2012 IEEE Fifth International Conference on Cloud Computing

FRIENDBOX: A Hybrid F2F Personal Storage Application

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

***Abstract*—Personal storage is a mainstream service used by millions of users. Among the existing alternatives, Friend-to-Friend (F2F) systems are nowadays an interesting research topic aimed to leverage a secure and private off-site storage service.**

However, the specific characteristics of F2F storage systems (reduced node degree, correlated availabilities) represent a hard obstacle to their performance. Actually, it is *extremely difficult* for a F2F system to guarantee an acceptable *storage service quality* in terms of transference times and data availability to end-users. In this landscape, we propose to resort to the Cloud for improving the storage service of a F2F system.

We present `FriendBox`: a hybrid F2F personal storage system. `FriendBox` is the first F2F system that efficiently combines resources of trusted friends with Cloud storage for improving the service quality achievable by pure F2F systems.

We evaluated `FriendBox` through a real deployment in our university campus. We demonstrated that `FriendBox` achieves high transfer performance and flexible user-defined data availability guarantees. Furthermore, we analyzed the costs of `FriendBox` demonstrating its economic feasibility.

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

I. INTRODUCTION

Increasingly, users will require ubiquitous *personal storage* to handle their ever-growing digital lives. Every user will handle hundreds of gigabytes to store digital information including photos, videos, work documents and communication flows like emails or social communication. To meet this demand, the number of Cloud offerings is large and growing, ranging from Dropbox and the likes, to IaaS providers like Amazon.

Although Cloud systems can meet the requirements of mass storage, some users are reluctant to move their data to the Cloud due to the large amount of control ceded to Cloud vendors, which includes the lack of data privacy [1] among other issues. In consequence, other alternatives have been devised to provide a *secure and private* off-site storage service.

In this sense, Friend-to-Friend (F2F) storage systems are nowadays an interesting research topic and they constitute an alternative approach to leverage personal storage [2], [3], [4]. The F2F paradigm is based on the synergy between social networks and storage systems: users store their data in a set of social 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 distributed systems —e.g. security, trust, incentives.

Nevertheless, the main drawback of F2F systems lies on the poor service quality they can provide. Actually, our research suggests that it is extremely difficult for a F2F system to provide adequate *data availability* and *transfer times* guarantees. Specifically, there are two simultaneous issues that put the feasibility of F2F systems at risk: **reduced node degree** and **availability correlations**.

The first problem refers to the *typically small number of available trusted friends to which store data*. For instance, over 63% of Facebook users have less than 100 friends [5]. In this line, other content distribution sites with social components (e.g. YouTube, Flickr, Friendster) present even lower connectivity among users. The bad news are, however, that for most users the majority of interactions occur only across a small subset of their social links —e.g. 20% of their friends account for 70% of all interactions [5]. This poses an important drawback: in a F2F system, users present *extremely small friend-sets*.

Second, real measurements on user behavior from online social networks suggest that friends are significantly correlated in their connectivity patterns [6], [7]. This implies that is rather probable to find all friends of a user simultaneously offline, particularly during night hours, which makes it impossible to maintain high data availability even when placing one replica at each friend. Consequently, *availability correlations degrade the storage service* experienced by users in a F2F system.

In this landscape, we propose a radically different approach: *resorting to Cloud storage services* for improving the performance of a F2F storage system.

We present `FriendBox`: a hybrid F2F personal storage system. `FriendBox` is the first F2F system that efficiently combines the resources of trusted friends with Cloud storage for improving storage service quality while preserving privacy. `FriendBox` provides a flexible and user-defined Cloud usage: *users are able to decide where to store their data*, which can be completely on friends, only in the Cloud or in a mix of them. We implemented in `FriendBox` advanced mechanisms which strategically use the Cloud to mitigate the problems of pure decentralization, improving the resulting service quality.

In this article, we describe the architecture and implementation of `FriendBox`. We validate `FriendBox` executing a real deployment in the Universitat Rovira i Virgili (URV) campus. We demonstrate that `FriendBox` provides high transfer performance with flexible user-defined data availability. Furthermore, we analyze the costs of `FriendBox` to evaluate its economic feasibility. Finally, we illustrate the impact of correlated availabilities on the performance of `FriendBox`; a hard problem for a pure F2F system performance.

The remainder of the paper is structured as follows. We discuss the related work in Section II, and describe the architecture of `FriendBox` in Section III. In Section IV, we overview the mechanisms that `FriendBox` implements to combat the main problems of F2F systems. In Section V, we illustrate the implementation of `FriendBox`. In Section VI we describe the experimental environment and the results that evaluate our system. Finally, we discuss about the future of `FriendBox` (Section VII) and we conclude in Section VIII.

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.20

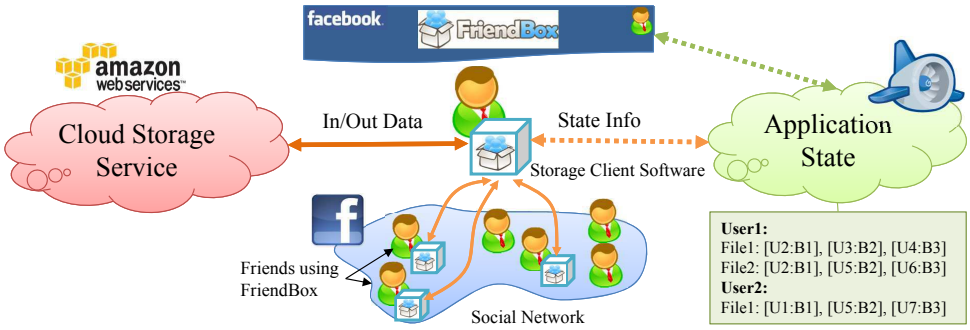

Fig. 1. Overview of FriendBox architecture. This picture illustrates how a user maintains storage links with some of his friends in a social network. Moreover, this user is able to store a fraction of his data in a Cloud storage service. The state information of a user's data is stored in the FriendBox Application State. Finally, users manage their storage relationships and check the state of their storage service in the FriendBox Social Front-end.

II. Related Work

Friend-to-friend. F2F systems emerged in the last years to overcome the limitations of P2P storage systems [8], [9]. Among the limitations P2P storage systems, the *instability and heterogeneity* of peers is an important issue that hinders the provision an appropriate service quality [10]. Furthermore, despite important efforts [11], the existence of *selfish behaviors* (free-riding) and the *lack of trust* among participants make end-users reluctant to adopt P2P storage systems.

For example, in [2], authors used friends instead of random peers to improve the stability of existing systems. They argued that P2P systems with random neighbor selection are very unstable and that using friends provides incentives for nodes to cooperate. This results in a more stable system which, in turn, reduces the cost of maintaining data.

Similarly, FriendStore [3] is a cooperative backup system where peers use their friends to store information. FriendStore aims to solve both the availability and denial-of-service problems thanks to trusted relationships. Users enter storage contracts with their friends via real-world negotiations. They validated their work developing a cooperative backup system.

A recent work in this field is [6]. Their empirical study of availability in F2F storage systems present a major drawback regarding data availability: if no friends are online, then data stored in the system will not be accessible. Whereas backup systems may be feasible using friends, storage systems require data availability that cannot be ensured with friends' resources.

In [12], authors present a Facebook application that aims to create a Social Cloud, enabling friends to share resources within the context of a Social network. Their prototype application is a marketplace where friends trade their resources using auctions and bidding mechanisms mediated by contracts. However, as stated previously, the availability of these resources cannot be guaranteed by the sole usage of friends.

Peer-assisted storage. Another related line of research is peer-assisted [13] storage systems that combine Cloud and peer resources. For example, in [14], authors present a hybrid architecture where resources at peers (spare bandwidth, storage space) are complemented with temporal usage of Cloud storage services. They demonstrated that hybrid systems can be comparable to traditional client-server architectures but at a fraction of their costs. In this line, FS2You [15] is a large-scale online storage system with peer-based assistance and semi-persistent file availability that was developed to reduce server bandwidth costs. Their measurement study demonstrated the feasibility of combining peers with Cloud resources.

To conclude, FriendBox is the first hybrid F2F solution that efficiently combines resources from trusted friends and Cloud services to provide a flexible, trusted and private personal storage service.

III. System Overview & Architecture

In this section, we overview FriendBox and its architecture, which is divided into three components: FriendBox Social Front-end, FriendBox Storage Client and FriendBox Application State.

A. FriendBox Social Front-end

In our architecture, the FriendBox Social Front-end provides three main services: social relationships provisioning, access control and service information web page.

The fact of coupling FriendBox within a social network facilitates the establishment of social storage relationships among users, a critical aspect for a F2F system.

User management in a massive distributed application is a complex task. For this reason, we employ a social front-end as entry point of FriendBox: only those users which are members of the social network are capable of accessing to our F2F system. Thus, user management and access control issues are partially delegated to the social network avoiding additional complexity to the storage system.

Finally, the FriendBox Social Front-end provides a very intuitive GUI to expose the most relevant service information to the user. For instance, users can inspect their monthly Cloud resource consumption and the distribution of their data within the system.

B. FriendBox Storage Client

In addition to the FriendBox Social Front-end, users must download the FriendBox Storage Client to connect to the system. This software enables users to perform basic data operations, such as storing and retrieving files from the system (friends and/or Cloud). Moreover, this software acts both as a client and a server in that our friends can store their information in the storage space we contribute to the system.

To cope with friends' instability, data redundancy must be added to the system to guarantee an acceptable level of data availability. To this end, a core functionality of the

FriendBox Storage Client is the generation of data redundancy before inserting a file into the system. We armed this client with various redundancy mechanisms as we describe in Section V-A.

One of the main novelties of FriendBox is to resort to Cloud storage services for improving the service quality achieved by pure decentralized F2F systems. The FriendBox Storage Client enables users to decide the distribution of data between their friends and the Cloud. This leaves to the user the control of his data and its placement within the system. In this line, the architecture of FriendBox supports simultaneous connections with several Cloud providers. This attractive feature opens the door for the development of new strategies to enforce privacy and avoid data lock-in.

In Fig. 1 we observe how a user distributes his data among friends and the chosen Cloud storage service.

C. FriendBox Application State

Essentially, the FriendBox Application State maintains up to date the data management information about users' files. This information expresses which friends store which files and the network address of each friend. This information is needed to perform friend-to-friend storage operations. The maintenance process of this information is carried out by FriendBox Storage Clients installed at participants. Clients communicate with the FriendBox Application State and update their state information via a Representational State Transfer (REST) API. In addition, this application is the backend of the FriendBox Social Front-end.

Storing the application state of FriendBox in the Cloud provides *service ubiquity*: a user is able to install the FriendBox Storage Client in any machine and obtain the information state from this Cloud application. To the best of our knowledge, no other F2F system provides this feature.

The role of the FriendBox Application State is illustrated in Fig. 1. In this figure, we show how a user communicates with the FriendBox Application State to transfer state information. In this example, a user sends a message informing that a new file has been stored in the system and containing which friends are responsible for data blocks from that file. As can be seen, the FriendBox Application State stores this information using mappings that relate data blocks with the friends who are responsible for them.

IV. Beyond FriendBox: Advanced Aspects

In this section, we provide a technical description about the problems that put the feasibility of F2F systems at risk: *reduced friend-sets* and *availability correlations*. Moreover, we overview the Cloud-based mechanisms introduced in FriendBox to mitigate these problems.

A. Understanding the problems of F2F systems

In FriendBox, we assume that friends alternate between *online* and *offline* states, with their online sessions being correlated over time. Correlation can be understood as the high probability that given an online user, his friends are also online, which corresponds well with the strong diurnal pattern empirically observed in social networks like *Facebook* [7].

Data Availability. To assure a given level of availability, FriendBox makes use of *Erasure Codes* (ECs), which has been proven to be more efficient in terms of redundancy than replication [16]. 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.

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:

$$n = \min\left\{ x \in \mathbb{N} \,\middle|\, \left(\sum_{i=k}^{x} \binom{x}{i} \bar{a}^i (1-\bar{a})^{x-i} \right) \geq A \right\}, \qquad (1)$$

where $\bar{a}$ is the average host availability. (1) simply accounts for all the possible combinations of finding k blocks (or more) out of n, times the probability that this happens.

Two important observations must be discussed here about (1). 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 (1). 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, (1) *assumes that hosts are not correlated*, which is by far not true in F2F systems. As before, this implies that (1) can highly underestimate the real data availability. Considering a diurnal pattern in host availability, it is almost impossible for a reduced number of storage nodes, irrespective of the amount of redundancy used, to provide high data availability, as all of them will be probably offline during night hours [6]. This leads to the following problem:

Problem 1. Offering a high level of data availability requires redundancy to be added to friends to mask their discontinuous correlated participation. If the amount of redundancy is too high, a F2F system could become impractical.

Data Transfer Scheduling. Concretely, when we refer to a *transfer*, we mean the connection with a remote node that causes the transfer of a single block of data to it. Clearly, a transfer may be interrupted if the remote node becomes offline during this process. This takes an amount of time, namely, *block transfer time* (BTT). For this reason, we refer to a *schedule* as the set of transfers concerning the same data object. Furthermore, 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 refer to the time to complete a schedule simply as the *time to schedule* (TTS) [17].

Neither the formalization of the data transfer scheduling problem nor the existing scheduling policies take into account availability correlations [17]. This implies that the 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 example of Fig. 2, where $n = |\mathcal{F}| = 3$ and the set of potential schedules is S_1 (exactly one encoded fragment to each friend). Because friends present discontinuous participation, we have depicted in gray the time slots where each node is online. This scenario clearly highlights the importance of a good schedule when storage friends are not available at all times.

With an optimal schedule the owner would send a fragment to p_3 in the first time slot, then another to p_1 in time slot t_2,

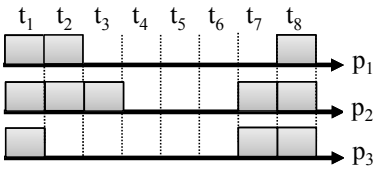

Fig. 2. Example of availability correlation.

and finally one to p_2 in time slot t_3, concluding the schedule. On the contrary, if the first block is sent to p_2, then the second block to p_1, the owner will have to wait for p_3 to come online again in time slot t_8 to complete the upload. If the availability pattern of p_3 had been considered, this schedule would have been discarded.

Problem 2. In a F2F system, finding the *most optimal scheduling plan* is important to provide an efficient storage service in terms of transfer times, which is not trivial. Furthermore, the role the *reduced number of friends* and *the availability correlations among them* is even more important in a F2F scenario.

B. Improving a F2F storage service with a Cloud backend

Indeed, as we describe next, a Cloud backend is *fundamental* to solve the problems illustrated in the previous section.

Improving data availability. As discussed in the previous section, 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 advocate for ensuring a high data availability during the day hours where friends are mostly online, what we call as *daily data availability*, measured in δ hours. Clearly, as a consequence of availability patterns, the required δ hours of data availability will be mostly concentrated during the online periods of nodes. To this end, we propose a novel hybrid redundancy scheme where a fraction of the data is permanently stored in the Cloud and the rest is supported by friends. To adjust the amount of redundancy required at friends, our algorithm make use of their availability histories[1].

First, `FriendBox` defines the fraction of a file to be stored in the Cloud as F_C. A high F_C will take more advantage of the superior availability of the Cloud, thereby reducing the amount of redundancy to be assigned to the friends to support the rest of the file: $1 - F_C$. However, a lower F_C will reduce the monetary costs of storing a file into the system. In this way, `FriendBox` allows users to trade monetary cost for data availability in order to meet their particular storage needs. We are not aware of *any other F2F system that benefits from the Cloud to provide a differentiated storage service per user*.

Note that for a $F_C < 1$ there will be less than k blocks permanently stored in the Cloud. This preserves the distinctive *data privacy* feature of F2F systems, since the Cloud vendor *cannot reconstruct the original file* by any means.

Reducing scheduling times. We resort to the Cloud for reducing scheduling times. In case of uploads, the Cloud acts as a temporal buffer to store those blocks belonging to unavailable nodes during the scheduling process. To this end, we implemented a *bandwidth maximizing friend-to-Cloud policy*. With this policy a user seeks to reduce transfer times as much as possible by fully utilizing its own bandwidth. If a node responsible for a block is not online, this policy automatically redirects that block to the Cloud. This leads to an important reduction of scheduling times when nodes are unavailable, in particular during the night hours. In any case, the extra blocks pushed to the Cloud are downloaded and deleted afterwards by the nodes to whom they were assigned.

For downloads, we prioritize block downloads from friends since they have no cost for the requester node. If there are less than k blocks available from friends, the remaining ones are downloaded from the Cloud.

[1]In `FriendBox`, the `FriendBox` Application State performs the availability monitoring of nodes within the system.

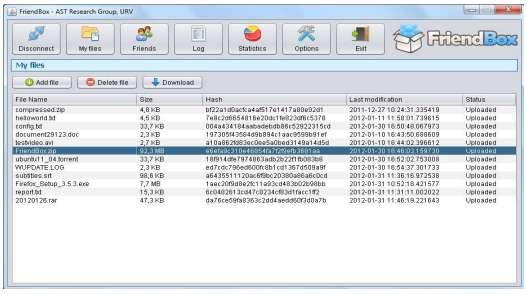

Fig. 3. View of the `FriendBox` Storage Client. The GUI enables users to perform several actions: data management operations (add, retrieve and delete files), inspecting which friends are currently participating in the storage system or modifying configuration settings (connection ports, Cloud accounts, etc).

V. Implementation

A. Storage Client: Blending Friend & Cloud Storage

The heart of the `FriendBox` Storage Client is implemented in Erlang[2], a programming language conceived to build massive distributed systems. Over $4,000$ lines of code implement the basic functionality of this software. This includes block level put/get/delete storage operations, management of parallel transfer connections, logging, etc.

The software architecture —based on the Erlang OTP Framework— permits to *easily extend* its functionalities. For example, currently the software includes two ways of providing data redundancy: *replication* and *Reed-Solomon erasure codes* (developed in C). However, other advanced data redundancy mechanisms and coding schemes can be introduced as independent modules in `FriendBox` without effort.

In the current version of `FriendBox`, the storage client provides a REST API to store data in Amazon S3[3]. Users can configure the fraction (F_C) of their files to be stored in the Cloud. In this sense, to introduce more flexibility on selecting a Cloud provider we are developing APIs for other Cloud storage services, such as Google Storage or OpenStack.

Finally, users interact with the storage client via a user-friendly GUI. This GUI has been implemented in Java and connected to the storage client through *Jinterface*, a package for communicating Java programs with Erlang processes. In Fig. 3 we show the storage client GUI.

B. Application State powered by Google AppEngine

The `FriendBox` Application State component has been developed with $3,000$ lines of Python code, plus HTML

[2]http://www.erlang.org
[3]http://aws.amazon.com/

and CSS to render the FriendBox Front-end in Facebook. This component has been hosted in GoogleApp Engine[4]. We employed this PaaS provided by Google for hosting our application since it offers an elastic and scalable service platform[5].

This application contains the necessary code for storing and updating the relational data structures that reflect the state of users and the location of their data. Moreover, this application defines a communication API for serving user requests. For accessing the API methods, the application requires an *authorizing token*. This security measure forces the periodical validation of users through the FriendBox Storage Client against the FriendBox State Application[6].

On the other hand, for its internal management, this application uses the REST API that Facebook exposes to get a range of user data including friends and profile information. Such an information is utilized, for instance, to manage the social relationships among users, i.e., FriendBox storage invitations.

C. Social Front-end in Facebook

The Social Front-end of FriendBox is integrated within the Facebook environment. This is possible since the FriendBox Application State follows the requirements and communication standards of Facebook.

From a user perspective, the Social Front-end presents a useful an ubiquitous source of informations about the state of a user's FriendBox storage service (see Fig. 4).

At the top of Fig. 4 we observe that the current user (Pedro García) has sent a FriendBox storage invitation to Marc Sánchez which is in *pending state*. This exemplifies how users interact among them to manage their storage relationships.

Another interesting feature we introduced in this component is a chart that shows how a user's data is distributed among his friends and the Cloud. This chart illustrates where a user's data is stored and to whom a user is storing data.

As many commercial services, we benefit from this component to present to users the amount of consumed Cloud storage resources. In the charts included in the "Cloud Consumption" section, we depict the traffic and storage capacity that have been consumed over the last months. Moreover, actions such as changing the FriendBox Storage Client password or listing the files stored in the system are present in this application.

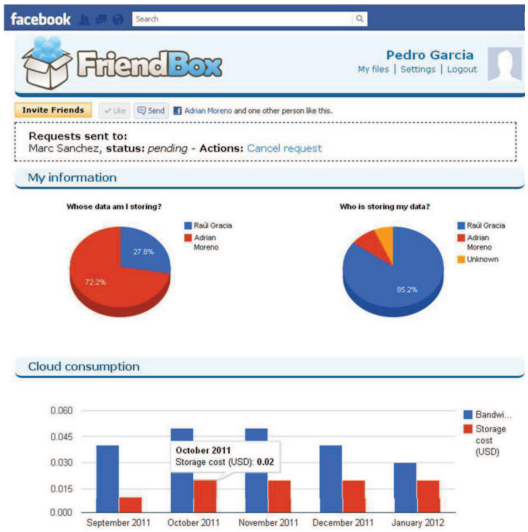

Fig. 4. FriendBox Social Front-end in the Facebook environment. This application enables users to manage their storage relationships, analyze the distribution of their data and check the Cloud resource consumption.

VI. Experimental Results

A. Scenario and Setup

Network. We deployed a group of FriendBox users in the URV campus, which are friends among them in Facebook. The group topology within the campus is depicted in Fig.5. In our scenario, users perform and serve storage operations in the system. However, we call nodes $1, 2$ and 3 *storage requesters* since they are automated FriendBox nodes, whereas the rest are real users. In this section, we analyze the service received by requester nodes in a real FriendBox deployment.

Availability. Users are connected to the system depending on the time they spend at the university. Thus, our experiments suffer from *availability correlations* since users in our campus present diurnal patterns. The availability of nodes during the experiment (24 hours) is depicted in the lower part of Fig. 5. In our experiments, requester nodes are continuously performing storage operations and therefore, they are always available.

Workload. The workload model of our experiments is simple. Requester nodes are alternatively performing file downloads and uploads during the whole experiment. These operations are performed over synthetic files of β bytes. We fixed $F_C = 0.5$ and the data redundancy[7] rate to $n/k = 2.5$. With these parameters we will analyze the resulting data availability δ. Moreover, data transfers of requester nodes are concurrently executed along the experiment to observe the effects of network congestion.

Data Transfer Scheduling. We applied a *random scheduling policy* to schedule transfers among friends for both uploads and downloads [17]. That is, the order of block transfers is chosen completely at random. In addition, FriendBox introduces a *bandwidth maximizing friend-to-Cloud policy* (Section IV). To wit, if the block's destination friend is unavailable, that block is uploaded to the Cloud. This mechanism reduces the upload TTS but incurs an extra cost, as later on the extra blocks must be downloaded from the Cloud by their owners.

Hardware. FriendBox clients are hosted in desktop computers (Intel Core2 Duo and AMD Athlon X2 processors) equipped with 4GB DDR2 RAM. The operating system employed is a Debian Linux distribution[8]. The users were connected via a 100Mbps switched Ethernet links. For gathering physical network information we used vnstat: a tool that keeps a log of network traffic for a selected interface. The rest of information presented in this section (e.g. scheduling times, data distribution) has been gathered by the FriendBox log system. Other important parameters in this experimental scenario are depicted in Table I.

[4]http://code.google.com/intl/en/appengine/

[5]Facebook applications are not hosted within the Facebook environment. Since we enable users to access their information from Facebook, we adopted a PaaS hosting instead deploying the application in our own servers.

[6]The FriendBox Storage Client must transact with the application every 20 minutes to refresh the token; otherwise the token expires.

[7]Although our data availability algorithm (Section IV-B) has been validated via simulation, a *long term* experimental evaluation is left for future work.

[8]FriendBox works for other platforms such as Windows and Linux Ubuntu.

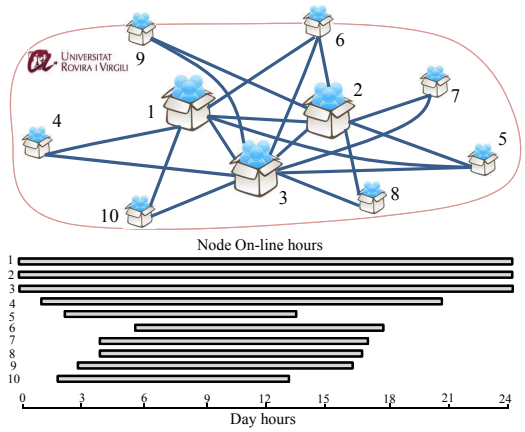

Fig. 5. Network topology and node availabilities configured in our experiments. Note that FriendBox nodes are correlated in their availability.

B. Measuring Scheduling Times

Next, we focus on analyzing the obtained times to schedule (TTS) among nodes. Fig. 6 illustrates the performance of upload and download TTS along one day.

First, we observe that scheduling times are notably short. This is due to the performance of the campus network (100Mbps). For instance, in case of uploads (Fig. 6), requester nodes transfer 500MB of data ($\beta = 200$MB, $n/k = 2.5$). However, none of these upload schedules takes more than 5 minutes to finalize. In case of uploading larger files, upload TTS are equally short: around 330 secs for $\beta = 400$MB and 474 secs for $\beta = 600$MB in average.

The average TTS is appreciably shorter for downloads than for uploads. The reason is that requester nodes only need to gather k blocks in order to reconstruct the original file. Further, since these k blocks are mostly collected from friends, transfers among campus nodes are faster than accessing to the Cloud, download TTS are short.

One of the most interesting points of this section refers to the TTS performance depending on the moment of the day. We can observe that scheduling times (uploads and downloads) are significantly shorter in the central hours of the day than in the rest. Actually, this is the time range where the majority of friends are online. From this result, we conclude that in these kind of scenarios the availability of friends improves the TTS performance. This is specially evident in case of downloads, since in the best cases there is no need of accessing the Cloud.

It is specially important to observe that FriendBox *is able to complete upload schedules irrespective of the availability of friends*. Thus, even when a fraction of friends are unavailable, the *bandwidth maximizing F2Cloud policy* redirects blocks to the Cloud in order to finalize the upload. Note that this is not possible in a F2F system; a user should wait until their friends come back on-line again to finalize the upload schedule.

In conclusion, FriendBox achieves fast data transfers among nodes and the Cloud. FriendBox also permits to upload files even when a user's friends are unavailable. Finally, we observed the impact of availability correlations on the TTS.

C. Data Availability

Now, we evaluate the data availability exhibited by requester nodes. In Fig. 7 we compare the data availability obtained by FriendBox and a pure F2F system in our scenario.

Parameter Description and Values	
Nodes in the system	10
Experiment duration	24 hours
Node storage capacity	10GB
Parallel upload/download connections	2, 2
Erasure codes original file fragments (k)	40
Cloud file fraction (F_C)	0.5
Object size (β)	200MB, 400MB, 600MB
Data redundancy (n/k)	2.5

TABLE I
PARAMETER CONFIGURATION IN OUR EXPERIMENTAL SCENARIO.

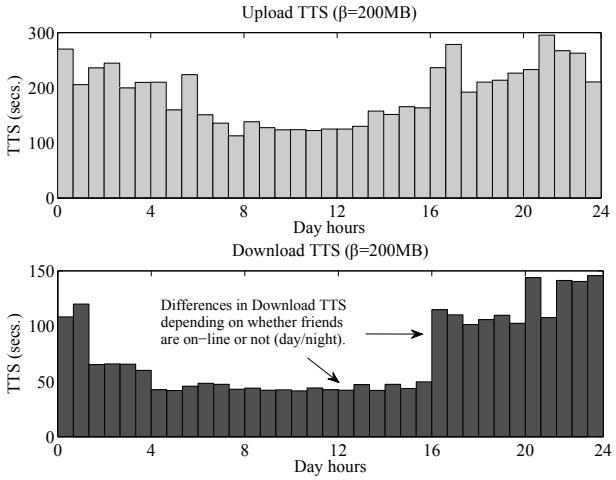

Fig. 6. Upload and download TTS along one day. Note that scheduling times are faster during the periods of higher node availability. This means that accessing to friends in our campus is faster than resorting to the Cloud.

As expected, for the same amount of data redundancy, users obtain a higher data availability with FriendBox than with a F2F system. For instance, FriendBox nodes 2 and 3 exhibit $\delta = 24$ hours of data availability per day. With a pure F2F approach, they exhibit $\delta = 15$ hours and $\delta = 20$ hours respectively. Thus, storing a fraction of a file into the Cloud alleviates the impact of offline friends on the resulting data availability. In this line, FriendBox *is able to obtain equal or higher levels of data availability than a F2F system with less data redundancy*. This is key to reducing scheduling times and making a F2F system scalable in terms of storage redundancy.

We observe that node 3 exhibits higher levels of data availability than other nodes. The reason is that node 3 has more friends for storing data than nodes 1 and 2. This gives us an important insight: *the size and behavior of a node's friend-set has important effects on data availability*.

In conclusion, FriendBox is flexible enough to satisfy the data availability requirements of users. Clearly, the amount of data redundancy, the role of the Cloud and the behavior of friends will dictate the service quality.

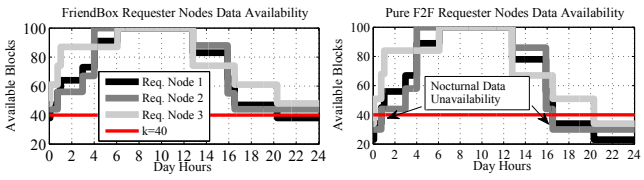

Fig. 7. Data availability of FriendBox compared with a pure F2F system.

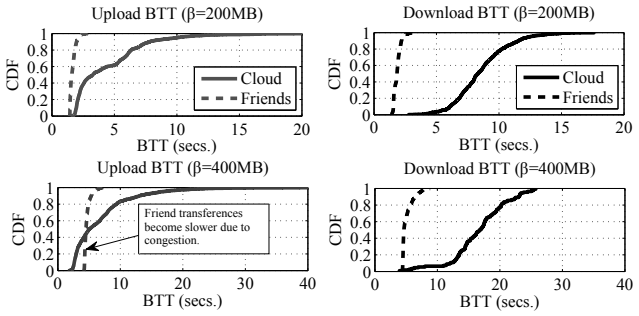

Fig. 8. Block transfer time (BTT) distribution depending on whether blocks are transferred from the friends or the Cloud.

D. Analyzing Transfer Capacity

In this section, we address the data transfer capacity of FriendBox. To this aim, we illustrate in Fig. 8 the distribution of block transfer times (BTT). Upload and download BTT distributions for friends and the Cloud are plotted separately.

First, the vast majority of block transfers among FriendBox nodes are faster than resorting to the Cloud. Again, this is mainly caused due to the high speed network available in the campus. Furthermore, we observe that the distribution of uploaded and downloaded blocks among nodes is quite similar. Obviously, in contrast to domestic network connections, the upload/download bandwidth asymmetry is not relevant within the campus.

Fig. 8 suggests that uploading blocks to the Cloud is slightly faster than downloading blocks from it. In our opinion, two distinct factors might lead to this result: First, downloading data from the Cloud is more common than uploading data to it. This might induce restrictions on download bandwidth served to a single user. Second, the download process implemented in the FriendBox Amazon S3 API is different to the upload process. Whereas blocks are loaded in memory and uploaded directly to the Cloud, the download process receives small chunks of data which are progressively stored on disk. This incurs extra time when downloading blocks from the Cloud. We will address this issue in depth in future tests.

Finally, Fig. 9 depicts the download bandwidth of requester node measured in two distinct moments: day and night. We observe that the diurnal download is faster than the nocturnal download. Basically, the bandwidth trace indicates that during the day *all* blocks are downloaded at high speed ($\approx$ 40Mbps); that is, they are gathered from friends. In case of a nocturnal download, only a fraction of blocks is downloaded from friends. These blocks are the first ones, which have a higher priority than blocks to be downloaded from the Cloud in our scheduling policy. Thus, the requester node should finally resort to the Cloud to finalize the download. We observe (Fig. 9) that the differences in transfer speed between the first part of the nocturnal download and the last part are significant. This result corroborates the results obtained in Section VI-B.

E. Monetary cost of FriendBox

In this section, we address the economic implications of our system to end-users. The monetary cost of FriendBox depends on the use of the Cloud. In our experiments we used

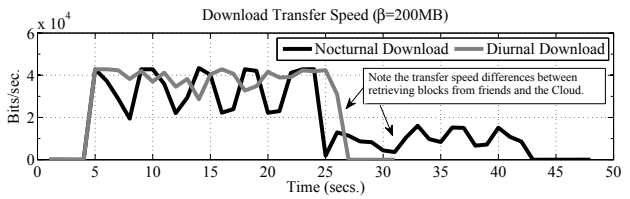

Fig. 9. Differences between diurnal and nocturnal download transfers. Clearly, the transfer speed of accessing to the Cloud is lower in our campus than downloading data blocks from friends.

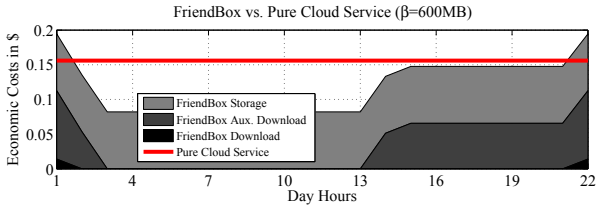

Fig. 10. Monetary cost comparison of FriendBox vs. Amazon S3. These costs capture the expenses to upload and download a file of β bytes, as well as keeping it stored for one month in the system.

Amazon S3 as a Cloud provider[9]. We compared the costs of FriendBox with the same service provided by Amazon S3. The costs depicted in Fig.10 capture the expenses to upload and download a file of β bytes, as well as keeping it stored for one month in the system.

First, we infer that FriendBox monetary costs are generally lower than Amazon S3 (Fig.10). Regarding storage costs, FriendBox is configured with $F_C = 0.5$. This means that only the half of a file is stored in the Cloud, inducing important savings. Clearly, the storage costs of FriendBox are limited by the value of F_C, which is selected by the user.

FriendBox has a particular cost labeled in Fig.10 as *auxiliary download*. This cost comes from the mechanism implemented in FriendBox for accelerating uploads (Section IV-B): when a node is uploading a file, all the *redundant blocks* whose recipients are offline at the moment of scheduling are redirected to the Cloud. When these previously offline nodes become available, they download those blocks they are responsible for. This introduces an extra economic cost. Thus, depending on the availability of nodes, the cost for auxiliary download will vary. Note that in our scenario, this cost is only present in the moments of lowest node availability.

Actually, in our experimental scenario *only* 2 *download schedules resorted to the Cloud to finalize*. This suggests that FriendBox can reduce costs in download traffic compared with a Cloud service. However, this cost is dictated by the amount of redundant blocks placed in friends and the availability of friends at the download instant.

Next we discuss the impact of availability correlations upon the monetary costs of the FriendBox service. Clearly, performing storage operations during the moments of lower availability of friends (night) is significantly more expensive than during the day. This leads us to a clear conclusion: FriendBox should be aware of the availability of a user's friends to give advise about the resulting service costs.

[9]According to Amazon's S3 at February 2012 we assume 0.120$ per GB of outgoing traffic and 0.140$ per GB/month of storage. Incoming traffic is free of charge.

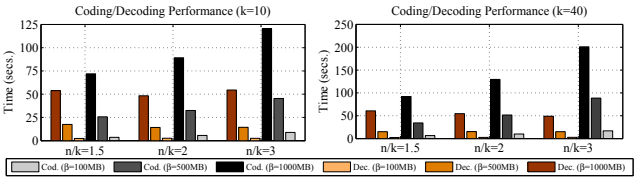

Fig. 11. `FriendBox` Erasure Codes performance.

We conclude that `FriendBox` can reduce the monetary cost of a pure Cloud solution, specially in the *long term*. These costs correspond to the permanent Cloud *storage* and the Cloud *outgoing traffic* of future file downloads. In addition, two parameters dictate the costs of `FriendBox` service: the fraction of data permanently stored in the Cloud (F_C) and the amount of redundancy generated to guarantee data availability (n/k). Finally, the availability patterns of friends produces variations in `FriendBox` service costs along the day.

F. Erasure Codes Performance

In the final part of the evaluation, we focus on the computational erasure coding scheme used in `FriendBox`. In this particular experiment, the hardware employed is different to the previous tests (Intel Pentium 4 3GHz and 2GB RAM). For each parameter configuration in Fig. 11 we plotted the average execution time from 100 executions.

At first glance, we infer from Fig. 11 that coding is generally affordable for domestic hardware when the size of the file is small or medium. However, when the size of the file to manage is large, coding becomes quite expensive in terms of both CPU time and memory. Therefore, although coding can be used for managing large files sporadically, it seems not adequate for performing *frequent* operations on large files.

As expected, we observe that whereas the encoding process is impacted by β, k and n/k, the decoding process is mainly affected by the size of the file β. We will study new ways to optimize the performance of the Erasure Codes scheme included in `FriendBox`.

VII. Work in Progress and Roadmap

Nowadays, `FriendBox` has released a first alpha prototype which has been tested in the URV campus with dozens of users. However, `FriendBox` is an ambitious project that is currently growing fast towards promising stages. Next, we describe the future stages of `FriendBox`.

Our primary objective is to leverage a secure, flexible and efficient Internet-scale storage system with `FriendBox`; to this end, we are currently introducing in `FriendBox` mechanisms such as NAT Traversal, data repair mechanisms and cryptography techniques. This represents the first milestone of `FriendBox`, that will be reached in the next months.

To undertake massive tests, we will benefit from our participation in the project RealCLOUD (IPT-2011-1232-430000)[10]. In the context of RealCLOUD, our research group is working on personal storage solutions over OpenStack Swift in collaboration with Tissat[11], a Spanish Cloud provider. In this line, `FriendBox` will leverage the Cloud storage resources provided by Tissat's Walhalla Data Center.

[10]http://www.realcloudproject.com
[11]http://www.tissat.es

VIII. Conclusions

In this work, we resort to a Cloud backend to overcome the limitations of pure F2F systems for providing a secure and private personal storage service.

We present `FriendBox`: the first hybrid F2F personal storage system. In `FriendBox` we implemented hybrid data transfer scheduling and data availability mechanisms to mitigate the drawbacks of pure decentralization. We validated `FriendBox` in a real deployment in the Universitat Rovira i Virgili campus. We demonstrated that `FriendBox` provides high data transfer performance and user-defined data availability guarantees. We examined the impact of the correlated availabilities on the performance of `FriendBox`; a hard problem for a pure F2F system. Furthermore, we analyzed the costs of `FriendBox`, showing its economic feasibility.

As a result of this research, we will release in short the first fully-equipped `FriendBox` version, in order to provide a flexible and private Internet-scale personal storage service[12].

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] S. Pearson, "Taking account of privacy when designing cloud computing services," in *Software Engineering Challenges of Cloud Computing*, 2009, pp. 44 –52.
[2] J. Li and F. Dabek, "F2f: Reliable storage in open networks," in *IPTPS'06*, 2006.
[3] D. N. Tran, F. Chiang, and J. Li, "Friendstore: cooperative online backup using trusted nodes," in *SocialNets'08*, 2008, pp. 37–42.
[4] G. Pallis, "Cloud computing: The new frontier of internet computing," *IEEE Internet Computing*, vol. 14, no. 5, pp. 70–73, 2010.
[5] 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.
[6] 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.
[7] 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.
[8] R. Kotla, L. Alvisi, and M. Dahlin, "Safestore: a durable and practical storage system," in *USENIX ATC*, 2007, pp. 10:1–10:14.
[9] J. Kubiatowicz et. al, "Oceanstore: An architecture for global-scale persistent storage," *ACM SIGPLAN*, vol. 35, pp. 190–201, 2000.
[10] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two," in *HotOS'03*, 2003, pp. 1–6.
[11] L. Cox and B. Noble, "Samsara: Honor among thieves in peer-to-peer storage," in *ACM SOSP'03*, 2003, pp. 120—-132.
[12] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social cloud: Cloud computing in social networks," in *IEEE CLOUD'10*, 2010, pp. 99–106.
[13] R. Sweha, V. Ishakian, and A. Bestavros, "Angels in the cloud: A peer-assisted bulk-synchronous content distribution service," in *IEEE CLOUD'11*, 2011, pp. 97–104.
[14] L. Toka, M. Dell'Amico, and P. Michiardi, "Online data backup: A peer-assisted approach," in *IEEE P2P'10*, 2010, pp. 1–10.
[15] Y. Sun, F. Liu, B. Li, B. Li, and X. Zhang, "Fs2you: Peer-assisted semi-persistent online storage at a large scale," in *INFOCOM*, 2009, pp. 873–881.
[16] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in *IEEE P2P'04*, 2004, pp. 90–97.
[17] L. Toka, M. Dell'Amico, and P. Michiardi, "Data transfer scheduling for p2p storage," in *IEEE P2P'11*, 2011, pp. 132–141.

[12]`FriendBox` Project Web page: http://ast-deim.urv.cat/friendbox