Computer Networks 83 (2015) 136–148

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Adaptive redundancy management for durable P2P backup

Matteo Dell'Amico [a,*], Pietro Michiardi [a], Laszlo Toka [b], Pasquale Cataldi [c]

[a] EURECOM, Sophia-Antipolis, France
[b] BME, Budapest, Hungary
[c] Nominet, Oxford, UK

ARTICLE INFO

Article history:
Received 18 February 2014
Received in revised form 1 September 2014
Accepted 5 March 2015
Available online 11 March 2015

Keywords:
P2P
Backup
Storage
Redundancy

ABSTRACT

We design and analyze the performance of a redundancy management mechanism for peer-to-peer backup applications. Armed with the realization that a backup system has peculiar requirements – namely, data is read over the network only during restore processes caused by data loss – redundancy management targets *data durability*, *i.e.* guaranteeing that data is not lost, rather than attempting to make each piece of information available at any time.

In our approach each peer determines, in an on-line manner, an amount of redundancy sufficient to counter the effects of peer deaths, while preserving acceptable data restore times. Our experiments, based on trace-driven simulations, indicate that our mechanism can reduce the redundancy by a factor between two and three with respect to redundancy policies aiming for data availability. These results imply an according increase in storage capacity and decrease in time to complete backups, at the expense of longer times required to restore data. We believe this is a very reasonable price to pay, given the nature of the application.

We complete our work with a discussion on practical issues, and their solutions, related to which encoding technique is more suitable to support our scheme.

1. Introduction

Online storage solutions are an extremely successful way of sharing and synchronizing data between machines, taking advantage of the ubiquity of Internet connectivity. Dropbox, Google Drive and Microsoft SkyDrive are only a few widely used examples within the plethora of applications that give this kind of service.

The aforementioned applications adopt a centralized "cloud" architecture, with all data residing on the data centers of a single vendor. Despite its success, such an architecture has some intrinsic shortcomings. Some of them have already shown up in news: data loss due to correlated failures [1], security blunders due to configuration errors [2]. Others, such as data theft from rogue employees, might happen eventually. Also, we argue that *long-term* storage is a case where the weaknesses of centralized storage are most important: indeed, the costs of storing large amounts of data over long periods are high, and services might shut down in the future as already happened to Drop.io [3], Nirvanix [4], Dell DataSafe [5], and Canonical's Ubuntu One [6], making data safety in the long run essentially impossible to evaluate.

Peer-to-peer (P2P) storage could solve these problems, providing cheap storage leveraging on excess bandwidth and disk space at the edge of the network. However, despite a considerable amount of research (see Section 2), P2P storage solutions failed to reach widespread usage.

* Corresponding author.
E-mail addresses: della@linux.it (M. Dell'Amico), pietro.michiardi@eurecom.fr (P. Michiardi), toka@tmit.bme.hu (L. Toka), pasquale.cataldi@nominet.org.uk (P. Cataldi).

http://dx.doi.org/10.1016/j.comnet.2015.03.006
1389-1286/© 2015 Elsevier B.V. All rights reserved.

Indeed, implementing a generic P2P storage application requires dealing with a variety of challenging problems, such as scalable handling of metadata, dealing efficiently with maintenance due to disk crashes, low-latency access and modification to individual files, and security issues such as ensuring data confidentiality even when usage permissions change dynamically. In this setting, keeping data available at all times in a situation of high churn is a daunting task [7].

We take a pragmatic approach: rather than trying to solve all the aforementioned issues at once and come up with a generic P2P file-system for the Internet, we design a system exclusively for *data backup*. Indeed, we argue that backup is a widely needed application that better fits the characteristics provided by P2P architectures. Costs and poor usability are among the main reasons why many existing backup solutions are not used: a P2P approach to data backup can be a viable technique to overcome such issues.

For backup applications, as we discuss in Section 3, the focus is on *durability*, which amounts to guaranteeing that data is not lost. The requirements for a specialized backup application are less stringent than those of generic storage in several aspects. First, backups should only be readable by their owner; this makes confidentiality requirements easy to satisfy with standard cryptographic techniques. Second, data backup often involves the bulk transfer of potentially large quantities of data, both during regular backups and, in the event of data loss, during restore operations. Therefore, read and write latencies of hours have to be tolerated by users. Third, owners have access to the original copy of their data, making it easy to inject additional redundancy in case data stored remotely is partially lost. Fourth, since data is read only during restore operations, the application does not need to guarantee that any piece of the original data should be promptly accessible in any moment, as long as the time needed to restore the whole backup remains under control.

In this work, we design and evaluate a new *redundancy management* mechanism tailored to backup applications. Simply stated, the problem of redundancy management amounts to computing the necessary redundancy level to be applied to backup data to achieve durability. The goal of this work is to design a mechanism that achieves data durability without requiring high redundancy levels nor fast mechanisms to detect node failures. Our solution to the problem stems from the particular data access workload of backup applications: data is written once and read rarely. The gist of our redundancy management mechanism, which is described in Section 4, is that the redundancy level applied to backup data is computed in an *on-line* manner. Given a time window that accounts for failure detection and data repair delays, and a system-wide statistic on peer deaths, a peer determines the redundancy rate during the backup phase. A byproduct of our approach is that, if the system state changes, then peers can adapt to such dynamics and modify the redundancy level on the fly.

The ability to compute the redundancy level in an *on-line* manner requires solving several problems related to coding efficiency and data management. In Section 5, we show how our scheme can be applied in practice, exploiting the properties of fountain coding.

Finally, we evaluate our redundancy management scheme using trace-driven simulations. In Section 6, we show that our approach drastically decreases strain on resources, reducing the storage and bandwidth requirements by a factor between two and three, as compared to redundancy schemes that use a fixed, system-wide redundancy factor. This result yields augmented storage capacity for the system and decreased backup times, at the expense of increased restore times, which is a reasonable price to pay if the specific requirements of backup applications are taken into account.

2. Related work

A claim that data can be backed up safely on a network of untrusted nodes may appear unintuitive and difficult to believe. Fortunately, several problems – which are orthogonal to the topic of redundancy – have been addressed in the literature. First, we provide an overview of solutions that can make safe backup feasible; we then conclude the section with an overview of how redundancy has been handled in related work.

2.1. P2P backup

Between the corpus of publications that target the broader topic of P2P storage, we are not the first to tackle backup. Many works provide a full system design, but focus on innovating on a few system components. Not unlike them, we focus on redundancy management; we note that in the large majority of cases a dynamic redundancy management mechanism such as ours can be implemented in those systems.

Early works [8–10] are based on distributed hash tables, and they focus on creating efficient de-duplication mechanisms; they however to not consider erasure coding techniques, which can drastically lower the required redundancy levels. Lillibridge et al. [11] adopt erasure coding and introduce a "symmetric storage" concept, where node A stores a data block for B only if B stores data for A; while restricting freedom in data placement, such mechanism provides incentives to cooperation. In Section 2.3, we discuss other ways of ensuring that peers behave correctly. Skowron and Rzadca [12] focus on the data placement problem: how to place data in order to optimize a given objective, such as time to backup or geographical data dispersion; such work considers the redundancy level to reach as an input to the system; it can be produced by a mechanism such as ours.

2.2. Data handling and security

In this work we assume the system saves opaque and immutable *backup objects*: these pieces of data should be encrypted so that only their owner can read them, and encode incremental differences between archive versions. Various techniques have been proposed to optimize computational time and size of incremental differences [13].

When more than one user back up the same piece of data, *deduplication* techniques can be used to avoid storing it more than once [14]. To protect user confidentiality, *convergent encryption* [9,10] can be used to guarantee that a user who does not own the files will not be able to guess their contents.

We consider a scenario where data maintenance is simple, being performed by a data owner with a local copy. When maintenance is delegated to nodes that do not have a local copy of the backup objects, various coding schemes can be used [15,16] to limit the amount of required data transit.

Using cryptography for the backed up data begs the question of where to backup the encryption key. This problem can be solved by generating the key as a function of an easy to remember secret: a *password*. Somehow contradicting common knowledge on the theme, recent research shows that a large portion of user-chosen passwords are resilient to guessing attacks [17–19], if appropriate standard techniques such as salting and strengthening [20] are used.

Efficient scheduling of data transfers is important to complete backups as soon as possible. Toka et al. [21] show that simple scheduling strategies such as least-available first are sufficient to obtain close-to-optimal transfer times.

2.3. Peer behavior and trust

It is reasonable to doubt on the fact that user behavior will be stable and trusted enough to guarantee that the backup will always be retrievable. Perhaps against intuition, however, it has been shown that user connectivity patterns are stable enough in the long run to allow predicting with good precision the probability that a given user will be using an application in a given moment, even *months* in advance [22].

Even if behaving in a stable way, it is conceivable that untrusted users may *cheat*, by erasing the data they are supposed to store: to avoid this problem, *provable data possession* protocols exist [23,24]: these protocols allow to verify whether a given peer is actually holding the data it is claiming to store. In addition, game-theoretic incentive models [25,26] have been devised to encourage peers to participate to the network according to the protocol: users who do not behave fairly will lose their backup. Another solution that provides incentives to behave well is segregating nodes in sub-networks with roughly homogeneous characteristics such as uptime and storage space [27,28]. Virtual currency [29–31] is yet another option to reward well-behaving peers.

An alternative approach is to store data on peers that are trusted to begin with. Besides easy use cases (*e.g.*, deploying the P2P backup application within a trusted organization), another possibility is to perform "friend-to-friend" storage [32,33], an approach where each user independently decides which users are trusted to store their data. More in general, trust needs not be limited to the nodes that a node directly knows: *reputation systems* can be used to build trust in networks of untrusted nodes: for more information, we point to the survey by Marti and Garcia-Molina [34].

2.4. Hybrid semi-decentralized systems

The episodes of cloud services shutting down we referred to in the Introduction [3–6] lead to believe that centralized cloud storage solutions cannot be considered safe on the long term. Conversely, P2P applications may be problematic *on the short term*, since the requirement of redundancy and limitations in availability and bandwidth increase the time needed to complete backups, and therefore to make data safe. A "best of both worlds" solution can be obtained by employing a hybrid architecture, where data is stored temporarily on data centers to complete backups as soon as possible, and then uploaded to peers when the backup is safe. Besides being safe both on the short term and on the long term, this solutions requires a fraction of the costs of centralized systems [35].

2.5. Redundancy

Redundancy rates and data repair techniques in P2P storage systems have been investigated from various angles. Earlier works [8–10] adopt simple replication strategies, resulting in higher storage and bandwidth costs for backing up and maintaining data in the system. In other proposals, erasure coding is used in order to obtain high durability while minimizing storage costs on nodes, but redundancy values are fixed parameters that are chosen by the designer independently of the system characteristics [11,36]. The Wuala online storage service encodes data on peers with a fixed redundancy level and avoids the need for maintenance by storing a full replica of the data in central servers [37]. More elaborate policies belong to two different categories: in some cases [38,39], redundancy is determined as a function of node failure rate in order to guarantee data durability at the expense of data availability. Many other approaches (*e.g.*, [40,41]) guarantee low latency through prompt data availability, but require high redundancy rates in typical settings. In contrast with these approaches, our proposal strives to provide *both* durability and performance at a low redundancy cost, relaxing prompt data availability by requiring that data becomes recoverable within a given time window. Finally, Pamies-Juarez et al. [42] investigate the relationship between redundancy and data retrieval times, but they center their investigations on cases where the online session length duration is orders of magnitude shorter than the length of a data transfer process; this scenario is clearly not applicable to our case of long restore processes.

3. Application scenario

Similarly to many online backup applications, we assume users (referred to as *data owners*) to specify one local folder containing important data to backup. Note that backup data remains available *locally* to data owners. This is an important trait that distinguishes backup from many online storage applications, in which data is only stored remotely.

We consider here the problem of long-term storage of large, immutable, and opaque pieces of data that we term *backup objects*. They consist of encrypted archives of changes to sets of files, such that recovering them allows reconstructing the history of data in the backup folder. We do not take into account the short-term storage of small modifications to the backup folder, which can be handled using known centralized or decentralized online storage solutions.

Backup objects are stored on remote peers, which are inherently unreliable. Peers may join and leave the system at any time, as part of their short-term online behavior: in the literature, this is referred to as *churn*. Moreover, peers may crash and possibly abandon the P2P application: this behavior is generally referred to as peer *death*. As such, the online behavior of peers must be continuously tracked, since it cannot be determined *a priori* [41].

While the literature provides a vast array of solutions to guarantee *data availability* when using failure-prone machines to store data [41,43], we claim that online data backup applications should instead target *data durability*. Moreover, backup applications often involve the bulk transfer of a large quantity of data. Therefore, such applications should cater throughput rather than aiming at low-latency read operations, in addition to be resilient against peer churn and deaths.

Similarly to data availability, data durability can be achieved by injecting a sufficient level of redundancy in the system. One key issue to address is to determine the redundancy level required to make sure data is not lost, despite peer churn. This problem is called *redundancy management*. A closely related problem is to deal with peer deaths, which cause the data redundancy level to drop. Hence, the focus of our work is to design a redundancy management mechanism that is tailored to the peculiar data access patterns of backup applications and that strives for data durability.

For the sake of clarity, we now explain the operation of a baseline P2P backup application. We gloss over the details of how data redundancy is achieved and discuss the salient phases of the life-time of backup data.

Using erasure coding, a backup object of size o is encoded in n *fragments* of a fixed size f which are ready to be placed on remote peers. Any k out of n fragments are sufficient to recover the original data[1]; when using optimal erasure coding techniques, $k = \lceil o/f \rceil$. The redundancy management mechanism determines the redundancy level $r = nf/o$.

During the **backup phase**, data owners upload fragments to some selected remote peers. We assume that any peer can collect a list of remote peers with available storage space: this can be achieved with known techniques, *e.g.* a central coordinator or a decentralized data structure such as a distributed hash table. The backup phase completes when *all* n fragments are placed on remote peers.

Once the backup phase is completed, the **maintenance phase** begins. The purpose of this phase is to reestablish the desired redundancy level in the system, that may decrease due to peer deaths: new fragments must be re-injected in the system. The crux of data maintenance is to determine when the redundancy of the backup object is too low to allow data recovery and to generate other fragments to rebalance it. In the event of a peer death, the system may trigger the maintenance phase immediately (eager repairs) or may wait for a number of fragments to be tagged as lost before proceeding with the repairs (lazy repairs) [41,15,16]. As such, it is important to discern *unambiguously* permanent deaths from the normal online behavior of peers: this is generally achieved by setting a time-out value, Θ, for long-term peer unavailability. Since user connectivity patterns have strong daily and weekly periodic behavior [22], typical practical choices for Θ are one or two weeks.

Note that, as peers hold a local copy of their data, maintenance can be executed solely by the data owner, or (as often done in storage systems) it can be delegated. In both cases, it is important to consider the timeframe in which data cannot be maintained. First, fragments may be lost before a host failure is detected using the time-out mechanism outlined above. This problem is exacerbated by the availability pattern of the entity (data owner or other peers) in charge of the maintenance operation: indeed, host failures cannot be detected during the offline periods. Second, data loss can occur during the restore process. For this reason, in Section 4, we consider a redundancy management policy that ensures data is not lost in the time-window $w = \Theta + a_{\text{off}}$, where a_{off} is the (largest) transient off-line period of the entity in charge of data maintenance. For example, if the data owner executes data maintenance: first, it needs to be on-line to generate new fragments and upload them, and second, the timeout Θ has to be expired. Additionally, our mechanism selects a redundancy level such that data loss does not occur before the restore process is completed.

Discerning dead peers through time-outs may lead to *false positives*, *i.e.* peers that are alive but considered as dead. This may trigger unnecessary maintenance, bringing redundancy levels to values that are higher than needed. False positives often have the result of triggering earlier maintenance operations that should anyway be carried out subsequently, and as long as this phenomenon is not extremely common, it only increases moderately the amount of resources used by the backup application.

In the unfortunate case of a disk or host crash, the **restore phase** takes place. Data owners contact the remote machines holding their fragments, download at least k of them, and reconstruct the original backup data.

Before proceeding, we now define the performance metrics we are interested in for this work. Overall, we compute the performance of a P2P backup application in terms of the amount of time required to complete the backup and the restore phases, labeled *time to backup* (TTB) and *time to restore* (TTR). Moreover, in the following sections, we use baseline values for backup and restore operations which bound both TTB and TTR. We compute such bounds as follows: let us assume an *ideal* storage system with unlimited capacity and uninterrupted online time that backs up user data. In this case, TTB and TTR only depend on the size of a backup object and on uplink bandwidth and availability of the data owner. We label these ideal values *minTTB* and *minTTR*. Formally, we have that a peer i with upload and download bandwidth u_i and d_i, starting the backup of an object of size o at time t, completes its backup at time t',

[1] For non-optimal erasure-coding techniques such as fountain coding, as described in Section 5, this guarantee is given probabilistically.

after having spent $\frac{o}{u_i}$ time online. Analogously, i restores a backup object with the same size at t'' after having spent $\frac{o}{d_i}$ time online. Hence, we have that

$$minTTB(i,t) = t' - t$$

and

$$minTTR(i,t) = t'' - t.$$

We use these reference values throughout the paper to compare the relative performance of our P2P application versus that of such an ideal system.

4. Redundancy management

We now discuss the key idea of our work: a redundancy management mechanism to achieve data durability. In practice, data can be considered as durable if the probability to lose it, due to the permanent failure of hosts in the system, is negligible. The problem of designing a system that guarantees data durability can be approached under different angles.

As noted in previous works [44,39], data availability implies data durability: a system that injects sufficient redundancy for data to be available at any time, coupled with maintenance mechanisms, automatically achieves data durability. These solutions are, however, too expensive in our scenario: the amount of redundancy needed to guarantee availability is much higher than what needed to obtain durability.

Instead of using high redundancy, data durability can also be achieved with efficient maintenance techniques. For example, in a datacenter, each host is continuously monitored: based on statistics such as the mean time to failure of machines and their components, it is possible to store data with very little redundancy and rely on system monitoring to detect and react immediately to host failures. Failed machines are replaced and data is rapidly repaired due to the dedicated and over-dimensioned nature of datacenter networks. Unfortunately, this approach is not feasible in a P2P setting. First, the interplay of transient and permanent failures makes failure detection a difficult task. Since it is difficult to discern deaths from the ordinary online behavior of peers, the detection of permanent failures requires a delay during which data may be lost. Furthermore, data maintenance is not immediate: in a P2P application deployed on the Internet, bandwidth scarceness and peer churn make the repair operation slow.

In summary: on the one hand durability could be achieved with high data redundancy, but the cost in terms of resources required by peers would be overwhelming. On the other hand, with little redundancy, durability could be achieved with timely detection of host failures and fast repairs, which are not realistic in a P2P setting.

The goal of this work is to design a redundancy management mechanism that achieves data durability without requiring high redundancy levels nor fast failure detection and repair mechanisms. Our solution to the problem stems from the particular data access workload of backup applications: data is written once, during backup, and read (hopefully) rarely, during restores. Hence, we design a mechanism that injects only the data redundancy level required to compensate failure detection and data repair delays.

When any lost piece of data is immediately repaired, data is never lost. In real systems, though, there are delays between data losses and repairs: we therefore define durability in function of a delay t accounting for such delays.

Definition 1. Data durability d is the probability to be able to access data after a *time window* t, during which no maintenance operations are executed.

Definition 2. The time window t is defined as $t = w + TTR$, where w accounts for failure detection delays and TTR is the time required to download a number of fragments sufficient to recover the original data.

As discussed in Section 3, w depends on whether the maintenance is executed by the data owner or is delegated, and can be thought of a parameter of our scheme.

The goal of our redundancy management mechanism is to determine the data redundancy that achieves a target data durability: we proceed as follows. A peer with n fragments placed on remote peers could lose its data if more than $n-k$ of them would get lost as well within the time window t. The data redundancy required to avoid this event is $r = n/k$.

Peer deaths can be determined by disks and host crashes, or by human events such as users uninstalling the application and leaving the network. Disk drives in practical settings have a lifetime of several years on average [45]; in the evaluation section, we will evaluate our strategies in challenging situations where peer lifetime ranges between a few months and a few years. Let us consider the probability of a node to be alive after a time t to be $A(t)$. Assuming death events are independent, data durability writes as:

$$d = \sum_{i=k}^{n} \binom{n}{i} (A(t))^i (1 - A(t))^{n-i}. \qquad (1)$$

Eq. (1) depends on t which, in turn, is a function of TTR. However, peers cannot readily compute their TTR, as this quantity depends on the characteristics of remote peers hosting their fragments. We thus propose to use the following heuristic as a method to *estimate* the TTR. Suppose peer p_0 is computing an estimate of its TTR. In the event of a crash, we assume p_0 to remain online during the whole restore process. In such a case, assuming no network bottlenecks, its TTR can be bounded for two reasons:

1. the download bandwidth D_0 of peer p_0 is the bottleneck;
2. the upload rate of remote peers holding p_0's data is the bottleneck.

Let us focus on the second case: we define the *expected upload rate* μ_i of a generic remote peer p_i holding a backup fragment of p_0 as the product of the availability of peer p_i and its upload bandwidth, that is $\mu_i = u_i a_i$.

Peer p_0 needs to download at least k fragments to fully recover a backup object. Let us assume these k fragments are served by the k remote peers with the highest expected upload rate μ_i. In this case, the "bottleneck" is the kth peer with the lowest expected upload rate μ_k. Then, an estimation of TTR, that we label *eTTR*, can be obtained as follows:

$$eTTR = \max\left(\frac{o}{D_0}, \frac{o}{k\mu_k}\right). \tag{2}$$

While TTR is the *real* time to restore that can only be measured *a posteriori*, eTTR is an estimation for TTR, and it computed *a priori* according to Eq. (2). The availability of the downloading peer is assumed equal to 1, since we consider that a node performing a restore will remain online until the restore is completed, to obtain a working system as soon as possible, and to minimize the risk of data loss due to other peer deaths during the restore period.

We now set off to describe how our redundancy management scheme works in practice: the redundancy level applied to backup data is computed by the combination of Eqs. (1) and (2). Let us assume, for the sake of simplicity, the presence of a central coordinator that performs membership management of the P2P network: the coordinator keeps track of users subscribed to the application, along with short-term measurements of their availability, their (application-level) uplink capacity and the average death rate T in the system. While a decentralized approach to membership management and system monitoring is an appealing research subject, it is common practice (*e.g.*, Wuala [37] to rely on a centralized infrastructure and a simple heartbeat mechanism.

During a backup operation, peers query the coordinator to obtain remote hosts that can be used to store fragments, along with their availability. A peer constructs a backup object, and subsequently uploads k fragments to distinct, randomly selected available remote hosts. Then the peer continues to inject redundancy in the system, by sending additional fragments to randomly selected available peers, until a stop condition is met. Every time one (or more) new fragment is uploaded, the peer computes d and eTTR: the stop condition is met if $d \geqslant \sigma_1$ and $eTTR \leqslant \sigma_2$. σ_1 and σ_2 are configuration parameters that tune the system according to the performance metrics that we target in this work: durability and time to restore. Selecting an appropriate durability target σ_1 should be easy, according to the guarantees required by the user; in the following we define σ_2 as $\sigma_2 = \alpha \cdot minTTR$, where α is a parameter that specifies the degradation of TTR with respect to an ideal system, tolerated by users.

We now discuss in details the influence of the two stop conditions on the behavior of our mechanism. $A(t)$ is the survival function (or complementary cumulative distribution function) of peer mortality, therefore the average peer lifetime can be computed as $T = \int -xA\prime(x)dx$. Given Eq. (1), we study the impact of the ratio $\frac{w+eTTR}{T}$:

- $T \gg w + eTTR$: this case is representative of a "mature" P2P application in which the dominant factor that characterizes peer deaths are permanent host failures, rather than users abandoning the system. Hence, $A(t)$ in Eq. (1) is close to 1 since t is estimated as $w + eTTR$, which implies that the target durability σ_1 can be achieved with a small n.
  As such, the condition on $eTTR \leqslant \sigma_2$ prevails on $d \geqslant \sigma_1$ in determining the redundancy level to apply to backup data. This means that the accuracy of the estimate eTTR plays an important role in guaranteeing acceptable restore times; instead, errors on eTTR have no impact on data durability.
- $T \sim w + eTTR$: this case (including also the case when $T < w + eTTR$, i.e., the average offline time of the data owner plus the estimated time it takes to fetch back its data can be greater then the average lifetime of peers) is representative of a P2P application in the early stages of its deployment, where the abandon rate of users is crucial in determining the death rate. In this case, the exponential in Eq. (1) can be arbitrarily small, which implies that $n \gg k$, *i.e.*, the target durability d requires higher data redundancy.
  In this case, the condition $d \geqslant \sigma_1$ prevails on $eTTR \leqslant \sigma_2$. Hence, estimation errors on the restore times may have an impact on data durability: *e.g.*, underestimating the TTR may cause n to be too small to guarantee the target σ_1. In Section 6, we study this scenario.

In summary, the key idea of our redundancy management mechanism is that the redundancy level applied to backup data is computed in an *on-line* manner, during the backup phase. This comes in sharp contrast to computing the redundancy level in an *off-line* manner, solely based on system-wide statistics, that characterize previous approaches to redundancy management.

A by-product of our approach is that our mechanism can *adapt* the redundancy rate r each peer applies to its data based on system dynamics. Now, we must prove that the system reaches a *stable state*: system dynamics must not bring the redundancy mechanism to oscillate around r. Based on Eqs. (1) and (2), we face a retroactive system in which a feedback loop exists on the durability d. Given a target durability d, a system-wide average death rate T and a time window $t = w + eTTR$, we can derive r. The problem is that eTTR depends on the short-term behavior of peers as well as the redundancy rate r.

First, we study how eTTR and d vary as a function of the redundancy rate r.

Proposition 1. *eTTR is a non-increasing function in r.*

Sketch of the proof: Recall that $r = \frac{nf}{o}$. Let us assume a peer p_0 has the following ranked list of remote peers:

$$\{\mu_1, \mu_2, \mu_3, \ldots, \mu_k\},$$

where without loss of generality, $\mu_i < \mu_j \forall i < j$. If r increases, then n increases: new fragments must be stored on new remote peers. For simplicity, assume a single fragment is to be placed on peer p_q.

Two cases can happen:

1. $\mu_q < \mu_k$; in this case, eTTR remains unvaried, since p_q is "slower" than the kth peer used to compute eTTR.

2. $\mu_q > \mu_k$; in this case, p_q "ejects" the current kth peer from the ranked list defined above.

As such, eTTR can only decrease. Note that eTTR may not reach the stop condition σ_2 if the parameter α is not appropriately chosen: simply stated, a *plateau* value of eTTR exists when placing fragments on all peers in the network.

Proposition 2. *d is an increasing function in r.*

Sketch of the proof: Eq. (1) is a composite function of eTTR. Hence, by increasing r, new fragments have to be placed on remote peers and it is not guaranteed, in general, that this contributes to decrease d. However, thanks to Proposition 1, eTTR is non-decreasing in r, hence $t = w + eTTR$ is non decreasing in r. As a consequence, d is an increasing function in r.

We can now state the following proposition:

Proposition 3. *The redundancy management mechanism presented in this section is stable.*

Sketch of the proof: By design, our redundancy mechanism shall only increase r. Now, Proposition 1 states that increasing r yields lower values of eTTR, hence, eventually, the system either arrives at the stop condition $eTTR \leqslant \sigma_2$, when α is chosen appropriately, or it reaches the plateau defined above. Similarly, by Proposition 2, increasing the redundancy in the system implies that d grows asymptotically to 1, hence the system eventually reaches the stop condition $d \geqslant \sigma_1$.

It is natural to ask why in Proposition 3 we omit the possibility of removing fragments from remote peers if r is too high. Let us consider such an operation: one possibility would be to drop a remote fragment at random. This operation would be unstable: indeed, for example, deleting a fragment from the "fastest" peer in the ranked list defined above would increase eTTR, decrease d, which as a consequence might require to re-inject a fragment. Instead, we could delete fragments starting from the "slowest" peer: in this case, the drop operation would be stable, but the storage load in the system may eventually become concentrated on fast peers only. Moreover, avoiding deletions can spare maintenance operations in the future should one or more of the remaining fragments on remote peers be lost. Due to these reasons, in this work we do not allow fragments to be dropped.

5. Coding and data management

With the redundancy management mechanism described in Section 4, the redundancy level applied to backup data is computed in an *on-line* manner. Instead, the redundancy rate used in most related work is usually computed *off-line*, given sufficiently representative statistics on the system, including transient and non-transient failures. These system-wide statistics are used to compute a unique redundancy rate that every peer will use. Instead, our approach requires each peer to compute an *individual* redundancy level: the time window t is a function of *eTTR*, which is different for every peer.

Erasure coding [46] introduces data redundancy by transforming an original file composed of fragments into a longer file such that the original file can be recovered from a subset of the encoded fragments. More formally, assuming the backup object to be segmented in blocks of k fragments each, each portion of the original data will be recovered if a sufficient number of the n encoded fragments will be successfully received. An erasure code is optimal if any k out of the n encoded fragments are sufficient to recover the original block. The *code rate*[2] is defined as $r = n/k$ and represents the number of "redundant" fragments per "useful" fragments generated by the encoder. Note that optimal codes are often costly when n is large: practical solutions usually have *quadratic* encoding and decoding complexity.

Among the optimal erasure coding techniques, Reed–Solomon (RS) codes are the most widely used in a number of applications [47]. Nevertheless, these codes lack of flexibility as the encoding is determined by the couple of parameters k and n, which are fixed a priori.

Another family which has been vastly studied in the literature are the Fountain Codes. These codes have found applications to digital communications [48], content delivery [49], storage [50] and P2P [51]. Because of their unique characteristics, they are particularly suitable to our goals too. In fact, the generation of an encoded fragment is independent from the others (*on-the-fly* property) and the number of encoded fragments that can be generated from the original data is potentially infinite (*rateless* property).

Fountain Codes are not optimal, in the sense that the number of encoded fragments necessary to recover the original data is slightly larger than the original number of fragments. This inefficiency depends on the parameters of the coding technique and on the block size[3] and is negligible for large data blocks. In practice, the loss of efficiency is acceptable, if one considers the increased computational efficiency (even linear with the block size) of this family of codes with respect to RS codes (typically quadratic).

In the context of this work, Fountain Codes are very simple to use in practice. Indeed, the information about the fragment generation should be shared between the encoder and the decoder.[4] Instead, in our application, the encoder and the decoder coexist in the same entity: the data owner. Hence, such information needs not a complex infrastructure to be set up between separate communicating parties, but can be simply treated as "metadata" information to be stored locally (and eventually backed up).

Fountain Codes make the mechanism described in Section 4 trivial to achieve: as long as the conditions on the *eTTR* and d are not met, the encoder continues to generate new unique encoded fragments on the fly. When the stop condition is reached, the encoding process terminates.

[2] In this work, code rate and redundancy rate are used as synonyms.

[3] In the context of Fountain Codes, the encoding block is defined exclusively by the number of fragments k, n not being defined *a priori*.

[4] This information can be transmitted together with the encoded fragment, or the choice of the degree distribution and the random generator can be shared.

In case system dynamics trigger the generation of new encoded fragments (*e.g.* because host availability decreases), these can be simply generated as needed, with the same procedure described above.

Fountain encoded fragments are statistically "interchangeable": any encoded fragment can be used to reconstruct the original data and any encoded fragment can be replaced by any newly generated encoded fragments. As a consequence, also maintenance operations are simplified as peers need not track of the exact encoded fragment to replace.

Another appealing characteristic of Fountain Codes is that, unlike RS codes, the block size is not mathematically constrained. Nevertheless, a solution based on Fountain Codes is not exempt from data management problems. While these codes allow maximum flexibility, the definition of the block size is a tradeoff between the coding inefficiency (which suggests to use large blocks) and the number of operations required for either encoding or decoding (even if linear-time implementations exist, memory and delay considerations suggest to use shorter blocks). This means that given a potentially large backup object, what is the best strategy, encoding the whole data thus minimizing the coding inefficiency, or segmenting into smaller blocks decreasing the complexity?

We argue that the data object should be partitioned in several blocks whose size should depend not only on coding complexity and inefficiency, but also on the user data generation rate. One of the coding strategies that can increase the performance of the code whilst maintaining shorter block size is the sliding-windowing approach [52,53]. This approach virtually increases the encoding block by allowing the overlap of two or more subsequent coding blocks (referred as "windows"). The block overlap is a design parameter that impacts the performance of the code and its value can be decided *a priori* or according to customized coding strategies. The typical drawback of using the sliding-windowing approach is an increase in the decoding delay and memory consumption, as shown by Bogino et al. [52]. However, for the considered application, if the block size is moderate, their impacts are acceptable.

The optimal design of the codes for this application goes beyond the scope of this paper as the coding parameters depend on the system and its characteristics. To give an example, for unreliable IP networks, such as mobile networks, it is advisable to fit one encoded fragment (and its associated metadata, if required) per IP packet; in this way, the loss of a packet corresponds to the erasure of a single fragment, which is preferable than having several fragments per packet. This choice impacts the design of the code and its performance: given a fixed size of the data, one fragment per IP packet means that either the data will be segmented in a larger number of blocks, or that the blocks will contain a higher number of fragments.

Nevertheless, in our context where we can assume the TCP protocol to work well, there is no need to constraint the size of the fragment to the IP packet size. In fact, the fragment size can be significantly larger, thus having blocks composed by fewer fragments and therefore decreasing the encoding/decoding computation time. As we will see in Section 6.2, this is the encoding strategy that we used in our system evaluation.

6. Performance evaluation

In the following, we proceed with a trace-driven system simulation, and focus on the performance metrics outlined in Section 3. That is, we are interested in studying the time required to backup and restore user data: we perform a comparative study of the results achieved by a system using our redundancy management scheme and the traditional approach used for storage applications. For the latter case, we implement a technique in which the coding rate is set once and for all based on a system-wide average of host availability.

Note that, for the purpose of our study, it is not necessary to implement in detail the coding mechanisms described in Section 5. All we need to know for the evaluation of transfer times is the number of fragments each peer has to upload during the backup operation.

We use traces as input to our simulator that cover both the online behavior of peers and their uplink and downlink capacities. Instead, long-term failures and the events of peers abandoning the applications, which constitute the peer deaths, are generated synthetically as described in Section 6.2. Due to the lack of traces that represent the realistic "data production rate" of Internet users, in this simulation study we confine our attention to a homogeneous setting: each user has an individual backup object of the same size.

6.1. Datasets

6.1.1. Availability trace

The *online behavior* of users, i.e., their patterns of connection and disconnection over time, is difficult to capture analytically. In this work we simulate a backup application using a real application trace that exhibits both heterogeneity and correlated user behavior. Our traces capture user availability, in terms of login/logoff events, from an instant messaging (IM) server for a duration of roughly 3 months [22]. We argue that the behavior of regular IM users constitutes a representative case study. Indeed, for both an IM and an online backup application, users are generally signed in for as long as their machine is connected to the Internet; as it can be gleaned from Fig. 1, in this dataset it is possible to observe strong diurnal and weekly patterns. Moreover, users have heterogeneous behavior – for example, some users often stay connected during workdays while others have a less predictable uptime.

In this work we only consider users that are online for an average of at least four hours per day, as done in Wuala [37]. Once this filter is applied, we obtain the trace of 376 users. Since in P2P storage systems the number of neighbors each node interacts with is very often limited by design and scalability issues [21], we believe this trace size is acceptable. As shown in Fig. 2, most users are online for less than 40% of the trace length, while some of them are almost always connected.

6.1.2. Bandwidth distribution

Uplink capacities of peers are obtained by sampling a real bandwidth distribution measured at more than 300,000 unique Internet hosts for a 48 h period from roughly 3500 distinct ASes across 160 countries [54]. These values have a highly skewed distribution, with a median of 77 KBps and a mean of 428 KBps. To represent typical asymmetric residential Internet lines, we assign to each peer a downlink speed equal to four times its uplink.

6.2. Simulation settings

The trace-driven online behavior of a peer is overridden only during the restore phase: in this work we make the assumption that in such case, a peer remains online for the whole duration of the restore process.

In our study, each peer has $o = 10$ GB of data to backup (as soon as the simulation begins), and dedicates 50 GB of storage space to the application. The high ratio between these two values lets us disregard issues due to insufficient storage capacity and focus on the subjects of our investigation. The fragment size is set to 160 MB, implying a minimum of $k = 64$ fragments needed for restores. In this simulation we do not consider the (negligible) inefficiency of fountain coding and we assume that 64 fragments will always be sufficient to recover the original data; in practical settings, a solution based on fountain coding will pay for added flexibility and efficiency with a slightly higher redundancy [55].

We define peers' lifetimes to be exponentially distributed random variables with an expected value $T = \{90\text{ days}, 1\text{ year}, 4\text{ years}\}$, resulting in $A(t) = e^{-t/T}$. These conservative values are noticeably lower than the disk failure rates measured in real-world scenarios [45] (see Section 4). Besides peer deaths, we study the impact of the w parameter, which contributes to the duration of the time-window for which our redundancy management policy guarantees data durability, without maintenance (see Section 4). As a reminder (see Section 3), w accounts for failure detection delays. In our experiments w takes values from 0 to 4 weeks.

Our adaptive redundancy policy uses the following parameters: we set the thresholds $\sigma_1 = 0.9999$, so that the durability $d \geqslant \sigma_1$ and $\sigma_2 \leqslant \max(1\text{ day}, 2 \cdot \mathit{minTTR})$ so that $\mathit{eTTR} \leqslant \sigma_2$. In this work, we compare against a baseline redundancy policy that aims to guarantee data availability [41], labeled here as "availability-based": in this case data availability is computed as

$$\bar{a} = \sum_{i=k}^{n} a^k (1-a)^{n-k},$$

where a is the average availability of nodes in the system. The value chosen for n is the lowest that satisfies $\bar{a} \geqslant \hat{a}$, where $\hat{a}$ is a target value set by the user. Here we set a target data availability of $\hat{a} = 0.99$, and use the system-wide average availability $a = 0.36$ as computed from our availability traces. Hence, we obtain a value $n = 228$ and a redundancy rate $n/k = 3.56$.

For each set of parameters, the simulation results are obtained by combining those of ten simulation runs.

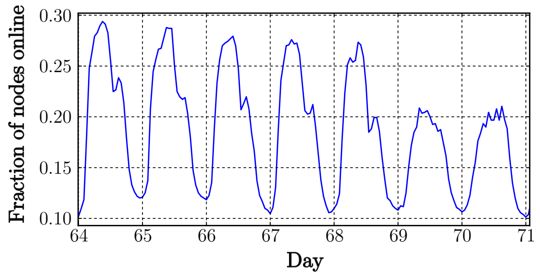

Fig. 1. Ratio of connected nodes in a representative week of the availability trace. More users are connected during days than during night; in weekends, the total number of connected users drops.

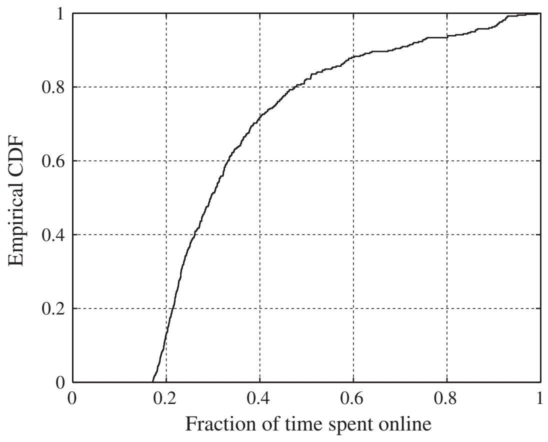

Fig. 2. CDF of the host availability from our traces. Note that users spending less than 4 h per day online are filtered from our data.

6.3. Results

We begin our discussion by showing the bounds on TTB and TTR, as defined in Section 3. Fig. 3 shows the cumulative distribution functions (CDF) of minTTB and minTTR obtained using the input traces discussed above. Our working assumption is that peers stay online during restore operations: as such, only the (ordinary) backup phase suffers from peer unavailability, and the distribution of minTTR depends only on the bandwidth distribution, while minTTB also depends on the availability traces. We notice that, for a large majority of users, even these ideal values for time to backup and to restore are of the order of hours; this result is in line with what is reported in other works [33,56,12].

While backup operations generally take days to complete, for a file size of 10 GB, restore operations are several times faster. This can be simply explained by the asymmetric bandwidth setup we use in our simulations, and – as discussed above – by unavailability of peers when data needs to be backed up. Since node bandwidth distribution is skewed, a few nodes with very large bandwidth experience a much lower value for both minTTR and minTTB; the tails with a very long minTTB value are instead due to peers that remained disconnected for very long time spans in our traces.

We now proceed to a detailed comparative study of our scheme to the traditional fixed-redundancy scheme. First, we focus on the data redundancy level (that is, the code rate r) imposed by each approach.

In Fig. 4, we show the average redundancy factor for our mechanism and the one computed for the availability-based scheme (which is fixed), as a function of the parameter w and for different values of T. We omit error bars from the plot as the variance around the mean is negligible. Clearly, for increasing values of w the redundancy rate increases, as it is possible to evince from Eq. (1). Note that our simulations account for a realistic bandwidth distribution and for some real on-line user behavior, which influence the eTTR computation. Fig. 4 also illustrates the impact of T: when the dominant effect of non-transient failures is the reliability of Internet hosts, that is T is large, our mechanism achieves data durability (and a controlled TTR) with a small redundancy factor. Instead, when peer deaths are dominated by peers abandoning the system, that is T is small, our mechanism compensates with a larger redundancy rate. In summary, our redundancy management scheme obtains a redundancy factor ranging roughly between *half* and *a third* of the availability-based scheme, increasing the storage capacity of the system by a corresponding factor between two and three. Since the amount of data to upload in case of a disk crash is proportional to the redundancy level, the impact of maintenance of system bandwidth decreases accordingly.

In addition to improving the aggregate storage capacity of the system, our redundancy management scheme impacts both backup and restore operations. Figs. 5 and 6 report the CDF of the ratio of TTB and TTR over their respective ideal counterparts, minTTB and minTTR. These plots are obtained with different values of w, for a fixed $T = 3$ months,[5] and illustrate the results of our mechanism and that achieved by the availability-based scheme. Fig. 5 indicates that, due to a lower redundancy factor, the median of the distribution of TTB is roughly reduced by a factor of four. Moreover, increasing values of w have essentially little impact on TTB. Fast backups are counterbalanced by longer restores: as shown in Fig. 6, restore operations take more time to complete w.r.t. a traditional approach to redundancy management. Here the parameter w plays an important role: for small w values, little redundancy is applied to backup data. As such, the opportunity to retrieve enough encoded fragments to restore data is largely affected by peer availability. Instead, when w is large, restore operations are more efficient and less sensitive to peer availability.

In summary, our results support the rationale underlying the design of our redundancy management scheme: TTB is generally several times larger than TTR, even in an ideal case (as shown in Fig. 3). Because of this unbalance, we argue that it is reasonable to use a redundancy management scheme that trades longer TTR (which affects only users that suffer a crash) for shorter TTB (which affects all users).

[5] We present results for $T = 3$ months because the effects of w are more marked. We obtain similar qualitative results for larger values of T. Also, for clarity of presentation, we omit the CDF for $w = 4$ weeks.

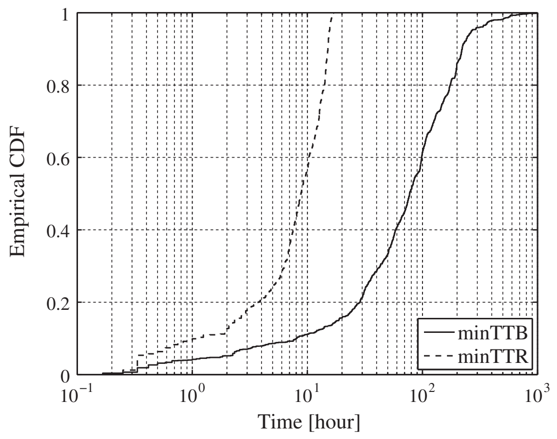

Fig. 3. CDF of the ideal times to transfer data: *minTTB* and *minTTR*.

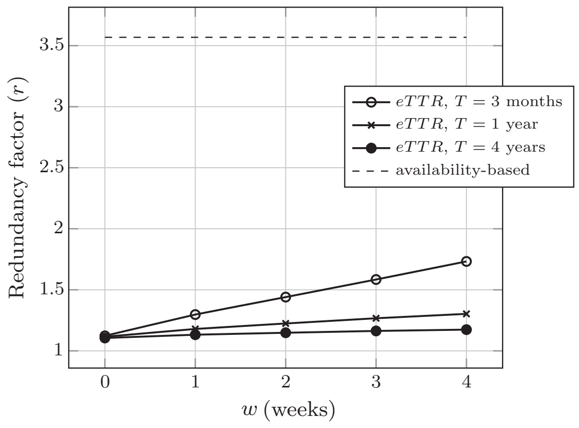

Fig. 4. Redundancy rate as a function of w, for different values of T.

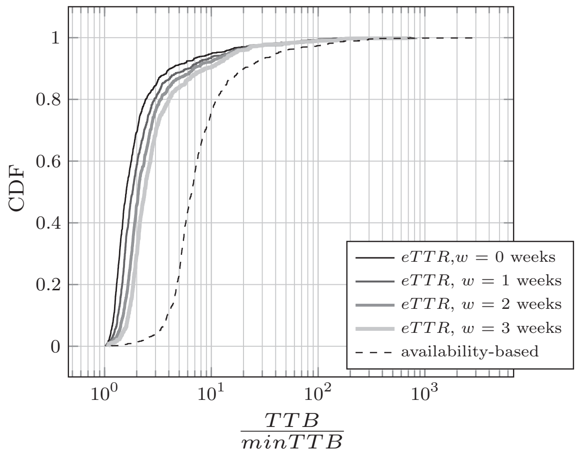

Fig. 5. CDF of *TTB* for different values of w, $\tau = 3$ months.

Now, we dive into the details of our scheme and study its sensitivity to errors due to the heuristic we use to estimate TTR. The main reason for errors on eTTR are due to the fact that the heuristic defined in Eq. (2) assumes k

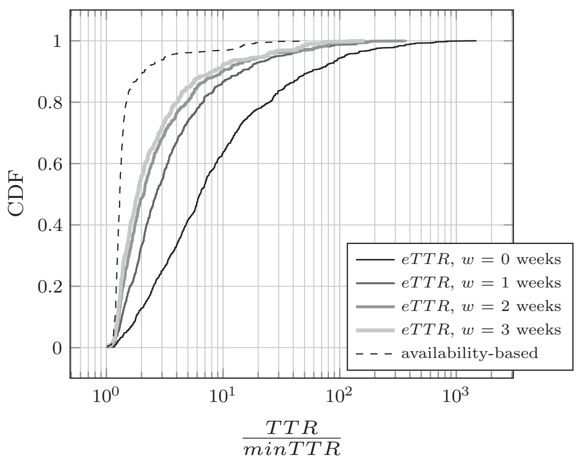

Fig. 6. Distribution of *TTR* for different values of w, $\tau = 3$ months.

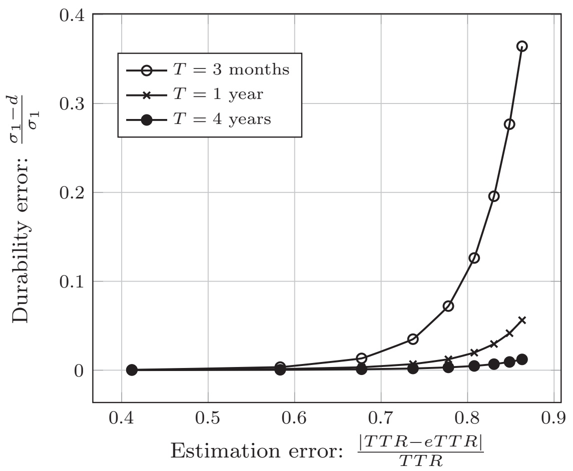

Fig. 7. Correlation between estimation errors and data durability.

encoded fragments to be downloaded from the k fastest peers that hold backup data. In practice, however, the k encoded fragments are downloaded from the peers that are available when a restore operation is executed. Depending on the bandwidth distribution of the peers in the system, such difference can cause the estimated TTR value to be different from what achieved in practice.

Now, if eTTR is larger than TTR, more redundant data is injected in the system, which has no negative impact on data durability. What is the impact on durability if peers underestimate the TTR? Using Eq. (1), we compute the redundancy factor r, as a function of eTTR, that meets the traget durability σ_1. Then, using TTR and r, we compute the data durability d. Fig. 7 shows the impact of the relative estimation error on the relative durability error using the procedure described above, for different values of T and for $w = 2$ weeks.

When T is large, we have that $w + eTTR \ll T$: as such, even large estimation errors have little impact on the durability d. Instead, when T is small, we have that $w + eTTR \simeq T$: in this case, data durability is more sensitive to estimation errors. As a consequence, data redundancy may not be sufficient and data loss events may occur.

Table 1
Categorization of data loss events.

Avg. lifetime (T)	Total events (%)	Incompl. backup		Failed restore
		Total (%)	Unav. (%)	
3 months	13	10.4	8.4	**2.6**
1 year	2.6	2.6	2.3	**None**
4 years	0.5	0.5	0.25	**None**

In Table 1, we illustrate the effects discussed by quantifying data loss events for $w = 2$ weeks. Here we count the percentage of peers that have not been able to restore their data after a local disk crash, averaged over ten simulation runs. We break down the data loss cases between *incomplete backup* and *failed restore*: the latter case encompasses all cases where peers lose data after completing their backup. Furthermore, we also specify the percentage of *unavoidable* cases in which peers fail before *minTTB*: in this case, not even an ideal system could have guaranteed a safe backup.

A lesson we can draw from Table 1 is that most data loss episodes are simply due to node failure *before the backup is completed*; this result confirms that it is sensible to optimize time to backup by reducing redundancy and hence also network load. We remark that a further possibility is to use a hybrid architecture, as we discussed in Section 2.4, to store data temporarily on a centralized "cloud" service to decrease time to backup [35]. In addition, it can be noted that a large majority of data loss episodes are *unavoidable* with any online storage solution: nodes with low bandwidth risk crashing before completing uploads even if saving data to a reliable server with 100% uptime and unlimited bandwidth.

"Failed restore" events can be seen as validations for the durability computed in Eq. (1): since backup is considered complete, the system has reached the condition where $d \geqslant \sigma_1 = 0.9999$: this should imply that failed restores are less than 0.01%. This happens for $T \geqslant 1$ year; the problematic case of $T = 3$ months is imputable to the impact of estimation error on durability as discussed above. However, we remark that the impact of this effect even in such a situation is outnumbered by the unavoidable data loss episodes; this leads us to conclude that nodes with very low lifetime are intrinsically unsuited to any kind of online storage solution, and not only to P2P backup.

7. Conclusion

In this work we focused on P2P backup systems, and designed a redundancy management mechanism tailored to the specific data access patterns that characterize data backup. The goal of our mechanism was to achieve data durability without requiring large redundancy factors (typical of storage applications) nor fast failure detection mechanisms.

Our experiments showed that, in a realistic setting, a redundancy that caters to data durability can be less than half of what is needed to guarantee availability. This results in a system with a storage capacity that is more than doubled, and backup operations that are much faster (up to a factor of 4) than on a backup system based on traditional redundancy management. This latter property is particularly desirable since, in most of the cases, peers suffering data loss were those that could not complete the backup before crashing.

We also showed that the price to pay for efficient backup operations was a decreased (but controlled) performance of restore operations. We argued that this was a reasonable penalty, considering that all peers in the system would benefit from backup efficiency, while only those peers suffering from a failure would have to bear longer restore times.

Finally, we studied data loss events: our results indicated that such events are practically negligible for a mature P2P application in which permanent host failures dominate peer deaths. We also showed the limitations of our technique for a system characterized by a high application-level churn, which is typical of new P2P applications that must conquer user trust.

References

[1] R. Wauters, Online Backup Company Carbonite Loses Customers' Data, Blames and Sues Suppliers, TechCrunch, 2009. <http://tcrn.ch/dABxRn>.
[2] A. Ferdowsi, Yesterday's Authentication Bug, Blog Post, 2011. <http://blog.dropbox.com/?p=821>.
[3] drop.io, An Important Update on the Future of drop.io, Blog Post, 2010. <http://http://blog.drop.io/2010/10/29/an-important-update-on-the-future-of-drop-io/>.
[4] T. Coughlin, Nirvanix Provides Cautionary Tale for Cloud Storage, Forbes, 2013. <http://www.forbes.com/sites/tomcoughlin/2013/09/30/nirvanix-provides-cautionary-tail-for-cloud-storage/>.
[5] V. Covic, Dell Shuts Down Dell Datasafe Cloud Storage and Backup Service, Cloudwards.net, 2014. <http://www.cloudwards.net/news/dell-shuts-down-dell-datasafe-cloud-storage-and-backup-service-3884/>.
[6] J. Silber, Shutting Down Ubuntu One File Services, Canonical Blog, 2014. <http://blog.canonical.com/2014/04/02/shutting-down-ubuntu-one-file-services/>.
[7] C. Blake, R. Rodrigues, High availability, scalable storage, dynamic peer networks: pick two, in: USENIX HotOS, 2003.
[8] C. Batten, K. Barr, A. Saraf, S. Trepetin, pStore: A Secure Peer-to-peer Backup System, Unpublished Report, MIT Laboratory for Computer Science.
[9] L. Cox, B. Noble, Pastiche: making backup cheap and easy, in: USENIX OSDI, 2002.
[10] M. Landers, H. Zhang, K.-L. Tan, Peerstore: better performance by relaxing in peer-to-peer backup, in: IEEE P2P, 2004.
[11] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, M. Isard, A cooperative internet backup scheme, in: USENIX ATC, 2003.
[12] P. Skowron, K. Rzadca, Replica Placement for P2P Redundant Data Storage on Unreliable, Non-dedicated Machines, CoRR abs/1212.0427.
[13] K. Tangwongsan, H. Pucha, D.G. Andersen, M. Kaminsky, Efficient similarity estimation for systems exploiting data redundancy, in: IEEE INFOCOM, 2010.
[14] F. Guo, P. Efstathopoulos, Building a high-performance deduplication system, in: USENIX Annual Technical Conference, 2011.
[15] A.G. Dimakis, P.B. Godfrey, M.J. Wainwright, K. Ramchandran, Network coding for distributed storage systems, in: IEEE INFOCOM, 2007.
[16] A. Duminuco, E. Biersack, Hierarchical codes: how to make erasure codes attractive for peer-to-peer storage systems, in: IEEE P2P, 2008.
[17] M. Dell'Amico, P. Michiardi, Y. Roudier, Password strength: an empirical analysis, in: 2010 Proceedings IEEE INFOCOM, IEEE, 2010, pp. 1–9.
[18] J. Bonneau, The science of guessing: analyzing an anonymized corpus of 70 million passwords, in: IEEE Symposium on Security and Privacy (SP), 2012, IEEE, 2012, pp. 538–552.
[19] P.G. Kelley, S. Komanduri, M.L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L.F. Cranor, J. Lopez, Guess again (and again and again): measuring password strength by simulating password-cracking algorithms, in: IEEE Symposium on Security and Privacy (SP), 2012, IEEE, 2012, pp. 523–537.
[20] M. Abadi, T.M.A. Lomas, R. Needham, Strengthening Passwords, Digital System Research Center, Tech. Rep 33, 1997, 1997.
[21] L. Toka, M. Dell'Amico, P. Michiardi, Data transfer scheduling for P2P storage, in: IEEE P2P, 2011.
[22] M. Dell'Amico, M. Filippone, P. Michiardi, Y. Roudier, On user availability prediction and network applications, IEEE/ACM Trans. Netw. (2014).
[23] N. Oualha, M. Önen, Y. Roudier, A security protocol for self-organizing data storage, in: IFIP SEC, 2008.
[24] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, D. Song, Remote data checking using provable data possession, ACM Trans. Inform. Syst. Secur. 14 (1) (2011) 12:1–12:34.
[25] L.P. Cox, B.D. Noble, Samsara: honor among thieves in peer-to-peer storage, ACM SIGOPS Operating Systems Review, vol. 37, Springer, 2003, pp. 120–132.
[26] N. Oualha, P. Michiardi, Y. Roudier, A game theoretic model of a protocol for data possession verification, in: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2007, WoWMoM 2007, IEEE, 2007, pp. 1–6.
[27] L. Pamies-Juarez, P. García-López, M. Sánchez-Artigas, Rewarding stability in peer-to-peer backup systems, in: IEEE ICON, 2008.
[28] P. Michiardi, L. Toka, Selfish neighbor selection in peer-to-peer backup and storage applications, in: Euro-Par, 2009.
[29] V. Vishnumurthy, S. Chandrakumar, E. Sirer, Karma: a secure economic framework for peer-to-peer resource sharing, in: P2P Econ, 2003.
[30] S. Nakamoto, Bitcoin: A Peer-to-peer Electronic Cash System. <http://nakamotoinstitute.org/bitcoin/>.
[31] S. Seuken, D. Charles, M. Chickering, S. Puri, Market design & analysis for a P2P backup system, in: Proceedings of the 11th ACM Conference on Electronic Commerce, EC '10, ACM, New York, NY, USA, 2010, pp. 97–108.
[32] D.N. Tran, F. Chiang, J. Li, Friendstore: cooperative online backup using trusted nodes, in: Proceedings of the 1st Workshop on Social Network Systems, ACM, 2008, pp. 37–42.
[33] R. Sharma, A. Datta, M. Dell'Amico, P. Michiardi, An empirical study of availability in friend-to-friend storage systems, in: 2011 IEEE International Conference on Peer-to-Peer Computing (P2P), IEEE, 2011, pp. 348–351.
[34] S. Marti, H. Garcia-Molina, Taxonomy of trust: categorizing p2p reputation systems, Comput. Netw. 50 (4) (2006) 472–484.
[35] L. Toka, M. Dell'Amico, P. Michiardi, Online data backup: a peer-assisted approach, in: IEEE P2P, 2010.
[36] A. Haeberlen, A. Mislove, P. Druschel, Glacier: highly durable, decentralized storage despite massive correlated failures, in: USENIX NSDI, 2005.
[37] T. Mager, E. Biersack, P. Michiardi, A measurement study of the Wuala on-line storage service, in: IEEE 12th International Conference on Peer-to-Peer Computing (P2P), 2012, 2012, pp. 237–248.
[38] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M.F. Kaashoek, J. Kubiatowicz, R. Morris, Efficient replica maintenance for distributed storage systems, in: USENIX NSDI, 2006.
[39] L. Pamies-Juarez, P. Garcia-Lopez, Maintaining data reliability without availability in p2p storage systems, in: ACM SAC, 2010.
[40] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, Oceanstore: an architecture for global-scale persistent storage, in: ACM ASPLOS, 2000.
[41] R. Bhagwan, K. Tati, Y. Chung Cheng, S. Savage, G.M. Voelker, Total recall: system support for automated availability management, in: USENIX NSDI, 2004.
[42] L. Pamies-Juarez, P. Garcia-Lopez, M. Sánchez-Artigas, Availability and redundancy in harmony: measuring retrieval times in p2p storage systems, in: IEEE P2P, 2010.
[43] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, R. Wattenhofer, FARSITE: federated,

available, and reliable storage for an incompletely trusted environment, in: ACM SIGOPS Operating Systems Review, vol. 36.
[44] A. Duminuco, E. Biersack, T. En-Najjary, Proactive replication in distributed storage systems using machine availability estimation, in: ACM CoNEXT, 2007.
[45] B. Schroeder, G.A. Gibson, Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you? in: USENIX FAST, 2007.
[46] L. Rizzo, Effective erasure codes for reliable computer communication protocols, in: ACM SIGCOMM Computer Communication Review, vol. 27(2), 1997, pp. 24–36.
[47] S.B. Wicker, V.K. Bhargava, Reed–Solomon Codes and Their Applications, John Wiley & Sons, 1999.
[48] J. Byers, M. Luby, M. Mitzenmacher, A digital fountain approach to asynchronous reliable multicast, IEEE J. Sel. Areas Commun. 20 (8) (2002) 1528–1540.
[49] J. Byers, J. Considine, M. Mitzenmacher, S. Rost, Informed content delivery across adaptive overlay networks, ACM SIGCOMM Computer Communication Review, vol. 32, ACM, 2002.
[50] A. Dimakis, V. Prabhakaran, K. Ramchandran, Distributed fountain codes for networked storage, in: IEEE ICASP, 2006.
[51] P. Cataldi, A. Tomatis, G. Grilli, M. Gerla, CORP: cooperative rateless code protocol for vehicular content dissemination, in: IFIP Med-Hoc-Net, 2009.
[52] M. Bogino, P. Cataldi, M. Grangetto, E. Magli, G. Olmo, Sliding-window digital fountain codes for streaming of multimedia contents, in: IEEE ISCAS, 2007.
[53] P. Cataldi, M. Grangetto, T. Tillo, E. Magli, G. Olmo, Sliding-window raptor codes for efficient scalable wireless video broadcasting with unequal loss protection, IEEE TIP 19 (6) (2010) 1491–1503.
[54] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, A. Venkataramani, Do incentives build robustness in BitTorrent? in: USENIX NSDI, 2007.
[55] A. Shokrollahi, Raptor codes, IEEE Trans. Inform. Theory 52 (6) (2006) 2551–2567.
[56] R. Gracia-Tinedo, M. Sanchez Artigas, P. Garda Lopez, Analysis of data availability in f2f storage systems: when correlations matter, in: 2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P), IEEE, 2012, pp. 225–236.

Matteo Dell'Amico is a researcher at EURECOM; his research touches both theoretical and practical topics of distributed computing. Matteo received his Ph.D. in Computer Science from the University of Genoa (Italy); he also worked at University College London. His research interests include data-intensive scalable computing, peer-to-peer systems, recommender systems, social networks, and computer security.

Pietro Michiardi received his M.S. in Computer Science from EURECOM and his M.S. in Electrical Engineering from Politecnico di Torino. Pietro received his Ph.D. in Computer Science from Telecom ParisTech (former ENST, Paris), and his HDR (Habilitation) from UNSA. Today, Pietro is an Assistant professor of Computer Science at EURECOM, where he leads the Distributed System Group, which blends theory and system research focusing on large-scale distributed systems (including data processing and data storage), and scalable algorithm design to mine massive amounts of data. Additional research interests are on system, algorithmic, and performance evaluation aspects of computer networks and distributed systems.

Laszlo Toka is currently an assistant professor at Budapest University of Technology and Economics and his work focuses on the area of big data analytics in mobile telecommunications. He is coauthor of several published and pending patents.
He received his PhD from Telecom Paris in 2011 before which he worked on economic modeling of distributed computer systems applying game theory and matching theory, on designing incentive schemes in peer-to-peer networks and on building dynamic radio spectrum allocation frameworks.
Based on his academic works in these topics he coauthored several papers e.g., in Elsevier's Computer Networks and Computer Communications.

Pasquale Cataldi earned his doctorate degree from Politecnico di Torino in 2009 working on applications of rateless codes for wireless networks. His research on multimedia transmission, vehicular networks and routing continued as visiting researcher at UCLA and as postdoctoral fellow at EURECOM. While at British Sky Broadcasting, he performed strategic analysis of state-of-the-art and innovative technologies and solutions to be used in the development of Sky products and provided insights on wireless technologies to other parties within the business. He is currently Researcher at Nominet investigating technologies and generating new ideas, taking them through modeling, proof-of-concept and prototyping.