Analysis of Data Availability in F2F Storage Systems: When Correlations Matter

Raúl Gracia-Tinedo, Marc Sánchez-Artigas and Pedro García-López
Universitat Rovira i Virgili, Tarragona (Spain)
{raul.gracia|marc.sanchez|pedro.garcia}@urv.cat

***Abstract*—Nowadays, the growing necessity for secure and private off-site storage motivates the appearance of novel storage infrastructures. In this sense, it is increasingly common to find storage systems where users interact just with a set of trustworthy participants, such as in Friend-to-Friend (F2F) networks.**

In general, these systems have been treated as a particular case of P2P networks where links among nodes are built upon social relationships. However, F2F systems are highly affected by *availability correlations* and *very small friendsets*. The combination of these phenomena poses new challenges which remain unsolved for providing an adequate storage service to users.

In this work, we examine the role that correlated availabilities play in the storage service provided by a small group of friends. We extensively analyze the performance of traditional approaches used to calculate data availability and their application in the estimation of data redundancy. Moreover, we propose a novel technique to calculate the data redundancy needed based on the historical availability of a group of friends.

Our study reveals important insights: i) Availability correlations in a small group of friends can provide a good trade-off between data availability and data redundancy. ii) Traditional redundancy calculation algorithms may importantly misestimate the data redundancy needed due to correlations. iii) Finally, our history-based algorithm exhibits high accuracy in the estimation of redundancy, providing important benefits to the system.

***Keywords*-friend-to-friend (F2F), storage systems, data availability, availability correlations**

I. Introduction

In the last decade, we have witnessed an intense proliferation of storage services, ranging from pure Cloud systems (e.g. Amazon S3, Dropbox) to completely decentralized approaches. Nowadays, many users can satisfy their demand to handle their digital lives, including photos, media files, and work documents. However, as the basic storage requirements are met, new ones arise (e.g. privacy, trust, cost), which motivates researchers to design novel storage infrastructures.

In this sense, Friend-to-Friend (F2F) storage systems are currently an interesting research topic and they constitute an alternative approach to leverage personal storage [1], [2], [3], [4], [5]. The F2F paradigm is based on the synergy between social networks and storage systems: users store their data in a set of trustworthy friends. Thus, data is neither stored in a centralized server nor in unknown peers, enabling users to retain the *control of their data*. Moreover, the social component of F2F systems alleviates many undesirable problems present in large-scale storage systems —e.g. security, trust, incentives.

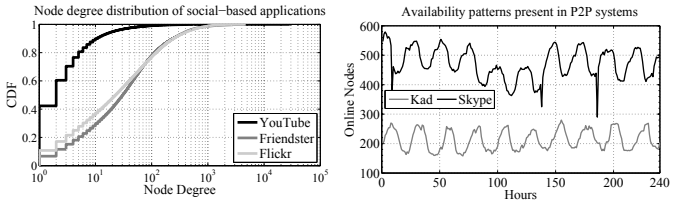
The applicability of F2F systems is currently large and growing: for instance, decentralized online social networks (DOSNs), such as SuperNova [6], propose to store social data of users in their friends. In this respect, the emerging SocialCloud paradigm [7] proposes a "scalable computing model in which virtualized resources contributed by users are dynamically provisioned by a group of friends" [2]. Furthermore, F2F storage also includes commercial applications like CrashPlan[1], which allow to back up data to friends.

Generally, F2F storage systems have been treated as a particular case of P2P systems where nodes are connected by social relationships [1], [3]. However, very little attention has been paid to the characterization of these systems. Understanding the characteristics of F2F systems is crucial for providing an adequate storage service to users (Fig. 1).

We consider two main aspects which clearly differentiate F2F systems from traditional P2P systems:

High Availability Correlations: Availability correlation can be understood as the high probability that given an ON (OFF) user, his friends are ON (OFF) as well. Indeed, measurements of online social networks have shown that friends present significant correlation in their activity patterns [8]—in line with other popular P2P applications [9], [10]. This implies that it is probable to find all friends of a user simultaneously offline (e.g. night) which makes it impossible to maintain high data availability even when placing one replica at each friend [11]. Fig. 1b shows the presence of correlations in real P2P systems.

Extremely Small Friendsets: Users in a F2F storage system are likely to hold a *reduced* number of trustable friends. To wit, over 63% of Facebook users have less than 100 friends [12], and what is even worse, most of their interactions occur only across a reduced subset of their social links. Concretely, 20% of their friends account for 70% of all interactions [12]. As we can see in Fig. 1a, other social-based applications present even lower connectivity among users.



(a) Degree distribution of social-based(b) Availability patterns of users in real content distribution applications [13]. P2P systems (10 days, 1000 nodes).

Fig. 1. Characterization of F2F storage systems.

The combination of these issues poses new challenges which remain unsolved in a F2F scenario. In this paper, we focus on the problem of *data availability in F2F systems*.

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

Our contributions in this work are summarized as follows:

- We analyze the impact of availability correlations on the data availability provided by a small group of friends. Contrary to conventional wisdom, we found that *correlations can be exploited to achieve an adequate trade-off between data availability and redundancy*.
- We evaluate the performance of common approaches of estimating data availability when users are correlated. We demonstrate that these techniques are *severely biased and this impacts on the data redundancy calculation*.
- We propose a history-based method to calculate data availability tailored to *heterogeneous and correlated availabilities*. Making use of it, we can precisely calculate data redundancy required for a user, providing important benefits to the storage service.
- Finally, we explore the relationship between data availability and download times. Our results suggest that, due to availability patterns, we should *distinguish between if a file is currently available and if it is retrievable in a reasonable amount of time*.

The rest of the paper is structured as follows. We discuss the related work in Section II, and describe preliminary concepts in Section III. We present our historical data availability and redundancy calculation algorithms in Section IV. In Section V we present the simulation framework, followed by the study of data availability in Section VI. We conclude in Section VII.

II. Related Work

Peer-to-Peer. In the last decade, extensive research in Peer-to-Peer (P2P) has been devoted to build decentralized storage infrastructures using users' resources [14], [15], [16]. However, P2P technologies face important drawbacks that limit their adoption in real off-site storage systems. First, the *unstability* and *heterogeneity* of peers [17] makes difficult and costly to provide high data availability. Furthermore, despite important efforts [18], the existence of *free riding* and selfish behaviors complicates the efficient management of the existing resources. Moreover, many users are still reluctant to store their data in unknown hosts due to *trust and security* reasons.

Friend-to-Friend. F2F storage systems originally emerged as an alternative to overcome many of the limitations of P2P storage systems, such as *free-riding* and the *lack of trust*. In fact, in the pioneering work of J. Li et. al. [3], the authors argued that a user should choose its neighbors (the nodes with which it shares data) based on existing *social links* instead of *randomly*. Such an approach provided incentives for users to cooperate and resulted in a more stable system which, in turn, reduced the cost of maintaining data.

Friendstore [1] looked at how to ensure *availability* and *durability* in a social backup system by storing data on trusted nodes only, and hence discouraging free-riding. However, Friendstore evaluation was conducted by using availability traces of corporate desktop machines, which did not account for *availability correlations* existing in Internet systems.

Very recently, our research in [4], [5] focused on improving the storage QoS of a F2F system with a Cloud backend.

Availability Correlations. Availability correlation is a well-known issue in large-scale distributed systems [10], [9], even for small groups [19]. For instance, [20] argues that the average user availability may be misleading when it is used to calculate data redundancy in the presence of availability correlations. Also, authors in [21] introduced new metrics to quantify the degree of online-offline correlation of nodes. They applied these metrics to improve the performance of task scheduling and file storage applications.

In line with our solution, in the presence of peer heterogeneity and availability patterns, Kermarrec et. al. proposed very recently [22] to resort to historical node availabilities to improve data placement and repairs. However, this paper differs from [22] in several aspects: i) Contrary to a P2P scenario, the number of available nodes in a F2F system is extremely reduced. ii) In [22] nodes are selected by their anti-correlation to store data. This is not the case for a F2F system where nodes cannot be selected from a large set: they are restricted to a user's *trustworthy* social links.

Data Availability. Important research efforts have been focused on providing an adequate replica placement to guarantee data availability [23], [24]. Instead of replicas, authors in [25] proposed to store data blocks to nodes proportional to their availability. They also provide a numerical method to estimate data availability in a system with heterogeneous nodes.

The only work we are aware of that studied data availability in F2F storage systems is [11], which showed that F2F systems cannot guarantee high data availability: if no friends are online, then the data stored in the system will not be accessible by any means. However, this work did not consider the correlated dynamics of friends and how this affects to the storage service.

This is the first work to explore the impact of the inherent characteristics of F2F systems —reduced friendsets, availability patterns– on data availability and their consequences to the storage service.

III. System Model and Background

For a node f, we denote by $\mathcal{F}$ the set of friends at which f wants to store data. We assume this set is built up by leveraging upon real-trust between users, for example, in an online social network (OSN) like *Facebook* or *Orkut*. Since our focus is on home users, we assume that node f has *limited* download and upload bandwidth, denoted by d_f and μ_f, respectively. We also *limit* the number of parallel connections to P_d and to P_μ for downloads and uploads, respectively. The storage capacity at node f is denoted by s_f.

Friends alternate between *online* (ON) and *offline* (OFF) states. In addition, their online sessions may be correlated over time. Correlation can be understood as the high probability that given an online user, his friends are online as well, which corresponds well with the strong diurnal pattern empirically observed in OSNs like *Facebook* [8].

To capture availability correlations, we distinguish between availability correlation for online sessions and correlations for offline sessions. As we will see next, this separation provisions us with valuable information about the impact of correlations on data availability that otherwise would remain hidden.

Technically, to represent the availability of a host f, we use a vector AV_f of size T, where its ith position $AV_v[i] = 1$ if f was ON at time t_i, or 0 otherwise, where $t_i = \Delta \cdot i$, $\Delta > 0$. Δ represents the length of a time slot. With this representation, we can measure both types of correlations adapting the metrics in [21] into Definition 1:

Definition 1. The *Presence Matching* (PM) metric measures the level of coincidence of the ON sessions of two nodes a, b:

$$PM_{a,b} = \frac{|AV_a[t] \cap AV_b[t]|}{|AV_a[t] \cup AV_b[t]|}, \forall t \in T \text{ where } AV_i[t] = 1, i \in \{a, b\}$$

Analogously, the *Disconnection Matching* (DM) metric gives the same information about OFF durations ($AV_i[t] = 0$).

We extend the above two metrics by calculating the average pairwise PM and DM measures within a group $\mathcal{F}$. We refer to these metrics by *Group Presence Matching* (GPM) and *Group Disconnection Matching* (GDM), respectively. Given a group $\mathcal{F}$, we calculate the GPM over $\mathcal{F}$ as follows:

$$GPM(\mathcal{F}) = \frac{\sum_{i,j \in \mathcal{F}, i \neq j} PM_{i,j}}{\sum_{i=1}^{|\mathcal{F}|-1} i} \tag{1}$$

The calculation of GDM is analogous and has been omitted due to space constraints.

A. Data Availability

As the number of friends is small and they can be temporarily offline, F2F storage systems provide data availability by means of redundancy.

To assure a given level of availability, our F2F system makes use of *Erasure Codes* (ECs), which has been proven to be more efficient in terms of redundancy than replication [26]. An EC scheme splits an input file into k fragments of $1/k$th the size of the original file. Then, these k fragments are encoded into n redundant blocks k, $k \leq n$, which are stored at different nodes to mask failures. The *data redundancy* required to store a file is thus $\frac{n}{k}$. The original file can then be recovered by collecting any subset of k blocks out of the total n.

Blocks generated during the encoding process are assigned to the friendset members. This assignment depends on a *data placement policy*. We denote by $b_f \in \{0, 1, \ldots, n\}$ the number of blocks assigned to a friend $f \in \mathcal{F}$. An assignment is represented as a vector $\vec{\mathbf{b}} = (b_1, \ldots, b_{|\mathcal{F}|})$, where the ith position is the number of blocks b_i stored at the ith friend.

Traditionally, given the number of fragments for a file k and the target level of data availability A, the number of encoded blocks to upload n and hence, the redundancy rate $\frac{n}{k}$, has been determined as follows:

$$A = \sum_{i=k}^{n} \binom{n}{i} \bar{a}^i (1 - \bar{a})^{n-i}, \tag{2}$$

where $\bar{a}$ is the average host availability of a group $\mathcal{F}$. (2) simply accounts for all the possible combinations of finding $\geq k$ blocks out of n, and the probability that this happens.

Two important observations must be discussed here about (2). The first is that this equation assumes that each fragment is stored at a distinct machine, because otherwise the failure of a single host would imply the loss of multiple fragments, thereby leading to an underestimation of the real data availability given by (2). This assumption is not realistic in our case. Due to the reduced number of friends (typically, between 5 and 20), it is very likely that a friend gets assigned more than one fragment.

Second, by employing the mean node availability $\bar{a}$, the *binomial approximation* (Eq. 2) does not take into account the heterogeneity of node availabilities within a group. This could potentially introduce estimation errors if we consider heterogeneous friendsets.

Recently, an *heterogeneity-aware calculation* has been proposed to calculate data availability with higher accuracy [25].

Considering a set of friends $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$, the *combinadic* $C^{|\mathcal{F}|,i}$ is the lexicographically ordered list with all the $\binom{|\mathcal{F}|}{i}$ possible combinations of i friends. In order to lexicographically sort this list, we consider the nodes in each combination sorted in ascending order using their subindexes. By abuse of notation we denote as $C_j^{|\mathcal{F}|,i}$ the j^{th} element of $C^{|\mathcal{F}|,i}$. For example, for a $|\mathcal{F}| = 3$: $C^{|\mathcal{F}|,2} = [[f_1, f_2], [f_1, f_3], [f_2, f_3]]$ and $C_2^{|\mathcal{F}|,2} = [f_1, f_3]$.

Once addressed the concept of combinadic, we have that:

$$A = \sum_{i=1}^{|\mathcal{F}|} \sum_{j=1}^{\binom{|\mathcal{F}|}{i}} \left[\gamma(k, C_j^{|\mathcal{F}|,i}, \vec{\mathbf{b}}) \prod_{f \in C_j^{|\mathcal{F}|,i}} a(f) \prod_{f \in \mathcal{F} \setminus C_j^{|\mathcal{F}|,i}} (1 - a(f)) \right], \tag{3}$$

where the γ function selects which combinations of storage nodes are storing together at least k data blocks, since they could store distinct amounts of data:

$$\gamma(k, \mathcal{L}, \vec{\mathbf{b}}) = \begin{cases} 1 & \left(\sum_{i \in \mathcal{L}} b_i\right) \geq k \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The heterogeneity-aware approach provides a remarkably more precise notion of data availability than the binomial approximation. Moreover, Eq. 3 avoids restrictive assumptions which are not necessarily present in real systems (e.g. every block must be stored in a distinct node).

However, both Eq. 2 and Eq. 3 assume that hosts are not correlated, which is by far not true in F2F systems. As before, this implies that these approaches can highly underestimate or overlook the real data availability. This motivates us to present a more accurate approach for representing node availabilities, and even, the collective dynamics of a friendset.

IV. Historical Data Availability & Redundancy

In general, the majority of real-world systems express node availabilities with simple averages of their past behavior. With this information, it is simple to estimate the data availability provided by storage nodes with Eq. 2 or Eq. 3[2].

The relevance of heterogeneity and availability patterns reported in social networks [8] and many P2P systems [10], [9] poses an important evidence: *availabilities cannot be accurately expressed by averaging the fraction of time nodes have been online*. Furthermore, in a F2F scenario, the reduced number of storage nodes makes availability correlations to be even more important [19]. Such a simplification completely hides the correlated dynamics of nodes, which may, in turn, produce considerable problems to the storage service; for instance, significant errors in *the estimation of data availability*.

In our system, we calculate the data availability provided by a friendset $\mathcal{F}$ in a window of T time units as follows:

[2]Other theoretical models to describe node dynamics, such as Markov chains, have reported limited applicability in practice [22], [27].

$$A = \frac{1}{|T|} \sum_{t=0}^{T} \alpha(k, \mathcal{F}, t, \vec{\mathbf{b}}), \forall t \in T \quad (5)$$

where $\alpha(k, \mathcal{F}, t, \vec{\mathbf{b}})$ is an indicator function which evaluates whether a file is available at instant t as follows:

$$\alpha(k, \mathcal{F}, t, \vec{\mathbf{b}}) = \begin{cases} 1 & \left(\sum_{i \in \mathcal{F}} AV_i[t] \cdot b_i\right) \geq k \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Therefore, we calculate the data availability provided by a group of friends based on their historical behavior. Note that the value of A in Eq. 5 is basically the aggregation of the timeslots of instant data availability, that is, those periods where the number of available blocks is greater than k.

This approach provides an exact notion of the data availability provided by a set of friends in a past window of time, even in the presence of high heterogeneity and availability patterns. *Our objective is to benefit from this mechanism to calculate the precise amount of data redundancy (n/k) to be stored in $\mathcal{F}$ for achieving a targeted data availability (δ).*

A. Historical Optimal Data Redundancy: Complexity

Ideally, the amount of redundancy used should be *minimal to optimize storage space* provided that the target level of *data availability* is met. Moreover, since friendsets are normally small, we should *maintain load balancing for providing fairness and reliability* in such a limited system.

This problem can be defined as follows (Definition 2):

Definition 2. Given a fixed k and a friendset $\mathcal{F}$, our objective is to find the minimal n, $n \geq k$, that achieves a targeted data availability δ, where each friend $f \in \mathcal{F}$ stores a number of blocks $b_f, b_{min} \leq b_f \leq b_{max}$, where constants $b_{min}, b_{max} \in \{0, \ldots, n\}$.

Solving this problem requires to *examine all the possible block assignments for each value of* n. The reason is that the resulting data availability depends on the availability history of friends, for which no assumption can be made on its exact behavior. Further, it can be easily seen that the optimal block assignment with $n+1$ blocks will never provide less data availability than the optimal one with n blocks. Consequently, we will be able to use binary search to optimize the search in n (Algorithm 1).

The most computationally expensive part of Algorithm 1 lies on the function $maxDA$. This function looks for the block assignment that maximizes data availability under the established redundancy and load balancing constraints. Intuitively, we confront a combinatorial optimization problem. To illustrate its complexity, we formalize the $maxDA$ function as a *bounded knapsack problem* [28].

Algorithm 1: Historical Optimal Data Redundancy

```
Input: F, δ, k, b_min, b_max
Output: n
n ← k, t ← |F| · k;
while n ≠ t do
    m = (n+t)/2;
    if maxDA(F, k, m, b_min, b_max) < δ then
        n ← m + 1;
    else
        t ← m;
    end
end
```

In our formalization, we assume that each friend has a *weight* b_f, that is, the number of blocks it stores. Moreover, to fit our problem into the formal knapsack definition, each friend f has a specific value function v_f defined as follows:

$$v_f(\vec{\mathbf{b}}) = \frac{1}{|T|} \sum_{t=0}^{T} AV_f[t] \cdot \tau(k, \mathcal{F}, t, \vec{\mathbf{b}}) \quad (7)$$

$$\tau(k, \mathcal{F}, t, \vec{\mathbf{b}}) = \begin{cases} \frac{1}{\rho(\mathcal{F}, t, \vec{\mathbf{b}})} & \alpha(k, \mathcal{F}, t, \vec{\mathbf{b}}) = 1 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

where the function $\rho(\mathcal{F}, t, \vec{\mathbf{b}}) = \sum_{f \in \mathcal{F}} AV_f[t] \cdot b_i$ represents the total number of available blocks at instant t. Therefore, v_f expresses the relative contribution of a friend f to the time periods where the file is available. Then, the optimization problem is:

$$\text{maximize} \quad V = \sum_{f \in \mathcal{F}} v_f(\vec{\mathbf{b}}) \cdot b_f, b_{min} \leq b_f \leq b_{max} \quad (9)$$

$$\text{subject to} \quad \sum_{f \in \mathcal{F}} b_f \leq n \quad (10)$$

The additional complexity of this problem w.r.t. the classical knapsack problem is that coefficients (v_f, b_f) depend on the assignment $\vec{\mathbf{b}}$, and therefore, they should be recalculated at each execution step. Moreover, one can infer that depending on the assignment of data blocks within a friendset $\mathcal{F}$, the resulting data availability will vary. This means that distinct assignments of equivalent redundancy and load balancing constraints may lead to different V values due to the collective dynamics of nodes. Thus, the complexity of our problem extends beyond the classical non-linear bounded knapsack problem, which is known to be NP hard [28].

For this reason, we propose an heuristic method to take advantage of the historical information in an efficient manner.

B. Estimating Data Redundancy with the History of Friends

As aforementioned, availability correlation in conjunction with a small friendset makes it hard to maintain a high data availability during the 24 hours of the day. For this reason, we strive to ensure a high data availability during the period of the day where friends are mostly online, what we call as *daily data availability*. This new perspective of data availability specifically *benefits from correlations* to provide an adjusted and scalable storage service. More specifically, our aim is to assure a high data availability during at least δ hours per day, instead of at all times as in traditional storage systems, where a sufficient number of uncorrelated nodes can be found.

The algorithm for this computation works as follows. The initial number of blocks to be transferred to the friendset is $n = k$. The algorithm then assigns the n blocks to the friends in a *round-robin* style in order to balance storage costs. Using

Algorithm 2: Historical data redundancy calculation

Input: $\mathcal{F}$, δ, k
Output: n
$n \leftarrow k-1$;
$h_{\text{avail}} \leftarrow 0$
while $h_{avail} \leq \delta$ ***and*** $\frac{n}{k} < |\mathcal{F}|$ **do**
  $n = n+1$;
  $n_{\text{timeslot}} \leftarrow 0$;
  for i ***in*** w **do**
    $n_{\text{avail}} \leftarrow 0$;
    for f ***in*** $\mathcal{F}$ **do**
      if $AV_f[i] \equiv 1$ **then**
        $n_{\text{avail}} = n_{\text{avail}} + n/|\mathcal{F}|$;
      end
    end
    if $n_{avail} \geq k$ **then**
      $n_{\text{timeslot}} = n_{\text{timeslot}} + 1$;
    end
  end
  $h_{\text{avail}} \leftarrow n_{\text{timeslot}} \cdot \Delta$
end

a past time window of w timeslots, it computes the number of timeslots $w_{timeslot}$ within the window where the number of available blocks n_{avail} is equal or greater than k. Note that at least few days should be considered in the time window w. If $w_{timeslot}$ times the duration of a timeslot Δ covers δ hours, the algorithm halts, and the value of n is returned. Otherwise, n is incremented by one block and the entire process is repeated (up to $|\mathcal{F}|$ replicas). This procedure is repeated again and again until the value of n guarantees δ hours of data availability. The pseudocode for this algorithm is shown in Algorithm 2.

V. Simulation Framework

In this section we describe the methodology we used to evaluate the behavior of data availability in a F2F system.

A. Setup & Methodology

In our tests, we modeled the alternating ON/OFF behavior of nodes using availability traces. Unfortunately, we could not find real traces of any F2F storage system, simultaneously including social graph and availabilities [11]. Thus, to evaluate the impact of availability correlations we employed real traces from *Kad* [10] and Skype [9]. Additionally, we used synthetically generated [29] traces of an Heterogeneous Yao model[3] [30]. From these traces, we excluded all the nodes whose availability was out of the range $[\bar{a}_{min}, \bar{a}_{max}]$ every 48 hours during the simulation window of $T = 12$ days. This filtering process was necessary to exclude from simulation extreme availability cases (e.g. superpeers, *permanent* churn). As a result, we obtained nodes with regular participation in the system and exhibiting strong correlation (see Fig. 2). Friendsets are formed selecting random nodes from these traces. Note that forming random groups from a strongly correlated trace will *generally* result in correlated groups.

[3]The average online session for friends was of 8 hours while the average downtime was of 16 hours ($\bar{a} = 0.33$). Both ON/OFF time durations were drawn from Pareto distribution (heavy-tailed) with shape parameter $\alpha = 3$, which has been reported to provide a tight fit to the real lifetime distribution found in decentralized systems.

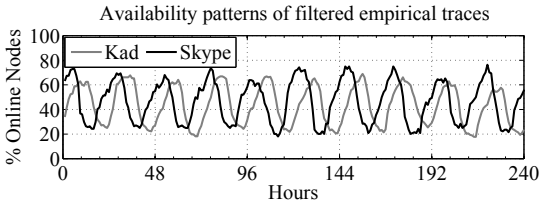


Fig. 2. ON/OFF behavior of nodes in our traces after the filtering process.

N	Filtered nodes	122 (Skype), 192 (Kad) 85 (Yao)
$[\bar{a}_{min}, \bar{a}_{max}]$	Node mean availability	Kad, Skype $[0.2, 0.8]$ Yao $[0.3, 0.4]$
T	Simulation time	12 days
μ_f, d_f	Node up/down bandwidth	30KBps, 120KBps
s_f	Node storage capacity	500GBytes
P_μ, P_d	Parallel up/down connections	1, 4
k	Original file fragments	40
β	Object size	500MB, 1GB, 2GB
$\|\mathcal{F}\|$	Friendset size	$[5, 10, 20]$

TABLE I
System parameters and description.

Our simulations are divided into three different phases, each during 4 days:

1) *Training phase*. In this phase, the users collect historical information in form of availability vector AV about each of their respective storage friends.
2) *Upload phase*. During this phase, the data owner uploads a single file of β bytes, plus the associated redundancy, to the system.
3) *Download phase*. Finally, in the *download phase* a user retrieves k data blocks from the system to reconstruct the uploaded file. During this phase we continuously inspect the *data availability* provided by the group of friends to which the file was uploaded. We measure data availability as the number of time-slots where the number of available blocks is $\geq k$ (Eq. 5).

In our F2F application, retrieving a file requires locating a sufficient number of blocks to perform a decoding operation. Thus, a user needs the network information of their storage friends in the system to initiate the downloading process. We do not restrict the way of storing this information; for instance, it can be stored in a social application or a tracker.

The important simulation parameters used in our simulations are depicted in Table I. Simulation results for each trace correspond to a collection of $1,000$ random friendsets.

B. Data Transfer Scheduling

In our analysis, we evaluate the time needed to download data as a complementary metric to measure data availability. In this sense, we refer to a *schedule* as the set of block transfers concerning the same file. The time needed to transfer a block is called *block transfer time* (BTT). Thus, we refer to the time to complete a schedule simply as the *time to schedule* (TTS).

We refer to as *scheduling policy*, the algorithm that decides the order according to which transfers must occur over time in order to minimize the time to complete a given schedule. We make use of *random policy*: among the remaining block transfers of currently available nodes, the user selects one chosen uniformly at *random* to continue the download process until gathering at least k blocks.

The *optimal time to schedule (OTTS)* refers to the shortest TTS assuming dynamic participation from friends [31]. To compute this time it is necessary to explore all the possible scheduling combinations with the *prior* knowledge of the exact ON/OFF pattern of each storage friend, which is not feasible in practice[4]. We use the download OTTS to measure the impact of different parameters on download times without introducing the bias corresponding to a certain scheduling policy.

Since we do not target backup scenarios, where each storage operation involves several Gigabytes of content, we consider that the data owner stays connected during the whole storage operation, which is common in file-sharing and storage applications. In addition, each friendset serves a single storage request, that is, we do not consider concurrent downloads.

C. Data Placement

A data placement policy is the algorithm which decides the recipient of a data block among a set of candidates. In this work, we evaluate two distinct placements: *round-robin* (RR) and *availability proportional* (AP).

The RR data placement is extremely simple to implement and preserves fairness among friends regarding storage load. In our framework, this placement is used when we estimate the necessary redundancy using the *binomial approximation* (BA) and our *history-based* (HB) algorithm (see Section IV-B).

In [32], authors formally demonstrated that, having a group of nodes with heterogeneous availabilities, assigning an amount of redundancy proportional to their availabilities maximizes the resulting data availability. Hence, we employ the AP data placement in the *heterogeneity-aware* (HA) calculation[5].

We believe that preserving load balancing in a reduced set of participants is essential to provide scalable storage and to avoid service bottlenecks. Our objective is to quantify the impact of data placement on storage fairness and reliability provided by a friendset.

VI. Results

A. Availability Correlations and Data Availability

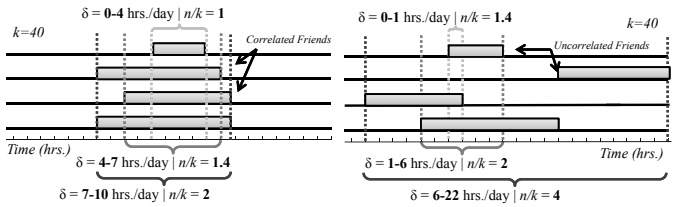
Generally, the presence of availability correlations has been considered in the P2P literature as a flaw which should be avoided to guarantee a high data availability. However, the actual impact of correlated availabilities on the data availability provided by a small group of participants remains unexplored.

To address this issue, we resort to the GPM metric to measure the degree of coincidence among the online sessions of a group of friends. We analyze the resulting data availability when these friends are highly correlated in their online sessions (high GPM) and in the opposite case (low GPM). Both cases are illustrated in Fig. 3. This figure depicts the relationship between data availability and redundancy depending on the GPM of a friendset.

For this analysis, we have synthetically generated 1K friendsets of cardinality 5 for both categories of GPM $\in [0.05, 0.15]$ and $\in [0.3, 0.4]$ from the Yao trace during the download phase.

[4]This time is computed by modeling the scheduling with dynamics friends as a *flow network*, and using binary search to find the *shortest-time max-flow* solution similarly to [31].

[5]We thank Matteo Dell'Amico and Lluís Pàmies for developing a dynamic programming solution to perform this calculation.



(a) Friendset with high GPM. (b) Friendset with low GPM.

Fig. 3. Impact of GPM on the data redundancy (n/k) a friendset requires to provide a certain degree of data availability (δ). Correlated friends provide low/moderate data availability values using less redundancy than uncorrelated ones. However, only uncorrelated friendsets can provide high data availability.

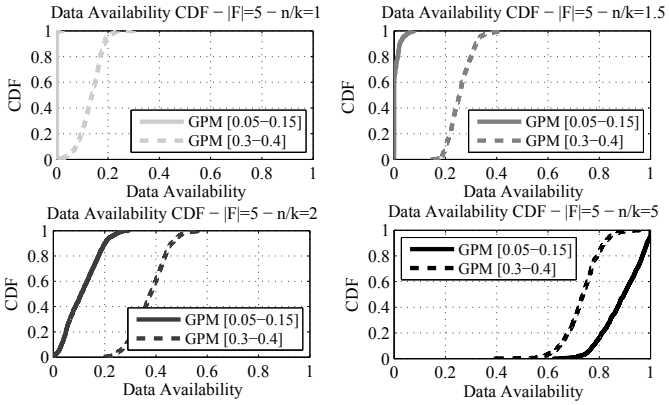
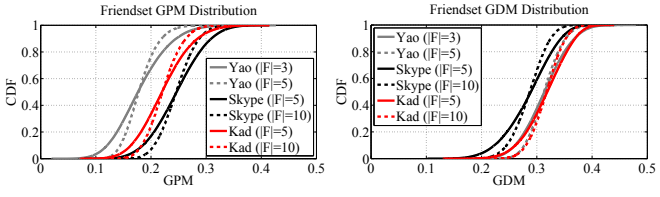


Fig. 4. Evolution of data availability in function of the data redundancy for correlated/uncorrelated friendsets (H. Yao trace, RR placement).

Note that from this trace we excluded all the nodes whose availability was out of the range $[0.3, 0.4]$ every 48 hours during the simulation. This will give us a clear picture of the impact of availability correlations, without the bias induced by high node heterogeneity.

First, in Fig. 4 we observe a clear distinction in the growth of data availability as a function of data redundancy depending on the GPM degree of a friendset. For low to moderate amounts of data redundancy, we see that *availability correlations improve the data availability provided by the friendset*. For instance, when two replicas are introduced in the system ($n/k = 2$), correlated friendsets double the data availability provided compared with the uncorrelated friendsets.

Although this observation may be contrary to conventional wisdom, it is clearly depicted in Fig. 3. In Fig. 3a, we observe that correlated friends provide moderate levels of data availability at a lower storage cost. The reason is simple: if



(a) GPM Distribution. (b) GDM Distribution.

Fig. 5. GPM/GDM distribution of random friendsets of different cardinalities collected from employed traces during the download phase (4 days).

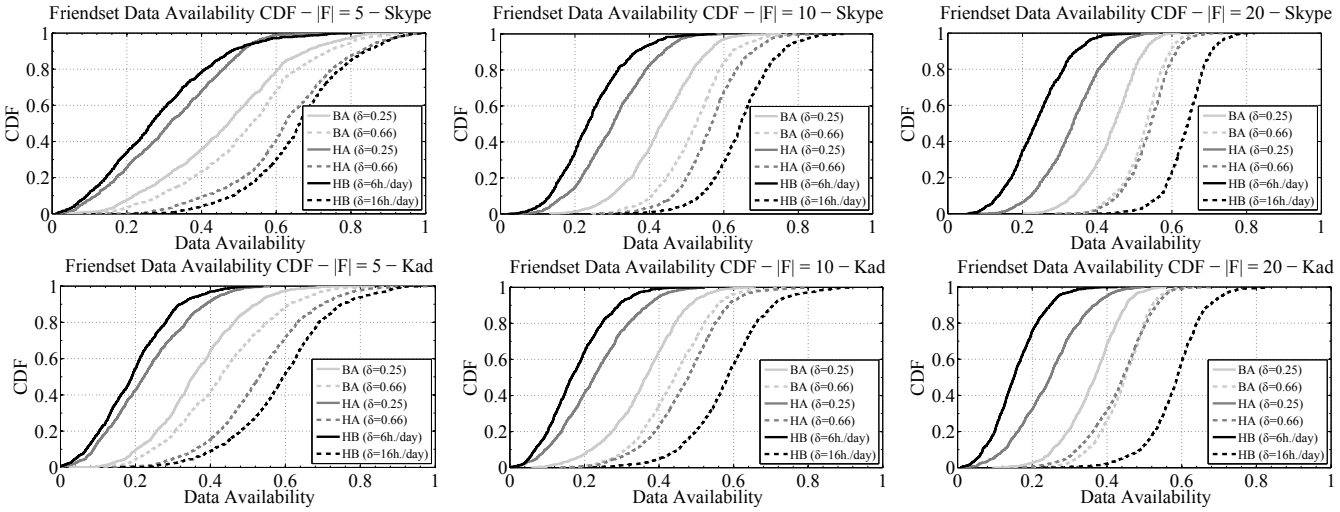


Fig. 6. Data availability obtained by different redundancy calculations approaches varying the expected data availability (δ) and the friendset size ($|\mathcal{F}|$).

friends coincide in their online sessions, the data availability is supported by all friends simultaneously. This reduces the amount of data redundancy needed *at each friend*. On the contrary (Fig. **??**), friendsets exhibiting low GPM have almost no common online periods among them. This implies that they should support large amounts of redundancy per friend (even a replica) to guarantee a high data availability.

However, when we introduce large amounts of data redundancy we observe that this behavior changes. In case of $n/k = 5$, uncorrelated friendsets provide a significantly higher data availability than correlated ones. That is, *for very large amounts of data redundancy, uncorrelated friends are able to cover the majority of the day time with k blocks or more*. On the other hand, we observed that if friends within a group coincide in their online sessions, they also coincide in their offline sessions (high GDM). This fact lead us to an important conclusion: *in a correlated friendset, the maximum data availability achievable is limited by the degree of coincidence in their offline sessions (GDM)*.

Fig. 5 depicts the GPM/GDM distribution of $50,000$ random friendsets selected from availability traces. First, in Fig. 5a we observe that the GPM distribution exhibits a wide range ($[0.1, 0.35]$) in all traces due to the randomness of the node selection. Since there are no real traces of F2F storage systems, we cannot generalize the observed GPM values in our traces to real systems. In our opinion, trusted friends will likely live in the same time-zone, probably exhibiting similar patterns [8]. In that case, the GPM distribution would be remarkably concentrated in high values.

Second, we observe that the GDM distribution of these traces exhibits higher values than GPM distribution. There are two reasons for this phenomenon: i) Low node availabilities make the coincidence of nodes in their offline sessions easier. ii) The nocturnal patterns of real traces increases the probability for a group of friends to be simultaneously offline.

In line with [19], we found that *it is easier to find high availability correlations among participants as the friendset becomes smaller*. This can be appreciated in Fig. 5: for small friendsets ($|\mathcal{F}| = 3$) the GPM/GDM distributions present more extreme values than for larger friendsets ($|\mathcal{F}| = 10$).

TABLE II
MEAN DATA AVAILABILITY DEVIATION ($\mu - \delta$) AND COEFFICIENT OF VARIATION (CV) OF REDUNDANCY CALCULATION TECHNIQUES - SKYPE

$\delta = 0.25 \vert 6$ hours/day	BA ($\vert\mathcal{F}\vert = 5$)	HA ($\vert\mathcal{F}\vert = 5$)	HB ($\vert\mathcal{F}\vert = 5$)
$\mu - \delta$	$+0.21(+84\%)$	$+0.07(+28\%)$	$+0.03(+12\%)$
CV (σ/μ)	0.37	0.46	0.54
$\delta = 0.66 \vert 16$ hours/day	BA ($\vert\mathcal{F}\vert = 5$)	HA ($\vert\mathcal{F}\vert = 5$)	HB ($\vert\mathcal{F}\vert = 5$)
$\mu - \delta$	$-0.14(-21.2\%)$	$-0.04(-6.1\%)$	$0.0(0\%)$
CV (σ/μ)	0.32	0.25	0.21
$\delta = 0.25 \vert 6$ hours/day	BA ($\vert\mathcal{F}\vert = 20$)	HA ($\vert\mathcal{F}\vert = 20$)	HB ($\vert\mathcal{F}\vert = 20$)
$\mu - \delta$	$+0.19(+76\%)$	$+0.08(+32\%)$	$-0.01(-4\%)$
CV (σ/μ)	0.17	0.25	0.34
$\delta = 0.66 \vert 16$ hours/day	BA ($\vert\mathcal{F}\vert = 20$)	HA ($\vert\mathcal{F}\vert = 20$)	HB ($\vert\mathcal{F}\vert = 20$)
$\mu - \delta$	$-0.13(-19.7\%)$	$-0.12(-18.2\%)$	$-0.02(-3\%)$
CV (σ/μ)	0.11	0.12	0.09

In conclusion, in F2F systems, the presence of availability correlations offers a good trade-off between data availability and data redundancy.

B. Data Redundancy Estimation

Next, we study the accuracy of traditional ways to calculate the necessary amount of data redundancy to achieve a certain data availability. To this end, in Fig. 6 we illustrate the data availability experienced by 1K friendsets of distinct cardinalities ($|\mathcal{F}|$) when varying the data availability required (δ).

In Fig. 6, we clearly observe that for low δ, the *binomial approximation* (BA) tends to *greatly overestimate the amount of data redundancy required*. Proof of that is that the system exhibits a much higher data availability than expected. In this sense, the *heterogeneity aware* (HA) calculation significantly improves the accuracy of the BA, thereby demonstrating the importance of considering heterogeneous availabilities [25].

On the other hand, both *BA and HA highly underestimate the necessary amount of redundancy needed for high values of δ, providing lower data availabilities than expected*. This is mainly due to strong availability correlations. Further, this phenomenon seems to be more evident as the friendset size grows since most members exhibit nocturnal patterns (Fig. 6).

We observe that, *our history-based (HB) redundancy calculation is very accurate in the presence of correlations, irrespective of δ and $|\mathcal{F}|$*. In Table II, we illustrate the mean data availability deviation ($\mu - \delta$) provided by each method

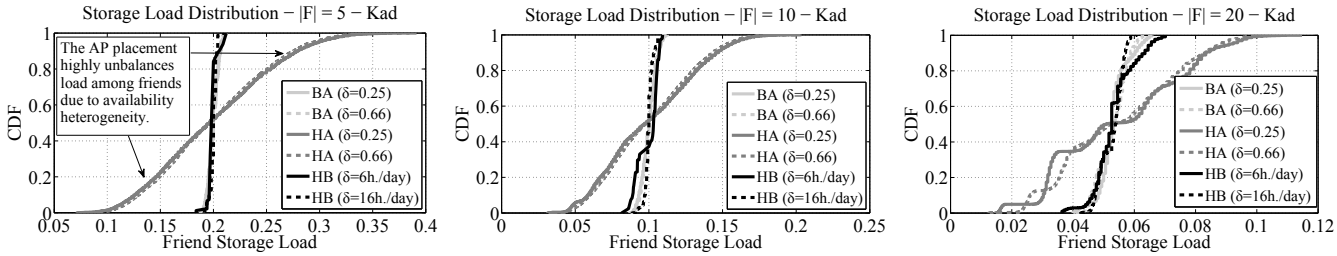


Fig. 7. Storage load balancing supported by nodes. Clearly, the BA and HB redundancy calculation algorithms preserve load balancing due to the use of RR placement whereas the AP placement incurs in high unbalance.

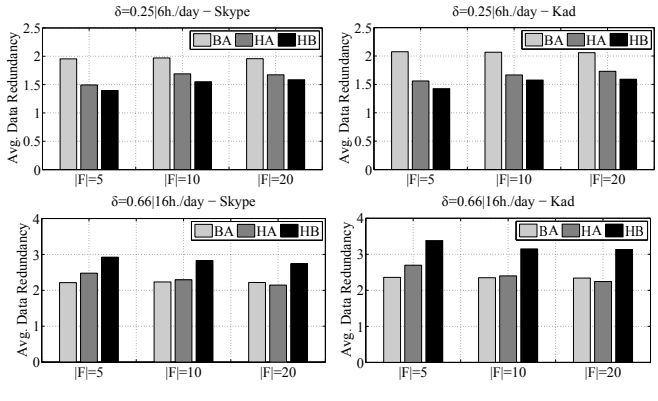


Fig. 8. Average data redundancy factor introduced by BA, HA and HB.

and the resulting coefficient of variation (CV). For instance, for $|\mathcal{F}| = 5$ and $\delta = 0.25$, the HB method has a mean deviation of $+12\%$ from the expected availability, whereas the HA and BA exhibit a deviation of $+28\%$ and $+84\%$ respectively.

In Table II we also infer that the data availability CV is higher for low δ values and small friendsets. This is congruent with results in Section VI-A, where in case of correlated friendsets and low δ, small increments of redundancy result in significant improvements in data availability. This implies that small variations of redundancy induce high variability (CV).

It is worth mentioning that the HB method provides a slightly lower data availability than expected for $\delta = 0.25$ in Kad. By inspecting the Kad trace, we noted that the selected nodes exhibited lower availabilities in the download phase (0.397) than in the training phase (0.449). This induces an underestimation of the necessary redundancy, as the HB algorithm uses the availability vectors of nodes during the training phase. Therefore, the HB method can lead to incorrect redundancy estimations in case of significant variations between the friendset history and the current friendset availability.

The resulting data availability comes from the generated data redundancy. Fig. 8 illustrates the differences in the average data redundancy factor exhibited by the different redundancy calculation approaches. When a user demands high data availability ($\delta = 0.66$), we observe that both BA and HA calculations provide much less redundancy than the HB approach (from -30% to -15%). Further, the deviation of BA and HA calculations techniques causes friendsets to not meet the required data availability (Fig. 6, Table II).

Clearly, for low values of δ both BA and HA introduce more data redundancy than the HB approach. That is, in

TABLE III
FILE RECOVERY PROBABILITY IN THE PRESENCE OF FAILURES

	Friend Random Failures (f_r) - $\|\mathcal{F}\| = 5$ (Kad)		
$n/k = 1.5$	$f_r = 1$	$f_r = 2$	$f_r = 3$
Av. Prop.	100%	99.6%	21.9%
Round Robin	100%	100%	0%
$n/k = 3$	$f_r = 3$	$f_r = 4$	$f_r = 5$
Av. Prop.	100%	85.2%	1.1%
Round Robin	100%	100%	0%
	Most Av. Friend Failures (f_a) - $\|\mathcal{F}\| = 5$ (Kad)		
$n/k = 1.5$	$f_a = 1$	$f_a = 2$	$f_a = 3$
Av. Prop.	100%	97.7%	0%
Round Robin	100%	100%	0%
$n/k = 3$	$f_a = 3$	$f_a = 4$	$f_a = 5$
Av. Prop.	100%	17.5%	0%
Round Robin	100%	100%	0%
	Friend Random Failures (f_r) - $\|\mathcal{F}\| = 10$ (Skype)		
$n/k = 1.5$	$f_r = 3$	$f_r = 4$	$f_r = 5$
Av. Prop.	100%	80.5%	0%
Round Robin	100%	100%	0%
$n/k = 3$	$f_r = 6$	$f_r = 7$	$f_r = 8$
Av. Prop.	100%	93.8%	25.7%
Round Robin	100%	100%	0%
	Most Av. Friend Failures (f_a) - $\|\mathcal{F}\| = 10$ (Skype)		
$n/k = 1.5$	$f_a = 3$	$f_a = 4$	$f_a = 5$
Av. Prop.	100%	0%	0%
Round Robin	100%	100%	0%
$n/k = 3$	$f_a = 6$	$f_a = 7$	$f_a = 8$
Av. Prop.	98.9%	1.4%	0%
Round Robin	100%	100%	0%

Skype for $|\mathcal{F}| = 10$ and $\delta = 0.25$, the BA and HA store 26.97% and 4.13% more redundant data than our proposal, respectively. Furthermore, in that case, this extra redundancy is unnecessary since the BA and HA calculations provide more data availability than expected. In a F2F system, *this burden in terms of unnecessary redundancy may pose important drawbacks (e.g. limited scalability, high uploading times)*.

In conclusion: i) Due to availability correlations, the BA and HA approaches produce significant deviations for calculating the necessary data redundancy for a targeted data availability, and ii) Our HB method is accurate enough to produce an adjusted amount of data redundancy in the presence of correlations. This provides important benefits to the system in terms of data availability and storage scalability.

C. Storage Load Balancing and Reliability

We investigate next the impact of data placement on the load balancing and reliability of the storage service. To this end, we compare two placement policies described in this paper: *round-robin* (RR) and *availability proportional* (AP) policies. As mentioned in Section V, we use the RR placement for the BA and HB redundancy calculation algorithms. The AP placement is inherently used by the HA calculation.

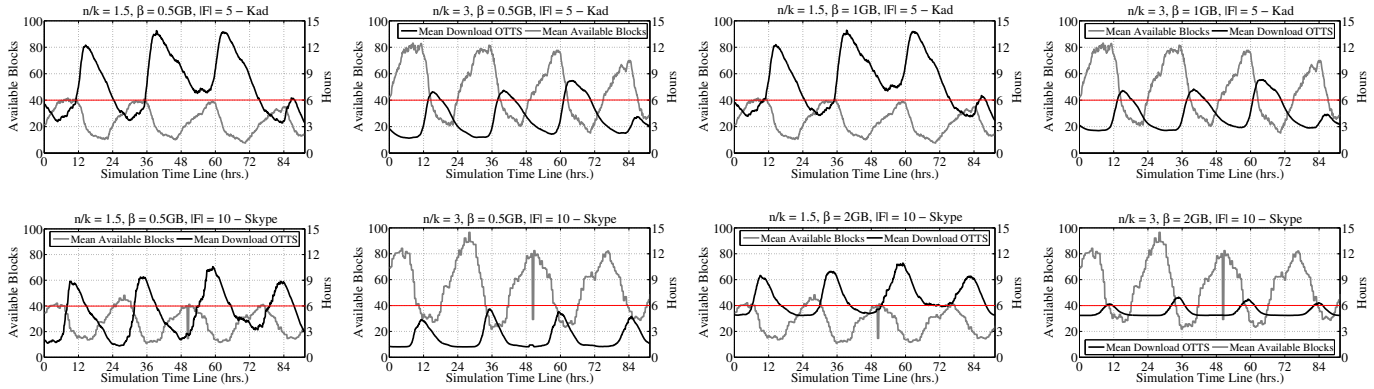


Fig. 9. Relationship between data availability and optimal download times (RR placement).

Fig. 7 depicts the storage load CDF experienced nodes in our simulations. Noticeably, *the AP assignment induces a high storage unbalance. This is specially evident for very small and heterogeneous groups.* To wit, for $|\mathcal{F}| = 5$, we note that the 20% of nodes store less than 15% of a file, whereas a 4% of nodes suffer a load greater than 30%.

Conversely, irrespective of the redundancy calculation and the values of δ and $|\mathcal{F}|$, *the RR placement offers high load balancing*. As expected, as the value of $|\mathcal{F}|$ grows, the differences between both placements become less important.

In our view, load balancing is a key property for such a limited storage system. Poor load balancing may in fact produce severe service problems.

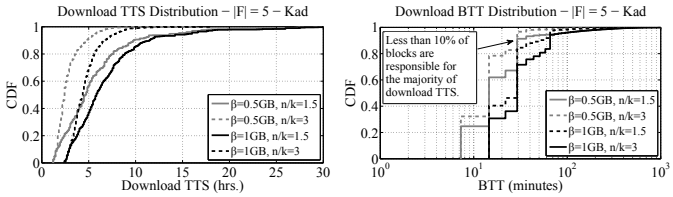
In this sense, Table III presents the file recovery probability in the presence of node failures. In these simulations, we loaded friendsets with a certain amount of data redundancy (n/k) using the AP and RR placements. Moreover, we considered two failure models across a group: random failures and failures occuring to the most available friends.

In general, *random failures within a friendset exhibit a similar impact on both placement strategies*. However, in extreme cases of random failures (rightmost column) we observe that the AP assignment provides better resilience. This is due the fact that random failures can occur to the majority of lowest available (and therefore least loaded) friends, thus providing higher recovery probabilities. In other cases we observe slightly better results from the RR strategy.

Nevertheless, when highest available nodes fail, *the RR placement offers a greater resilience than the AP*. Actually, in the Skype scenario, the RR strategy tolerates one node failure more than the AP placement. This represents a difference of 10% in the recovery probability for the same amount of storage redundancy. Therefore, we conclude that in a F2F scenario it is important to preserve storage load balancing to provide fairness and reliability, as well as to avoid service bottlenecks.

D. Data Availability vs Download Times

Conversely to large-scale systems, providing a certain degree of data availability in a F2F network does not necessarily imply that data can be retrieved in a short period of time. This is mainly due to the *reduced number of available friends, their bandwidth limitations and their availability patterns*. In this section, we analyze the relationship between availability and download times in a F2F system.



(a) TTS distribution. (b) BTT distribution.

Fig. 10. TTS and BTT distributions of $1,000$ downloads at the start of the download phase using a random scheduling policy (RR placement).

Fig. 9 illustrates the behavior of the data availability and download optimal times to schedule (OTTS) during a time series analysis of 4 days for different values of n/k, $|\mathcal{F}|$ and β. First, we can observe in a time-series representation the impact of availability correlations on the mean number of available blocks maintained by friendsets. Clearly, *the strong patterns of nodes produce periods of data redundancy over-provisioning and under-provisioning*. This effect becomes more significant as the redundancy rate n/k grows.

As can be inferred from Fig. 9, the availability correlations of nodes also impact on the download OTTS. As expected, the download times present a near-opposite behavior than data availability; the more redundancy is available, the shorter the download time a user can achieve. However, we also see that the download OTTS values are highly dependent on the redundancy introduced in the system. That is, in Kad for $\beta = 1$GB and $n/k = 3$, the mean download OTTS values range from 2.54h. to 8.32h. ($\mu = 4.39$h.), whereas when $n/k = 1.5$ they range from 3.67h. to 13.94h. ($\mu = 7.98$h.). Further, results for Skype suggest that *larger friendsets notably improve the time to retrieve a file from the system.*

It is interesting to note that both availability and download OTTS plots are not completely opposite to each other. To wit, *we can find periods without availability and low download times, being the contrary also true*. Hence, start downloading a file in the evening may take several hours, due to the simultaneous disconnection of nodes before the download is completed. On the contrary, retrieving a file in early morning exhibits low download times even though the availability at the start of the download indicates that the file is unavailable.

In Fig. 9 we observe that the lowest TTS values are clearly dependent on the file size (β). However, the largest download

times seem to be less affected by the file size, specially in Kad. To explore this issue in depth, in Fig. 10 we present the TTS and BTT distributions obtained by a *random scheduling policy* for different values of β and n/k.

As can be observed in Fig. 10a, we see that for low TTS values the file size plays an important role. The dominance of low TTS values in Fig. 10a is because these downloads are performed at the beginning of the day. However, we observe that *as the download times become higher, TTS distributions become similar irrespective of the value of* β. The reason of that is that the order in which transfers are executed do not take into account availability correlations. Hence, TTS may grow significantly if, for instance, the least available friend was scheduled last when all friends follow a diurnal pattern. This fact is mirrored in the BTT distribution.

In Fig. 10b we present the BTT distribution of the schedules depicted in Fig. 10a. Note that whereas the majority of blocks are transferred in a reasonable time, there is a reduced number of blocks ($< 10\%$) exhibiting very large transfer times. Thus, the presence of availability patterns cause the unavailability of these blocks, which importantly increase the final TTS. Therefore, irrespective of the file size, *the largest TTS values are dictated by a minimal fraction of blocks whose schedule is affected by availability correlations*.

VII. Conclusions

In this work, we illustrated that F2F systems have specific characteristics (reduced friendsets, availability correlations) which need a special treatment. In this sense, we focused on the data availability provided by a F2F storage system.

First, contrary to conventional wisdom, we found that the presence of availability correlations offers a good trade-off between data availability and data redundancy. Moreover, we found that traditional approaches to calculate data availability are not suitable in a F2F scenario due to the presence of high correlations in small groups of friends. In consequence, they produce severe deviations when they are used to calculate data redundancy. This has a negative impact on the storage service.

To solve this problem, we proposed to use historical information on the availability of friends and use it to accurately calculate data redundancy. In our simulations, our technique obtained significant improvements compared with traditional redundancy calculation approaches.

Further, we showed the importance of properly balancing the storage load among friends in such systems. Finally, we unveiled the relationship between data availability and download times in this scenario. Our results suggest that in a F2F system, due to availability patterns, we should distinguish between if a file is currently available and if it is retrievable in a reasonable amount of time.

Acknowledgements

This work has been partly funded by the Spanish Ministry of Science and Innovation through projects DELFIN (TIN-2010-20140-C03-03) and RealCloud (IPT-2011-1232-430000).

References

[1] D. N. Tran, F. Chiang, and J. Li, "Friendstore: cooperative online backup using trusted nodes," in *SocialNets'08*, 2008, pp. 37–42.

[2] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social cloud: Cloud computing in social networks," in *IEEE CLOUD'10*, 2010, pp. 99–106.

[3] J. Li and F. Dabek, "F2f: Reliable storage in open networks," in *IPTPS'06*, 2006.

[4] R. Gracia-Tinedo, M. Sánchez-Artigas, and P. García-López, "F2BOX: Cloudifying F2F Storage Systems with High Availability Correlation," in *IEEE CLOUD'12*, 2012, pp. 123–130.

[5] R. Gracia-Tinedo, M. Sánchez-Artigas, A. Moreno-Martínez, and P. García-López, "FRIENDBOX: A Hybrid F2F Personal Storage Application," in *IEEE CLOUD'12*, 2012, pp. 131–138.

[6] R. Sharma and A. Datta, "Supernova: Super-peers based architecture for decentralized online social networks," in *COMSNETS'12*, 2012, pp. 1–10.

[7] G. Pallis, "Cloud computing: The new frontier of internet computing," *IEEE Internet Computing*, vol. 14, no. 5, pp. 70–73, 2010.

[8] S. A. Golder, D. M. Wilkinson, and B. A. Huberman, "Rhythms of social interaction: Messaging within a massive online network," in *Communities and Technologies*, 2007, pp. 41–66.

[9] S. Guha, N. Daswani, and R. Jain, "An experimental study of the skype peer-to-peer voip system," in *IPTPS'06*, 2006.

[10] M. Steiner, T. En-Najjary, and E. Biersack, "A global view of kad," in *ACM SIGCOMM IMC'07*, 2007, pp. 117–122.

[11] R. Sharma, A. Datta, M. Dell'Amico, and P. Michiardi, "An empirical study of availability in friend-to-friend storage systems," in *IEEE P2P'11*, 2011, pp. 348–351.

[12] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *EuroSys'09*, 2009, pp. 205–218.

[13] R. Zafarani and H. Liu, "Social computing data repository at ASU," 2009. [Online]. Available: http://socialcomputing.asu.edu

[14] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, "A survey of peer-to-peer storage techniques for distributed file systems," *Information Technology: Coding and Computing*, vol. 2, pp. 205–213, 2005.

[15] 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," in *ASPLOS'00*, 2000.

[16] R. Kotla, L. Alvisi, and M. Dahlin, "Safestore: a durable and practical storage system," in *USENIX ATC'07*, 2007, pp. 10:1–10:14.

[17] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two," in *HotOS'03*, 2003, pp. 1–6.

[18] L. P. Cox and B. D. Noble, "Samsara: honor among thieves in peer-to-peer storage," in *SOSP'03*, 2003, pp. 120–132.

[19] M. Sánchez-Artigas and E. Férnandez-Casado, "Evaluation of p2p systems under different churn models: Why we should bother," in *Euro-Par Parallel Processing*, 2011, pp. 541–553.

[20] R. Dunn, J. Zahorjan, S. Gribble, and H. Levy, "Presence-based availability and p2p systems," in *IEEE P2P'05*, 2005, pp. 209–216.

[21] S. Le Blond, F. Le Fessant, and E. Le Merrer, "Finding good partners in availability-aware p2p networks," in *Stabilization, Safety, and Security of Distributed Systems*, 2009, vol. 5873, pp. 472–484.

[22] A. M. Kermarrec, E. Le Merrer, G. Straub, and A. Van Kempen, "Availability-based methods for distributed storage systems," http://hal.inria.fr/hal-00521034/en, 2010.

[23] J. W. Mickens and B. D. Noble, "Exploiting availability prediction in distributed systems," in *USENIX NSDI'06*, 2006, pp. 6–6.

[24] K. Rzadca, A. Datta, and S. Buchegger, "Replica placement in p2p storage: Complexity and game theoretic analyses," in *ICDCS'10*, 2010, pp. 599–609.

[25] L. Pàmies-Juárez, P. García-López, and M. Sánchez-Artigas, "Heterogeneity-aware erasure codes for peer-to-peer storage systems," in *IEEE ICPP'09*, 2009.

[26] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in *IEEE P2P'04*, 2004, pp. 90–97.

[27] K. M. Greenan, J. S. Plank, and J. J. Wylie, "Mean time to meaningless: Mttdl, markov models, and storage system reliability," in *USENIX HotStorage'10*, 2010.

[28] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer Verlag, 2004.

[29] E. Fernández-Casado, M. Sánchez-Artigas, and P. García-López, "Affluenza: Towards universal churn generation," in *P2P'10*, 2010, pp. 1–2.

[30] Z. Yao, D. Leonard, X. Wang, and D. Loguinov, "Modeling heterogeneous user churn and local resilience of unstructured p2p networks," in *ICNP'06*, 2006, pp. 32–41.

[31] L. Toka, M. Dell'Amico, and P. Michiardi, "Data transfer scheduling for p2p storage," in *IEEE P2P'11*, 2011, pp. 132–141.

[32] L. Pàmies-Juárez, P. García-López, M. Sánchez-Artigas, and B. Herrera, "Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures," *Computer Networks*, vol. 55, no. 5, pp. 1100–1113, 2011.