacker to impersonate another user, an attacker can still generate multiple personalities (also known as Sybil attack) to thwart the black list mechanism. That is, when a *UserID* is blacklisted because of misbehavior or too much debt with other peers, the attacker could just ask for another *UserID* to the bank. This problem can be mitigated by hardening the process to create a new user, either requesting a real world identity, charging some real-world money (e.g. 10$ that are exchanged for virtual money), or a combination of both.

To guarantee the integrity and the privacy of the users' information, backup data should be encrypted before being stored at the peers, and it should also include a HMAC [33] digest code to avoid tampering. For performance reasons, chunks should be encrypted employing a symmetric cipher, albeit each user can choose its preferred encryption mechanism for its own chunks. Therefore, since the user is the only one who knows the encryption key, neither the peers storing the chunks nor the user's bank are able to access the backup information. The only additional precaution is that at least one copy of this secret key as well as the user's private RSA key must be securely stored (i.e. protected by a passphrase) in some well-known and safe location in order to recover the backup in case of any fatal local hardware or software failure.

Of course, the digital cheque for off-line transactions requires additional security measures. The cheque must be digitally signed with the private key of the user that

issues it, in order to allow that the peer and the bank could assess its authenticity using the user's public key. An agile revocation mechanism for cheques is proposed to minimize the state required at the bank: new cheques are issued to peers whenever chunks are updated, and then the user just notifies to the bank which is the creation date of the last set of cheques in each delayed transfer. Therefore, the bank only needs to check the creation date of a cheque to decide whether it has already been revoked or it is still valid.

The security of the remote chunk verification mechanism should also be analyzed. The usage of a cryptographic hash function prevents a malicious peer to solve the challenge without having the chunk. The malicious user could however still try to generate all the possible challenge nonces and then store their responses instead of the chunk itself. Therefore, in order to thwart this attack, the dictionary with all challenges' responses must be larger than the chunk itself. An arbitrary large number could be chosen, but this would lead to larger nonce lists, which should be retrieved and cached by the bank. Therefore, it seems better to choose an appropriate nonce length (N bits), based on the chunk size (S bits) and digest's length of the hash function (H bits). Then:

$$S \leq 2^N \cdot H \implies N \geq \log_2(S/H) \tag{7}$$

Which means that with $S = 1MB = 2^{23}$ bit-long chunks and MD5 hashes ($H = 128 = 2^7$ bits), a nonce of just N=23-7= 16 bits is necessary. If chunks of $S = 1MB$ are considered, we need N=26 bits to prevent a complete dictionary attack. Thus, a 32-bit nonce seems to be a good value to avoid partial dictionary attacks with high probability. This value leads to a nonce list with 20 bytes per entry, and thus two months worth of challenges (i.e. 60 nonces) could be stored in as little as 1200 bytes.

Finally, since nonce lists are initially stored by the peers themselves, it is necessary to protect the nonces and challenges' responses from them. To do so, the whole nonce list should be encrypted so only the bank can decrypt it. We could use the public key of the bank for this purpose, however public key cryptography is much more CPU intensive than symmetric ciphers. Therefore it is much better to first encrypt the nonce lists with a secret key (e.g. AES key of 128 bits), randomly generated by the user. The problem now is how to convey this secret key to the bank. In this case, it is now feasible to employ the public RSA key of the bank to encrypt this short key. The encrypted key can be then added to the cheque, because all the nonce lists of a peer can be encrypted with the same key. Furthermore, the cheque also contains the hash of the different nonce lists to prevent a malicious peer to send an old nonce list, with known challenges, to the bank for validation.

Therefore, when the bank receives a cheque, first, it has to verify that it features a valid signature from its user (thus it is both legit and has not been tampered). Secondly, it needs to verify the validity and creation dates of the cheque by comparing them, respectively, with the current date and the last revocation date specified by user, and finally decrypt the nonce list key in order to obtain the challenges stored in the nonce list, whose validity is checked using the hash carried by the cheque itself.

7. Conclusions

The incentive and verification mechanisms proposed in this paper present certain characteristics that make them interesting for long-term peer-to-peer storage services such as P2P backup. On one hand, the secure and lightweight verification mechanism ensures that the user's chunks are safely stored in the peers as claimed. On the other hand, the usage of monetary incentives encourages peers to share their own resources proportionally to the ones they consume from the P2P system to backup their information. Furthermore, paying each successful operation performed by a peer, prevents users from wasting bandwidth and storage resources.

Although there are many micro-payment mechanisms that could be employed to implement the proposed incentive mechanism, for simplicity we have chosen delayed payments through a central bank (or multiple banks that trust each other). The bank also charges an additional fee for each monetary transaction it performs, which first provides funding for maintaining its infrastructure, but also helps to reduce its load since this is also is an incentive for peers to minimize the number of bank transactions.

Finally, the main contribution of this paper is the adoption of secure *digital cheques* to enable the long-term storage of information when an user goes off-line for extended periods of time. During the user's absence, the bank keeps verifying and updating the chunks on behalf of the user. These actions are performed by means of a compact set of verification challenges created by the user, and only when the peers try to cash their cheques. The proposed cost model for updating chunks in the off-line case also ensures that peers will try to cash their cheques as less as possible and, due to the higher cost of this operation compared to the on-line case, users will remain on-line as much as possible. Therefore, a higher number of peers will be available, which leads to a positive increment on the system resources and availability.

We validated our cheque-based mechanism and its associated incentive aspects using an ad-hoc, cycle based simulator. The obtained results show how the incentive mechanism allows the expected long-term storage capability as well as how well-behaved peers obtain a better performance with respect to worse-behaved peers.

Acknowledgments

This work has been funded by the by the Regional Government of Madrid under the MEDIANET project (S2009/TIC-1468).

References

[1] B. Yang and H. Garcia-Molina, "Ppay: micropayments for peer-to-peer systems," in *10th ACM conference on Computer and Communications Security (CCS'03)*, 2003, pp. 300–310.

[2] Z. Jia, S. Tiange, H. Liansheng, and D. Yiqi, "A new micro-payment protocol based on p2p networks," *IEEE International Conference on E-Business Engineering*, pp. 449–455, 2005.

[3] K. Wei, A. J. Smith, Y.-F. R. Chen, and B. Vo, "Whopay: A scalable and anonymous payment system for peer-to-peer environments," *International Conference on Distributed Computing Systems*, 2006.

[4] K. Chaudhary and X. Dai, "P2p-netpay: An off-line micro-payment system for content sharing in p2p-networks," *Journal of Emerging Technologies in Web Intelligence*, vol. 1, no. 1, 2009.

[5] N. Liebau, O. Heckmann, A. Kovacevic, A. Mauthe, and R. Steinmetz, "Charging in peer-to-peer systems based on a token accounting system," in *Lecture Notes in Computer Science*, 2006, vol. 4033, pp. 49–60.

[6] X. Dai, K. Chaudhary, and J. Grundy, "Comparing and contrasting micropayment models for content sharing in p2p networks," *International IEEE Conference on Signal-Image Technologies and Internet-Based System*, pp. 347–354, 2007.

[7] S. Seuken, D. Charles, M. Chickering, and S. Puri, "Market design & analysis for a p2p backup system," in *11th ACM conference on Electronic commerce*, ser. EC '10, 2010, pp. 97–108.

[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGPLAN Not.*, vol. 35, pp. 190–201, November 2000.

[9] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," in *5th Symposium on Operating Systems Design and Implementation (OSDI*, 2002, pp. 1–14.

[10] C. Batten, K. Barr, A. Saraf, and S. Trepetin, "pStore: A secure peer-to-peer backup system," Massachusetts Institute of Technology Laboratory for Computer Science, Technical Memo MIT-LCS-TM-632, October 2002.

[11] P. Druschel and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," pp. 75–80, 2001.

[12] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, "A cooperative internet backup scheme," in *USENIX Annual Technical Conference*, 2003.

[13] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, 2000.

[14] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *5th ACM conference on Electronic commerce*, 2004, pp. 102–111.

[15] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge, "Incentives for sharing in peer-to-peer networks," in *Second International Workshop on Electronic Commerce*, 2001, pp. 75–87.

[16] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster, "To share or not to share: An analysis of incentives to contribute in collaborative file sharing environments," in *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[17] C. Buragohain, D. Agrawal, and S. Suri, "A game theoretic framework for incentives in p2p systems," in *3rd International Conference on Peer-to-Peer Computing*, 2003.

[18] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer Systems, P2PECON*, 2003.

[19] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy," in *Symposium on Operating Systems Design and Implementation (OSDI)*, 2002, pp. 285–298.

[20] L. P. Cox and B. D. Noble, "Samsara: Honor among thieves in peer-to-peer storage," in *Nineteenth ACM Symposium on Operating Systems Principles*, 2003, pp. 120–132.

[21] G. Utard and A. Vernois, "Data durability in peer to peer storage systems," in *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004*, april 2004, pp. 90 – 97.

[22] L. Toka, M. Dell'Amico, and P. Michiardi, "Online data backup: A peer-assisted approach," in *Peer-to-Peer Computing*, 2010, pp. 1–10.

[23] P. Maille and L. Toka, "Managing a Peer-to-Peer Data Storage System in a Selfish Society," *Selected Areas in Communications, IEEE Journal on*, vol. 26, no. 7, pp. 1295–1301, 2008.

[24] P. Michiardi and L. Toka, "Selfish neighbor selection in peer-to-peer backup and storage applications," in *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*. Springer-Verlag, 2009, pp. 548–560.

[25] L. Pamies-Juarez, P. García-López, and M. Sánchez-Artigas, "Rewarding stability in peer-to-peer backup systems," in *16th International Conference on Networks, ICON 2008*. IEEE, 2008, pp. 1–6.

[26] ——, "Enforcing fairness in p2p storage systems using asymmetric reciprocal exchanges," in *IEEE International Conference on Peer-to-Peer Computing (P2P)*, Sept 2011, pp. 122 –131.

[27] N. Oualha and Y. Roudier, "Securing p2p storage with a self-organizing payment scheme," in *5th international Workshop on data privacy management, and 3rd international conference on Autonomous spontaneous security*. Springer-Verlag, 2011, pp. 155–169.

[28] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in *Fourth International Conference on Peer-to-Peer Computing*, ser. P2P '04, 2004, pp. 90–97.

[29] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Information and System Security*, vol. 14, pp. 12–34, Jun 2011.

[30] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (Proposed Standard), Internet Engineering Task Force, May 2008.

[31] J. Jonsson and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," RFC 3447 (Informational), Internet Engineering Task Force, Feb. 2003.

[32] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008.

[33] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Internet Engineering Task Force, Feb. 1997.