Giving Wings to Your Data: A First Experience of Personal Cloud Interoperability

Raúl Gracia-Tinedo*, Cristian Cotes, Edgar Zamora-Gómez, Genís Ortiz, Adrián Moreno-Martínez, Marc Sánchez-Artigas and Pedro García-López

Universitat Rovira i Virgili (Tarragona, Spain)

Raquel Sánchez

eyeOS (Barcelona, Spain)

Alberto Gómez and Anastasio Illana

NEC (Madrid, Spain)

Abstract

Personal Clouds are becoming increasingly popular storage services for end-users and organizations. However, the competition among Personal Clouds, their proprietary nature and the heterogeneity of synchronization protocols have led to a complete lack of interoperability among them. Regrettably, this situation impedes that users *share data transparently* across multiple providers. Even worse, the lack of interoperability has associated serious risks, such as *vendor lock-in*, in which users get trapped in a single provider due to the cost of switching to another one.

In this work, we contribute `DataWings`: The first interoperability protocol for Personal Clouds. `DataWings` consists of an *authentication management protocol* and a *storage API for file storage, synchronization and sharing* that adhere to the current authentication (OAuth) and REST standards, respectively. Moreover, we demonstrate the feasibility of `DataWings` by implementing the protocol in various providers (NEC, StackSync, eyeOS) and performing a real deployment evaluated with real trace replays of production systems (UbuntuOne, NEC). To our knowledge, this is the *first real-world experience* of Personal Cloud interoperability. Our experiments provide new insights on the performance implications that different *types of user activity* and the underlying *sharing network topology* have on the implementation of our protocol. We conclude that `DataWings` is flexible enough to leverage interoperability for heterogeneous Personal Clouds, opening the door for a broader adoption by other vendors.

Keywords: Cloud Storage; Personal Clouds; Syntactic Interoperability; Vendor Lock-in;

1. Introduction

Over the last years, the concept of Personal Cloud has been materialized by several successful commercial offerings. Services like Dropbox, Box or SugarSync provide online file *storage*, *synchronization*, and *sharing*, as well as accessibility from a variety of mobile devices and the Web. Furthermore, Personal Clouds

*Corresponding author.

Email addresses:

`{raul.gracia|cristian.cotes|edgar.zamora|genis.ortiz|adrian.moreno|marc.sanchez|pedro.garcia}@urv.cat` (Raúl Gracia-Tinedo*, Cristian Cotes, Edgar Zamora-Gómez, Genís Ortiz, Adrián Moreno-Martínez, Marc Sánchez-Artigas and Pedro García-López), `raquel.sanchez@eyeos.com` (Raquel Sánchez), `{alberto.gomez, anastasio.illana}@emea.nec.com` (Alberto Gómez and Anastasio Illana)

are also becoming a popular platform to deploy *external applications*, such as photo viewers or document editors, which access the storage layer via Representational State Transfer (REST) Application Programming Interfaces (APIs) and give added value to the storage service itself. According to the market reports [1; 2], Personal Clouds are meeting well users' storage and collaboration needs; for instance, Dropbox's user population reached 500 million in 2016 [3].

However, with the competition among vendors growing at such a high pace, the success for a Personal Cloud service lies on being an *all-in-one* personal storage solution for an increasing number of users. Whereas many customers may be unaware of the implicit risks, Personal Clouds are increasingly becoming large data silos that aim at concentrating as many applications and as much as data as possible. This yields that *native interoperability* across providers is non-existent today [4], probably due to the lack of incentives of big players to provide such service. Similarly, external applications managing Personal Cloud data need to be adapted to the semantics of each vendor. The roots of this problem lie deeply in that Personal Clouds are still essentially proprietary and they have not been the subject of standardization [5].

Regrettably, this *data silo model* impedes that users and applications share data transparently across multiple providers. Even worse, there are associated risks in the Personal Cloud model, such as *vendor lock-in*, where users stick to a single vendor due to the high cost and complexity of switching to another one [6; 7; 5]. From the viewpoint of users and companies, vendor lock-in has serious and diverse implications, ranging from unexpected Quality-of-Service (QoS) changes [8] to the lack of trust on unilateral variations on pricing policies or terms of service [9].

In this sense, one way for allowing customers to be more agile in responding to vendor lock-in is by offering a standard interoperability protocol among providers. Naturally, these protocols may benefit from the existing open standards already adopted by most Personal Clouds, such as REST APIs and OAuth [10]. While an open and standard protocol is a generic prerequisite for interoperability, we identify two specific factors that may favor interoperability among Personal Clouds: i) To design a protocol that *covers the core operational requirements* of most existing services, and ii) To encourage an *initial set of vendors* to deploy the protocol.

First, conversely to other services that exhibit an heterogeneous and ever-growing number of functionalities and varying data types (e.g., on-line social networks), most Personal Clouds provide similar functionalities on users' data (storage, synchronization and sharing). Such a *de facto* set of functionalities brings the opportunity of defining APIs that cope with the requirements of heterogeneous services in the long term, limiting the "agreement effort" among Personal Clouds.

Second, deploying an interoperability protocol on an initial group of providers may represent a great step towards its broader adoption. This is specially true in the Personal Cloud arena, where providers have not focused on interoperability as a primary feature. Therefore, a pilot interoperability experience among real-world vendors may encourage others to integrate the protocol as it can be seen as a new, value-added feature to the service.

All in all, we believe that the future will require Personal Clouds to provide users and applications with technical means to manage data transparently among vendors. Indeed, an interoperable Personal Cloud market will open up this sector to SMEs and strengthen their market position. We benefit from our previous observations to design the first interoperability protocol for Personal Clouds. As we show later on, we identify the necessary points of agreement among these services to enable interoperability: *user authentication* and *storage APIs*. Moreover, an open implementation of the protocol and a real deployment on various services certifies its feasibility, opening the door for a broader adoption.

1.1. Contributions

In the context of the EU project CloudSpaces[1], we aim at leveraging horizontal interoperability for the next generation of Personal Clouds. This motivates us to investigate and define open APIs and standard formats to share information between Personal Clouds, but also to migrate data from/to heterogeneous providers. Specifically, our contributions in this work are the following:

[1] http://cloudspaces.eu

- **Interoperability protocol design**: We design `DataWings`, a syntactic interoperability protocol tailored to Personal Cloud services. `DataWings` defines an *authentication management protocol* to grant external users access to Personal Cloud resources. Moreover, data exchanges among providers and applications is based on a novel *REST API for file storage, synchronization and sharing*.
- **Open-source protocol implementation**: We provide the *implementation* of `DataWings` in a clean service framework promoting adoption and third-party development. At the time of this writing, `DataWings` is integrated in NEC, StackSync and eyeOS cloud services.
- **Use-case applications**: We demonstrate the capabilities of the protocol with an advanced *proof of concept*: The eyeOS Cloud Desktop. Concretely, eyeOS has integrated `DataWings` to provide users with transparent access of data stored at StackSync and NEC Personal Clouds.
- **Real-world multi-vendor deployment**: Finally, to validate our protocol, we perform a real deployment and execute a workload based on back-end traces of NEC and UbuntuOne (U1) [11], the Personal Cloud of Canonical Ltd. We also investigate the performance implications that different *types of user activity* (upload/download oriented users) and the underlying *sharing network topology* have on the implementation of `DataWings` by NEC and StackSync. To the best of our knowledge, this work is the first to evaluate interoperability among Personal Clouds.

The rest of the paper is organized as follows. In Section 2, we overview the related works, stressing the main differences of this paper compared with current interoperability approaches. Section 3 provides basic background on Personal Clouds, jointly with a brief description of REST and OAuth technologies. In Section 4, we describe our interoperability protocol, namely `DataWings`. Section 5 briefly describes the cloud services already adopting`DataWings` —i.e, eyeOS, NEC and StackSync—, as well as the details of their respective implementations. Section 6 presents the experimental framework and the performance results of the Personal Clouds when making use of `DataWings`. We provide some discussion and conclude the paper in Section 7.

2. Related Work

The vendor lock-in problem has been recognized by the European Network and Information Security Agency as a high risk [12], and the Personal Cloud is not an exception. The root of this difficulty is the lack of interoperability and standard mechanisms to exchange data across providers. Thus, solving the vendor lock-in problem requires new technical means to share data horizontally across Personal Clouds.

There are two types of interoperability: *syntactic* and *semantic*. Syntactic means that cloud providers are trying to communicate through standardized mechanisms, including interfaces, data formats, and communication protocols [13; 14]. Semantic interoperability, on the other hand, means that information is not only exchanged but also interpreted in order to be used [15]. There exist various approaches to achieve semantic interoperability, such as mediators and middlewares that abstract the functionalities of heterogeneous providers [16] or ontologies that describe the meaning and structure of service entities and operations [17; 18], to name a few. In this work, we focus on *syntactic interoperability* as an essential building block to provide users with transparent data access and exchanges across Personal Clouds, while minimizing the "agreement effort" on the provider's side.

Cloud APIs. The advanced progress in syntactic interoperability at the platform and infrastructure level (e.g., the FP7 projects VISION Cloud [19], mOSAIC [20] and REMICS [21]) provide the technical means for easily sharing data and applications across clouds. There are also a number of companies, such as Deltacloud[2] and RightScale[3], committed to aggregate a variety of cloud APIs to unify the access to these services from a user's viewpoint [22]. On the other hand, organizations like Distributed Management Task

[2]https://deltacloud.apache.org/
[3]http://www.rightscale.com/

Force (DMTF), Open Grid Forum and Storage Network Industry Association (SNIA) are also leading efforts on standardizing APIs among cloud services. However, despite remarkable efforts such as the Cloud Data Management Interface (CDMI) [13] or Open Cloud Computing Interface (OCCI)[23], major Infrastructure-as-a-Service (IaaS) providers such as Amazon Web Services and Microsoft Azure are still imposing their own APIs. Similarly, Personal Clouds are not already adhered to standard APIs.

In our view, there could be two plausible reasons for this: First, most vendors provide their own open API[4] as no prior work or organization proposed protocols to leverage interoperability across Personal Clouds. That is, most well-known interoperability initiatives target IaaS providers (e.g., CDMI) and do not meet all Personal Cloud requirements, such as efficient syncing or data sharing functionalities, which require advanced metadata primitives on files that are not offered by CDMI. To inform this argument, authors in [24] build a CDMI-compliant deduplication that only provides simple `PUT`, `GET`, `DELETE` primitives to users. Given that Personal Clouds need metadata information on file versions and members within a shared folder, this approach seems insufficient to leverage interoperability among these systems.

Second, there is no prior pilot experience of a group of providers implementing interoperability. We believe that a pilot interoperability experience may incentivize other vendors to adhere to the protocol. We address both issues in this work.

Cloud federations. Heimbigner and McLeod proposed the concept of service federation in the mid 1980's as a way of coordinating isolated information management systems [25]. A more contemporary definition can be found in the work of Kurze et al. [7], where authors define that a cloud federation comprises services from different providers aggregated in a single pool supporting three basic interoperability features: resource *migration*, *redundancy* and *combination*. We found particularly interesting the concept of resource migration [26] to mitigate the vendor-lock in problem for Personal Clouds.

Among the existing approaches, the Reservoir model [27] is an architecture to federate groups of cooperative cloud providers with business-aligned SLAs. Authors in [28] present a distributed network of proxy agents that disseminate information about the available cloud resources to facilitate autonomic resource sharing among them. Vernik et al. [14] propose an on-boarding federation system. It enables data exchange among cloud providers while providing continuous access and a unified view over the data in the old cloud and the new cloud, and over data in transit. Closer to our approach, Villegas et al. [29] devise a federation model in which interoperability among cloud providers occurs independently at each layer of the cloud stack. In fact, our interoperability protocol can be seen as an horizontal Personal Cloud federation that encompasses syntactic interoperability agreements at the *storage* and *user authentication* layers [27; 29].

Apart from resource management, user authentication is a particularly interesting aspect of cloud federations [30]. For instance, authors in [31] propose a Cloud Single-Sign On (SSO) Authentication protocol to enable clouds within a federation to access and offer resources. In [32], Huang et al. present a SSO brokering system that cloud providers should trust as an identity service within the federation.

However, we consider that the agreement effort of delegating identity management to an external service is too high in the Personal Cloud arena. As other proposals based on OAuth [33], we present an authorization protocol to enable external users and applications to access shared resources of a Personal Cloud, but the identity, security and access settings are internally controlled by each provider. This agreement boundary for access control leaves Personal Clouds the option of adopting the protocol without changing core security policies and keeping their own identity service.

Multi-cloud systems for storage diversity. In the last decade, several works proposed systems that provide users with transparent and unified management of the storage space of multiple cloud providers [34; 35; 36]. These systems aim at mitigating vendor lock-in and/or improve performance [37; 36] and reliability [38] based on a simple principle: If a user was able to transparently store and migrate data across multiple providers, taking benefit of a new provider would entail switching only a fraction of the data. In the field of Personal Clouds, authors in [39] propose an storage mediator that picks a combination of providers based on active monitoring. However, this type of *multi-cloud brokers* require from a dedicated platform to operate. Even worse, data migrations are still costly in terms of transfer bandwidth since data objects should be downloaded first from the old provider to be uploaded to the new one.

[4] `https://www.dropbox.com/developers-v1/core/docs`

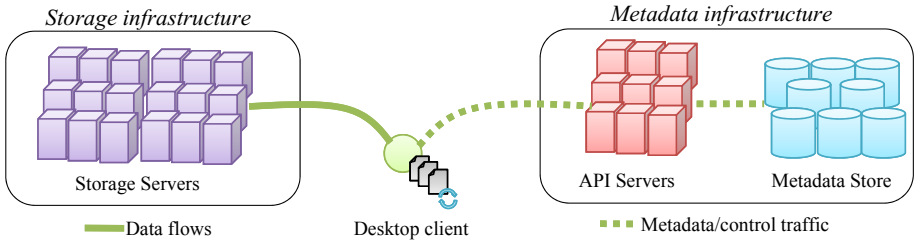


Figure 1: High-level overview of the architecture of Personal Clouds like Dropbox. As can be appreciated, these systems explicitly decouple the management of file metadata and data.

Our protocol achieves horizontal interoperability among providers, which yields a better way of exploiting storage diversity than dealing with data stored at several providers from the client-side. That is, users delegate data migrations on Personal Clouds via our protocol that, in turn, can make such migrations more efficient as direct transfers among providers. Nonetheless, our interoperability protocol can also provide applications with *client-side interoperability*. As we demonstrate with our implementation in eyeOS cloud desktop, having a unified storage protocol on several providers (i.e., NEC, StackSync) greatly facilitates the task of managing storage diversity in Personal Clouds.

Key differences with previous works. Conversely to prior research, this work aims at leveraging horizontal and client-side interoperability among Personal Clouds. To this end, we identify the core functionalities that should be agreed among Personal Clouds (authorization, storage API) and design an interoperability protocol that addresses these functionalities. As a distinctive contribution of this work, we also provide a real-world performance evaluation of vendors that already implement our protocol. We evaluate the impact that user behavior has on the performance of our protocol [40], such as the type of user activity (upload/download oriented) or the degree of connectivity among the users accessing the shared folders.

3. Background

3.1. Personal Clouds in a Nutshell

A Personal Cloud is an online cloud service for personal information that enables users to *store*, *synchronize* and *share* data from a variety of devices and OSes. Moreover, a Personal Cloud is a platform to deploy third-party applications that provide value-added services on users' data. Numerous popular services such as Dropbox, SugarSync and Box fall under this definition.

As can be observed in Fig. 1, from an architectural viewpoint a Personal Cloud exhibits a 3-tier architecture consisting of: i) *Desktop clients*, ii) *Synchronization or metadata service* and iii) *Data store* [41; 11]. Thus, these systems explicitly decouple the management of file contents (data) and their logical representation (metadata). Many services, such as GoogleDrive and Microsoft OneDrive, own the infrastructure for managing both metadata and data storage. However, the design of Personal Clouds enables other services —e.g., UbuntuOne or Dropbox until 2014[5]— to only own the infrastructure for running the metadata service, which processes requests that affect the virtual organization of files in user volumes. The contents of file transfers can then be outsourced and securely stored in a third-party Cloud vendor, such as Amazon S3. An advantage of this approach is that a Personal Cloud service can easily scale out its storage capacity thanks to the "pay-as-you-go" model, avoiding costly investments in storage resources.

In general, Personal Clouds provide clients with 3 main types of access to their service: i)Web/mobile access, ii) REST APIs [42] and iii) Desktop clients. In this paper, we focus on the REST API access provided by Personal Clouds in order to enable interoperability among services.

[5] https://blogs.dropbox.com/tech/2016/03/magic-pocket-infrastructure/

In this sense, Personal Clouds provide third-party applications with REST APIs to interact with the service, making it possible for them to execute data management operations (`PUT`, `GET`, etc.) on files stored in user accounts. One can easily find similarities between these APIs (files/accounts) and the operation of object storage services (objects/containers) [43]. A great advantage of these REST APIs is that they abstract clients from the service internals in order to manage files within user accounts.

We found that these API services are powerful abstractions that have not received enough attention from the research community. We believe that designing a unified storage REST API for Personal Clouds may be a great step towards the interoperability of these services. Thus, in the following we provide a closer look to the functioning of these REST APIs services.

3.2. REST APIs & OAuth

REST is an approach to design and communicate with Web services, typically running over HTTP (Hypertext Transfer Protocol). REST services define a contract or API in form of XML/json file. REST services are specially suitable for client-server applications, in which clients can perform a limited number of operations (labeled as *verbs* corresponding to HTTP methods) on a certain set of resources (labeled as *nouns*) , and identified by unique Universal Resource Identifiers (URIs). To draw an example, let us consider the following request in the Dropbox API:

`https://content.dropboxapi.com/1/files/auto/{my_file}`

In this case, the client is requesting the `content.dropboxapi.com` server to perform an action on a resource. In particular, the requested resource is a file (`/files/`) identified by "my_file" and the action to be performed is to download it, since the HTTP method invoked is `GET`. This simple example illustrates the simplicity of REST services, that are prevalent in today's mobile and social networking applications, IaaS providers, and many other applications with automated business processes.

Most Personal Clouds provide proprietary REST APIs, along with their client implementations, to make it possible for developers to create novel applications. There are two types of API calls: Metadata and data management calls. The former type refers to those calls that retrieve information about the state of the account (i.e., folders, file names), whereas the latter are those calls targeted at managing the stored files in the account.

To authorize and authenticate API calls, OAuth [10] is a standard technology that is normally used with REST. With OAuth, a client (i.e., resource owner) provides applications with "secure delegated access" to server resources on behalf of him. That is, after a token exchange procedure, the OAuth authorization server issues *access tokens* and sends them to a third-party application, given the approval of the resource owner. At this point, the application uses the access token to access the protected resources hosted by the resource server. In this sense, most Personal Cloud APIs incorporate OAuth for granting third-party applications and their own clients (Web, mobile, etc.) access to the files stored in user accounts.

4. DataWings: Protocol Description

4.1. Protocol Design and Requirements

`DataWings` is a *provider-centric* interoperability protocol. That is, Personal Clouds that opt to enable data and metadata access to folders/files via `DataWings` should also provide a proper protocol implementation in their back-end infrastructure. In particular, the provider-centric design of our protocol yields several advantages in terms of *simplicity*, *data integrity*, and *data security*. Accessing to remote files and folders via `DataWings` is as simple as in the case of accessing a third-party Personal Cloud, given that it uses the same open standards, such as REST and OAuth, which are already known by providers and developers. As we show later on, the protocol makes it easy to develop algorithms to store, synchronize and share data from a client's perspective (see Section 5).

Considering file synchronization, the integrity of data is a paramount aspect. That is, several users in a Personal Cloud may be concurrently updating a given file via our protocol, which may potentially cause edition conflicts. Therefore, it is a requirement for providers adopting `DataWings` to implement data

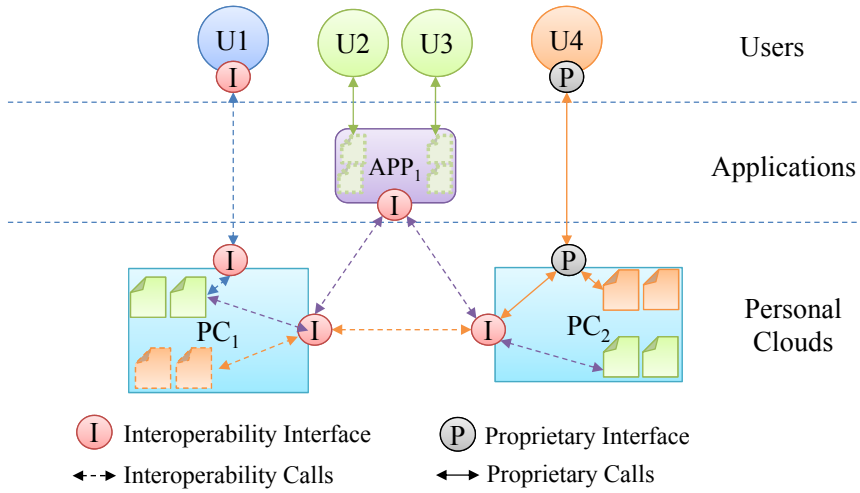


Figure 2: Different deployment models of the `DataWings` protocol. We can observe users and applications that directly resort to `DataWings` in order to manage their data across multiple providers (client-side interoperability). Moreover, we can observe a user that is still making use of the old proprietary storage API and his primary Personal Cloud manages his data stored in an external vendor (provider-side horizontal interoperability).

integrity mechanisms; for instance, providers may choose to implement serialization or locking mechanisms to enforce strict file integrity (e.g., WebDav), or they may enable concurrent edits on a file by offering the potential conflicting file versions for manual conflict resolution (e.g., Dropbox or StackSync [44]). In fact, guaranteeing data integrity on files stored across diverse Personal Clouds in such a provider-centric protocol design is more practical than recently proposed client-centric protocols [36], which achieve data integrity via complex distributed consistency algorithms (e.g., Paxos).

Moreover, the provider owning the data accessed via `DataWings` is the one controlling the access to it. This means that the customers accessing data via our protocol are always subject to the *internal security policies of the data owner*: Interoperable files are not scattered across multiple vendors that were not selected by the user creating the data, but only accessed with appropriate credentials and permissions by other users/vendors.

`DataWings` is flexible enough to leverage a variety of deployment strategies that involve *users*, *applications* and *Personal Cloud vendors*. Broadly speaking, we can divide these strategies in two types: *Client-side* interoperability and *provider-side* horizontal interoperability. These strategies refer to the way storage is managed and they can be observed in Fig. 2.

On the one hand, by *provider-side horizontal interoperability* we refer to a model in which a Personal Cloud is in charge of transparently performing the required data exchanges with other services to provide users with a unified view of their data. More importantly, users of one service *do not need* to be registered in other services they aim at interoperating with; `DataWings` offers means of providing data access to external users of a Personal Cloud. With this model, users and applications can still use the existing proprietary API, as the provider can hide the interoperability interactions with other services. This can be observed in Fig. 2 in which user u_4 is managing his own files stored at PC_2, as well as shared files from PC_1. In this specific case, u_4 does not interact directly with PC_1; instead, PC_2 transparently forwards u_4's requests on files stored at PC_1 on behalf of him after the appropriate access credential exchange (see Section 4.2).

On the other hand, *client-side interoperability* means that clients can manage *their own accounts at various Personal Clouds* adhered to the protocol. Retaking the example in Fig. 2, users u_1, u_2 and u_3 are users of PC_1 and share files. However, users u_2 and u_3 access these files at PC_1 via a third-party application that also aggregates their own files stored at PC_2. As can be inferred, this type of interoperability encompasses both

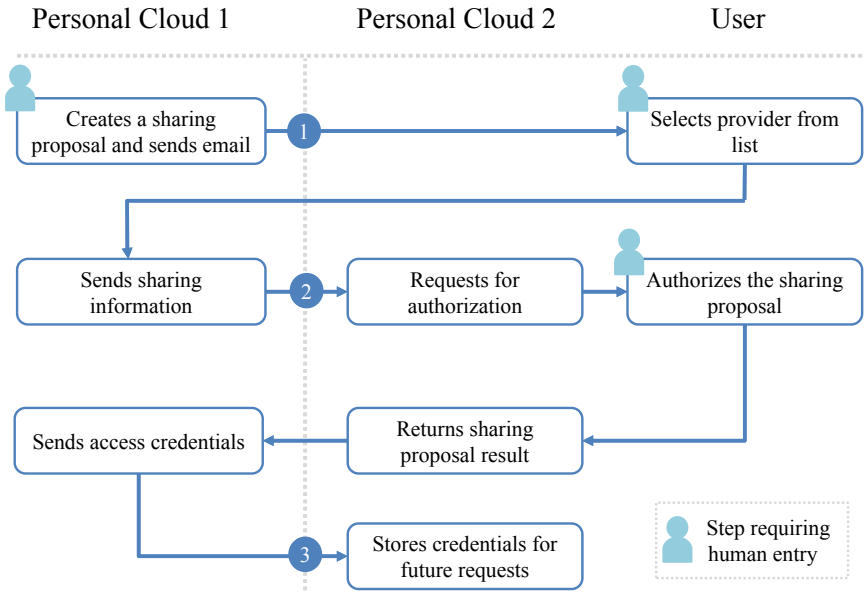


Figure 3: High-level diagram of the credential exchange workflow between users of two Personal Clouds integrating `DataWings`. The ultimate purpose of this process is to provide a user in Personal Cloud 1 (PC_1) with secure delegated access to files owned by a user in Personal Cloud 2 (PC_2).

users and applications, and leverages a variety of benefits and use cases. For instance, multi-cloud storage systems [34; 35] could greatly simplify the management of heterogeneous and ever-changing proprietary storage APIs. As we demonstrate in Section 5 with eyeOS, applications accessing Personal Clouds via `DataWings` are able to provide transparent access to user's files that reside at multiple providers.

From Fig. 2 it is interesting to note that the integration of `DataWings` to the operation of vendors and applications does not force them to exclude the existing proprietary protocols. Our protocol can be used in parallel with proprietary storage APIs and authentication mechanisms for a smooth adoption process.

Technically, `DataWings` has two main building blocks: The *authentication management protocol* (Section 4.2) and the *storage API for file storage, synchronization and sharing* (Section 4.3). As we describe next, the former defines a standard way of granting external users access to specific folders and files (i.e., resources), whereas the latter provides a standard way to manage data and metadata for Personal Clouds.

4.2. Authentication Protocol Workflow

There are three entities that are involved in the credential protocol: *Users*, *Personal Clouds*, and *end-points*. Users are the active entities that aim at exchanging data from various Personal Clouds via the interoperability protocol. By a Personal Cloud user, we mean that the same physical person may represent more than one user if he or she aims at making use of interoperability across his/her storage accounts. On the other hand, Personal Clouds are the entities that store part of the data to be exchanged and provide a compliant implementation of the interoperability protocol.

In this sense, our protocol also defines end-points that are accessible via a URL. In particular, end-points represent system actions to manage users' access to resources upon interoperability requests. The end-points in our protocol are the following:

- *Share end-point*: End-point that presents the interoperability proposal to the user and obtains authorization to enable external access to a specific resource.
- *Unshare end-point*: End-point used to finish the interoperability agreement between two users related to a certain resource.

	Field	Type	Description
Proposal	share_id	string	A randomly generated value that uniquely identifies the interoperability proposal.
	owner_name	string	An absolute URL to access the shared resource located in Personal Cloud 1.
	resource_url	string	The name corresponding to the owner of the folder.
	owner_email	string	The email corresponding to the owner of the folder.
	folder_name	string	The name of the folder.
	permission	string	Permissions granted to the recipient. Options are read-only and read-write.
	recipient	string	The email corresponding to the user who the folder has been shared with.
	callback	string	An absolute URL to which the Personal Cloud 2 will redirect the User back when the invitation step is completed.
	protocol_version	string	MUST be set to 1.0. Services MUST assume the protocol version to be 1.0 if this parameter is not present.
Accept	share_id	string	A randomly generated value that uniquely identifies the interoperability proposal.
	accepted	string	A string indicating whether the invitation has been accepted or denied. true and false are the only possible values.
Grant	share_id	string	A randomly generated value that uniquely identifies the interoperability proposal.
	auth_protocol	string	The authentication protocol used to access the shared resource (e.g. OAuth).
	auth_protocol_version	string	The version of the authentication protocol (e.g. 1.0a).

Table 1: Credential exchange message fields in `DataWings`.

- *Credentials end-point*: End-point needed to provide the access credentials for an external user to a resource of a Personal Cloud.

Granting access to an external user to a Personal Cloud service requires a credential exchange process, which consists of *user invitation*, *invitation acceptance* and *access credential provision*. Fig. 3 shows a sequence diagram of the process.

User invitation. Let us imagine a user u_a who wants to share with user u_b a resource r that resides in Personal Cloud PC_1. Therefore, in the domains of PC_1, u_a selects r as a resource that he wants to share with u_b. To start the sharing process, u_a should provide PC_1 with sufficient information about u_b to send him an interoperability invitation. As we show later on, in our implementation we make use of a user's email address to send such invitations, although other channels (e.g., mobile phone) may be valid as well. At this point, we assume that u_a's *interoperability invitation* is received by u_b and that he or she agrees on proceeding with the process.

Inside the invitation, u_a provides an URL pointing to the *share end-point* located at the domains of PC_1. By making use of this URL, u_b is taken to the PC_1 *share end-point* where he is prompted to enter his Personal Cloud service, in this particular case PC_2.

As u_b finishes selecting its own Personal Cloud, PC_1 creates an *interoperability proposal* (see Table 1). Technically, the interoperability proposal is an HTTP request to an URL that points to PC_2's *share end-point*. This request tells PC_2 that u_b will not only manage folders and files contained in PC_2, but also an external resource r stored at PC_1.

Invitation acceptance. At this point, let us assume that the interoperability proposal has been received by PC_2. Once u_b returns to PC_2's domains, the Personal Cloud provides u_b with the details of the invitation request, including the resource r to be shared. To explicitly accept the invitation, u_b should provide PC_2 with his credentials —if he is not already logged-in the service— to authenticate the confirmation.

Once PC_2 has obtained approval or denial from u_b, PC_2 must use a callback to inform PC_1 about u_b's decision. PC_2 uses the callback to construct an HTTP GET request and redirects u_b to that URL with the acceptance decision added as a query parameter (see Table 1).

Granting Access Credentials. When PC_1 receives the affirmative proposal result it must provide the access credentials to PC_2 in order to be able to obtain the shared resource r. PC_1 sends an HTTP request to an URL that points to PC_2's *credentials end-point*. This request will provide the necessary authentication protocol information to PC_2 for transparently generating the credential for accessing r from u_b's viewpoint.

In this sense, since the credential subsystem may vary among Personal Clouds, the credential request should specify certain parameters (see Table 1). For instance, this forces PC_1 to specify what type of authentication protocol and version must be used to access the resource. The authentication protocol and version used by PC_1 is beyond the scope of this specification, but OAuth 1.0 or OAuth 2.0 is recommended. There

REST Call Description	HTTP Method	URL
Create a file	POST	`/file?{name=file_name}&{parent*=folder_id}`
Upload file data	PUT	`/file/{file_id}/data`
Download file data	GET	`/file/{file_id}/data`
Delete a file	DELETE	`/file/{file_id}`
Get file metadata	GET	`/file/{file_id}`
Update file metadata	PUT	`/file/{file_id}?{name*=new_file_name}&{parent*=folder_id}`
Get file versions	GET	`/file/{file_id}/versions`
Get file version metadata	GET	`/file/{file_id}/version/{version_number}`
Get file version data	GET	`/file/{file_id}/version/{version_number}/data`
Create a folder	POST	`/folder?{name*=folder_name}`
Delete a folder	DELETE	`/folder/{folder_id}`
Get folder metadata	GET	`/folder/{folder_id}`
Get folder content metadata	GET	`/folder/{folder_id}/contents`
Update folder metadata	POST	`/folder/{folder_id}?{name*=new_folder_name}&{parent*=parent_folder_id}`
Share a folder	POST	`/folder/{folder_id}/share`
Unshare a folder	POST	`/folder/{folder_id}/unshare`
Get folder members	GET	`/folder/{folder_id}/members`

Table 2: Storage API specification for Personal Clouds. Fields marked with (*) are optional.

are authentication-specific parameters in the credentials request that may include values like tokens, timestamps or signatures, which have been omitted due to space constraints. The full protocol specification and request parameters is available on-line[6].

It is important to note that our credentials management protocol does not force a Personal Cloud to change its own identity service. As we describe in Section 5, Personal Clouds may internally link an external user accessing the service with a new user created during the credentials exchange for this purpose. This enables Personal Clouds to implement our protocol while keeping its own identity system, while enforcing the same access control and security policies either to external and internal users.

Thus, when this protocol completes, u_b gets the access credentials and she/he is able to transparently access a shared resource r located at PC_1 via a storage API from PC_2's domains. In the following, we describe the design principles and operation of this storage API for Personal Clouds.

4.3. Standard API for Personal Clouds: Storage, Sync & Sharing

To design a storage API that is flexible enough to be broadly adopted by heterogeneous Personal Clouds, we believe it is necessary to consider the main storage functionalities that distinguish these services. Concretely, most Personal Clouds make intensive use of *file storage, synchronization* and provide *sharing capabilities* on users' data [41; 11; 44].

Unfortunately, existing storage API standards such as CDMI fall short when it comes to satisfy these sophisticated Personal Cloud functionalities (e.g., sharing, syncing). This is mainly due to the fact that they were devised for storage management in IaaS providers. Therefore, to cope with these requirements, we propose a storage API tailored to Personal Clouds that is compliant with REST standards[7] (see Table 2).

First, we can find different Personal Cloud APIs for managing *storage and file synchronization*[8]. However, a common denominator among most existing APIs is that: i) They explicitly decouple data and metadata calls; and ii) They provide support to object versioning. Accordingly, in our API we differentiate data/metadata management calls to files and folders.

In addition, we provide an abstraction to manage file versions. As our API design considers a version as a *resource*, it enables separately managing metadata and versions of data, which represents an additional degree of freedom for synchronization algorithms. Clearly, Personal Clouds adopting our API can still operate with their own versioning mechanisms and policies, but exhibiting a common front-end from a user's

[6] https://github.com/cloudspaces/interop-protocol

[7] `https://github.com/stacksync/swift-API`

[8] `https://launchpad.net/ubuntuone-file-storage-api`

perspective. As discuss in Section 7, this opens the door to investigate novel inter-cloud synchronization algorithms that exploit interoperability.

Our API also provides a *file sharing interface*. In particular, `share` and `unshare` methods abstract the actions of enabling external users to access files contained in a given folder. Moreover, the API provides a method to control the membership of a shared folder (i.e. `members`).

All in all, we empirically demonstrate that our API is flexible enough to satisfy the storage, synchronization and sharing requirements of two Personal Clouds with very different architectures: NEC, which is based on WebDav following a client-server model, and StackSync, which fully decouples metadata instances in a separate layer that enables horizontal scalability. This gives a sense on the potential of our storage API for being adopted by other services.

In the following, we describe the reference implementation of `DataWings` in our use case cloud providers.

5. Implementation on Pilot Services

In this section, we briefly describe the pilot cloud services that already integrate `DataWings` to interoperate among them. Also, we provide details on the protocol implementation based on the interoperability model adopted by these services (i.e., client-side or horizontal.)

5.1. Interoperable Cloud Services: StackSync, NEC and eyeOS

We introduce the cloud services that already support our interoperability protocol. In particular, StackSync and NEC are the Personal Clouds that enable external users (of each other) accessing and exchanging data. Moreover, eyeOS will be an advanced use case of our interoperability protocol, being capable of transparently managing data in a real deployment of NEC and StackSync services.

StackSync: StackSync[9] is an open-source Personal Cloud with advanced file synchronization elasticity and security mechanisms [44]. StackSync explicitly decouples data —object storage with OpenStack Swift— and metadata processing. A core contribution of StackSync is to rely on a lightweight communication framework for providing programmatic elasticity to distributed objects using message queues as their underlying communication middleware (ObjectMQ[10]). ObjectMQ unicast and multicast communication primitives have considerably simplified the code of the synchronization protocol. It also enables efficient and transparent notification of file changes on top of the underlying messaging service.

StackSync provides a reference implementation and useful tools for rapid prototyping and evaluation. StackSync is a stable open source project after two years of development that is being used in several public institutions and data centers, being already available for desktop and mobile clients.

NEC Personal Cloud: NEC Cloud Storage[11] is a carrier oriented online storage platform integrated with multiple fixed and mobile devices for ubiquitous information access. NEC decouples data storage (OpensStack Swift) and metadata management, which is based on WebDav [45]. The service provides a battery of attractive features, ranging from synchronization, sharing and collaboration, support for mobile devices and an intelligent cache technology that stores on your local device the most commonly used files for flawless offline access. NEC Personal Cloud is being actively used by dozens of companies world-wide.

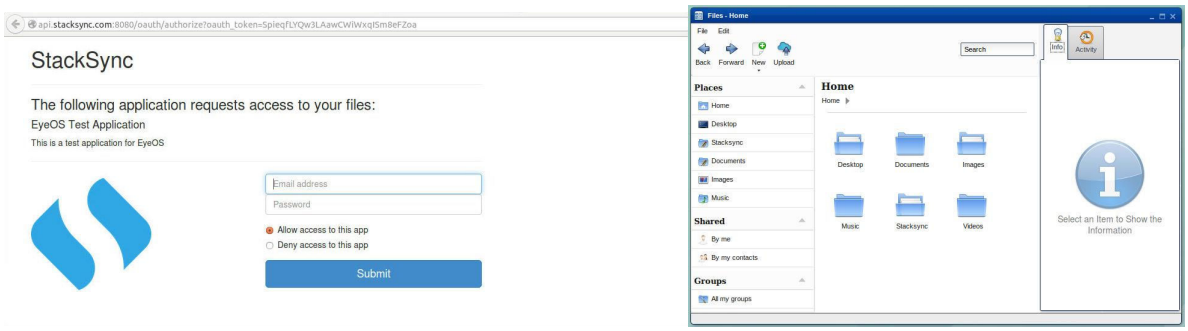
eyeOS Cloud Operating System: eyeOS[12] is a private-cloud application platform with a Web-based desktop interface. Commonly called a cloud desktop because of its unique user interface, eyeOS delivers a whole desktop from the cloud with file management, personal management information and collaborative tools, and with the integration of the client's applications. In a nutshell, the system's architecture is clearly divided into components that run on JavaScript and that run on PHP. Applications are divided into two parts: The part that runs on the client system (Web browser) and the part interpreted by PHP (Web server). The client side is used to render the user interface using JavaScript, and when an operation needs to be carried out, e.g., read the database, the client-side triggers a server-call to the part interpreted by PHP through an Ajax request.

[9] `http://stacksync.org/`

[10] `https://github.com/cloudspaces/objectmq`

[11] `http://www.nec.com/en/global/solutions/cloud/portfolio/storage.html`

[12] `http://www.eyeos.com/`



(a) Invitation acceptance.

(b) eyeOS cloud desktop integrating a StackSync folder.

Figure 4: Implementation of eyeOS. In this figure, an eyeOS user performed the credential workflow against StackSync to manage data seamlessly in eyeOS and StackSync.

5.2. *Client-side Interoperability in eyeOS cloud desktop*

eyeOS is a cloud desktop that embodies an advanced proof-of-concept of `DataWings`. eyeOS has integrated `DataWings` as a storage API in order to provide users with transparent access of their data stored at StackSync and NEC Personal Clouds. eyeOS is in charge of addressing the requests to the appropriate providers, depending on where data is living. As defined in our protocol, eyeOS adapted two main services to interoperate with StackSync and NEC: *Authentication* and *storage management*[13].

Accessing to interoperable Personal Clouds. For authenticating a Personal Cloud user via eyeOS, the process is as follows. First, the eyeOS platform uses OAuth authentication in order to interact with a user's protected data stored in a Personal Cloud. When a eyeOS user accesses the file manager for the first time, a newly developed plugin is used to get a security token that enables the interaction with the data stored in the corresponding Personal Cloud. More concretely, the *access token* and the *token secret* are stored in the "token" table of a relational database (MySQL) under the boundaries of eyeOS. These tokens are linked with the user who has logged into the platform, meaning that the system can determine the access tokens for a given user who attempts to use the service at any stage.

Once the user is identified within eyeOS, the credentials exchange for accessing a folder residing at an interoperable Personal Cloud (e.g., StackSync) is as follows. A user requests from StackSync the *consumer key* and *secret token* that identify eyeOS as the resource consumer. This communication is done via email. Then, a user gets the request token and provides StackSync with the redirect URL to eyeOS once the user grants authorization. StackSync responds to the previous request by giving a valid *request token* and an *authorization URL*. The user is then redirected to the authorization URL where he may grant eyeOS access to his private space (see Fig. 4a). Once StackSync verifies the user, it redirects the user to the eyeOS URL provided in the previous step. Last, the user get the access token and token secret from StackSync, with which eyeOS will identify itself when accessing the user's private space in the Personal Cloud[14].

Managing external folders. As users finish the credentials exchange between eyeOS and the targeted Personal Cloud, they can then use eyeOS's Web file manager and all its features with their Personal Cloud files through the `DataWings` API. For example, users can display online their documents saved in their Personal Cloud, create directories, move files, share documents, etc. Users access the files in their Personal Cloud using a special folder labeled accordingly. For instance, in Fig. 4b we can see how a user in eyeOS has a "StackSync folder" that represents his own data at this Personal Cloud.

To get the file and directory structure of a Personal Cloud, a call is made to `DataWings`'s API. This call returns metadata with all the structural information of the files and directories, which eyeOS uses to

[13] `https://github.com/cloudspaces/eyeos-u1db\#implementation-of-stacksync-api-into-eyeos`

[14] The complete description of the process can be found at `http://cloudspaces.eu/deliverables/doc_download/55-d5-2-service-platform-reference-prototype`

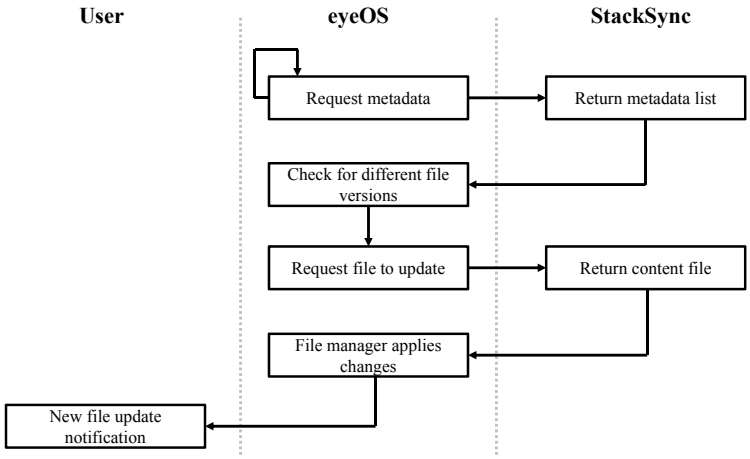


Figure 5: eyeOS file manager synchronization to obtain updates from StackSync folders.

generate a local view of the file system structure (without contents). When the user selects an element and performs an operation on it, i.e. she opens, moves or copies it, the element's contents are then downloaded. By doing this, the system is not overloaded unnecessarily by retrieving information that the user will not use at that moment. If the content of a file or directory has already been retrieved and there are no changes regarding the remote copy, it will not be updated (i.e., local cache).

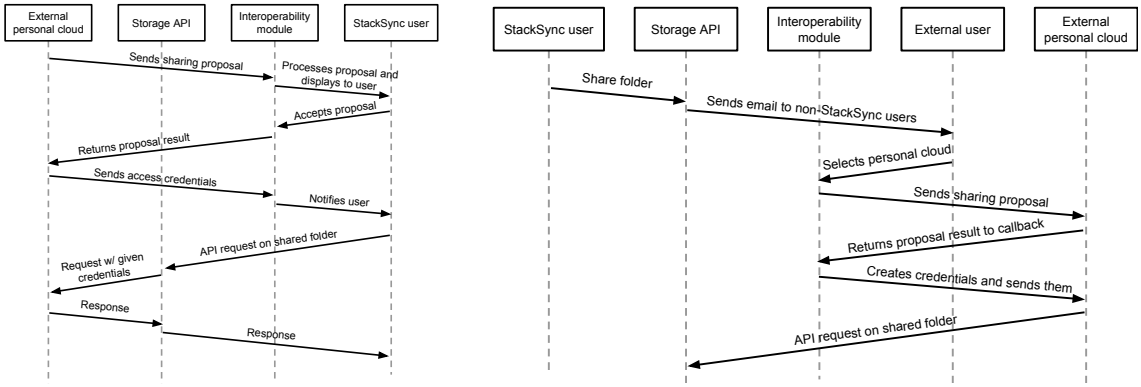
The eyeOS file manager can also retrieve previous versions of a file by exploiting calls related to *object versions* in the `DataWings`'s API. That is, eyeOS shows a list of all the available versions of a file, letting the user retrieve the contents of the desired version. If the user makes changes to a previous version, when those changes are saved, a new version is created in the Personal Cloud. The contents of the current directory are synced with the Personal Cloud directory in a background process, which sends queries every 10 seconds to check whether there are any changes (i.e., pull-based). If there are any changes, the current data structure is updated. This process can be observed in Fig. 5.

5.3. Horizontal interoperability: NEC and StackSync

Sharing folders with external users. Both NEC and StackSync Personal Cloud systems have implemented the complete protocol and have passed conformance tests[15] (i.e., authentication and storage API). To implement authentication, internally both systems *create a new user* upon the arrival of a share request of an external user, which is identified by the token included in the request. In other words, to provide access to an external user, these Personal Clouds transparently provide a new internal identity to him/her for enforcing control and security mechanisms. This approach provides two advantages: First, the external user credentials are not exposed in any request, and second, the user created to authorize external access is subject to the same control and security mechanisms that regular users in each Personal Cloud. Similarly to eyeOS, a sequence of graphical Web interfaces ease users to get external access to these services through the credential exchange process. As both systems exhibit a similar implementation, in the following we describe the StackSync implementation of `DataWings` authentication protocol (see Fig. 6).

As defined in the protocol, StackSync implements three endpoints: One to receive share proposals from external Personal Clouds; another to receive cancellations of already-established sharing agreements; and the last one to receive access credentials for accepted sharing proposals. To this end, we created a Django module (i.e., *interoperability module*) to embody the functionality of these end-points. This new module can be located either with the synchronization service or in an independent server.

[15] `https://en.wikipedia.org/wiki/Conformance_testing`



(a) StackSync user requesting access to an external Personal Cloud.

(b) External user requesting access to StackSync.

Figure 6: Credentials exchange workflow in the implementation of StackSync.

The sequence in Fig. 6a depicts a scenario where StackSync receives an external sharing proposal from a Personal Cloud that already implements the interoperability protocol such as NEC. The entry point for the request is the interoperability module. It will receive the sharing proposal and check that it is well formed. Afterwards, the invitee, which is a StackSync user, will be shown the proposal details in a graphical Web interface similar to Fig. 4a, where she will either accept it or decline it. The result is then forwarded to the source Personal Cloud, which will reply with the access credentials in case the user accepted the proposal. These credentials are assigned to the recently shared folder and saved into the database. At this time, the interoperability process is completed and the StackSync user is notified about its newly shared folder.

Conversely, Fig. 6b illustrates the situation where a sharing proposal is generated by a user in StackSync. In this case, the interoperability module receives the request and sends an email to the invitee, which is a user belonging to an external Personal Cloud that also implements the interoperability protocol. The invitee is forwarded to a Website located in StackSync where she will select her Personal Cloud from a list of all compatible services.

As soon as the user selects the Personal Cloud service, she will be redirected to a Website located on her Personal Cloud where she will be shown the details of the folder and will be prompted to accept or deny the proposal. StackSync will get notified of the user's choice and will hand the access credentials (i.e., OAuth tokens) to the other Personal Cloud. Afterwards, whenever the invitee wants to access to the shared folder, the StackSync API will receive the request and process it as any other API call.

Adapting the Personal Cloud to handle external folders. The storage API has been implemented by both systems (StackSync in Python, NEC in C#) and it enables horizontal interoperability between providers. The API permits users and applications to manage folders and files in both systems, either being primary users or having external access. Moreover, in StackSync, the API currently serves both desktop clients and Web/mobile clients, which gives a sense of its capabilities.

Retaking the example of StackSync, once a user has access to an interoperable external folder, the metadata back-end should be ready for handling metadata operations upon the execution of API calls. For instance, when the StackSync Web or mobile application shows all the folders of a user, it executes an API operation to retrieve the metadata of these folders (see Algorithm 1). However, as external and native folders reside in different locations, both types of folders need to handled in different ways. In the back-end implementation of the `DataWings` API, this aspect should be strongly considered.

For instance, a naive approach could be to fetch all the metadata of both native and external folders when the metadata of a user is requested. However, the problem of this approach is that the exchange of metadata messages between Personal Cloud services may be too high, inducing high latency on the operation's execution. This is specially true considering that most of that metadata requested in this kind of operations is not used by end-users.

Algorithm 1: Get metadata of a user's root folder in StackSync.

```
Data: User user
Result: APIGetMetadata
/* Result variables                                                    */
ItemMetadata responseObject = null;
List<ExternalFolderMetadata> externalFolders = null;
Integer errorCode = 0;
Boolean success = false;
String description = "";
try {
  /* First, retrieve metadata from the root folder                     */
  ItemMetadata responseObject = this.itemDao.findByUserId(user.getId(), false);
  /* Then, retrieve metadata from folders in external Personal Clouds  */
  List<ExternalFolderMetadata> externalFolders = this.itemDao.getExternalFolders(user.getId());
  success = true;
}catch(DAOException e) {
  description = e.getError().getMessage();
  errorCode = e.getError().getCode();
  logger.error(e.toString(), e);
}
APIGetMetadata response = new APIGetMetadata(responseObject, success, errorCode, description,
externalFolders);
return response;
```

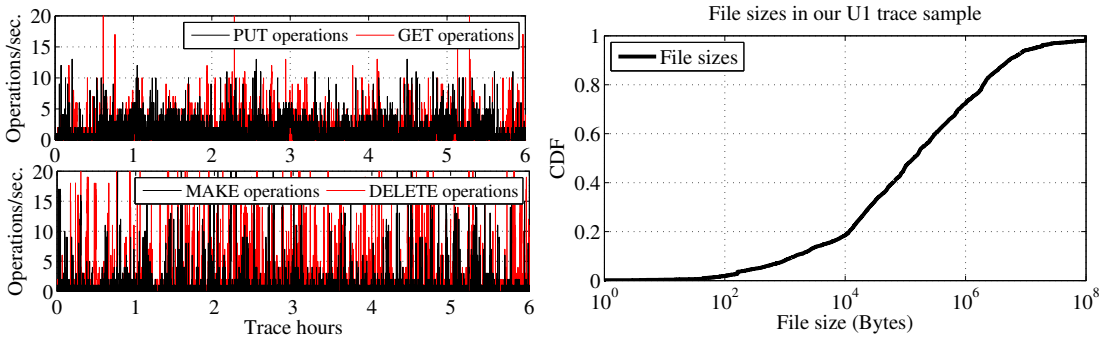
For this reason, in StackSync we have adopted an alternative implementation. As can be observed in Algorithm 1, StackSync retrieves the metadata of files and folders depending whether they are native or interoperable ones. On the one hand, all the metadata corresponding to native files and folders is available in the metadata store. On the other hand, StackSync only stores the name, URL and access tokens of external folders, which are also retrieved in Algorithm 1. With this optimization, StackSync Web or mobile applications can efficiently render a preliminary view of a user's folders without performing requests to external Personal Cloud providers.

Of course, to obtain the complete metadata of external files and folders in StackSync, it is necessary an additional call on-demand that retrieves that information from the external Personal Cloud. This is possible thanks to the external folder metadata stored in StackSync (i.e., URL, name and tokens). Similarly, whenever the user aims at retrieving or uploading data to the shared external folder, the API will seamlessly detect the special condition of the folder and forward the request to the origin Personal Cloud with the given credentials. This process is performed in a transparent manner for the user.

Once described the implementation of `DataWings` by real use-case services, we proceed to demonstrate the feasibility of our protocol. To this end, we present a deployment and extensive validation of Personal Clouds implementing `DataWings`.

6. Experiences with DataWings

We evaluate a real deployment of our interoperability protocol in two real-world Personal Clouds: NEC and StackSync. Inspired by the eyeOS use-case, we evaluate our `DataWings` deployment from a *client-side* perspective; this is the model used by eyeOS cloud desktop to offer an integrated view of user files stored at multiple Personal Clouds. The objective of this evaluation is twofold: First, to ensure the correct implementation of the complete `DataWings` protocol (authentication, storage API). Second, we aim at analyzing the performance implications that distinct types of user activity (e.g., download/upload oriented) or the underlying sharing network topology have in these providers [40; 46]. Note that the evaluation of how



(a) Workload executed in our experiments.

(b) File sizes in our trace.

Figure 7: Workload and file sizes of the U1 users selected to build our traces (6 hours).

different types of user activity, jointly with their social relationships, impact on the resource consumption of interoperable Personal Clouds has not yet been studied in the literature.

6.1. Experimental Scenario

Workload. The workload used to evaluate the system consists of replaying traces of a real-world Personal Cloud, namely UbuntuOne (U1) [11]. Concretely, we executed a trace consisting of almost 1K users (983) performing storage (31.9K) and metadata (31.6K) operations for 6 hours (14/02/2014). As visible in Fig. 7a, we mapped the activity of U1 users to perform 4 types of operations: `make` (create a file/directory metadata entry), `put` (upload file content), `get` (download file content) and `delete` (erase a file/directory). Moreover, Fig. 7b points out that the files sizes in the trace tend to be small (e.g., 72.3% of files are $<$ 1MB), as reported by several Personal Cloud measurements [41; 11].

To replay this trace, we developed a workload generator in Python that integrates a client for `DataWings` API and manages the credentials of both owned and shared folders across Personal Clouds. Note that the access credentials for both owned and shared folders were created once beforehand via our authentication protocol, and they were reused multiple times for each experiment execution. In our workload generator, each request to the storage API was executed on a separate thread to enable parallelism and to avoid that failed requests could delay the trace replay.

Moreover, we wanted to observe the impact of different types of user behavior on the system. To this end, following the guidelines of Drago et al. [41], we identified the users that exhibited more than three orders of magnitude of difference between upload and download (e.g., 1GB versus 1MB) traffic. We classified them as either download-only or upload-only. Thus, from the total of users that appear in our trace replay, 581 are upload-only users and 402 are download-only users.

Moreover, sharing interactions among users are defined by a social network, as we describe next. In particular, each user in the workload trace represents a node in the social graph. In our experiments, we balanced both the number of users and their types in StackSync and NEC. Users were randomly distributed across the social graph. Thus, before executing a storage operation, the execution thread performed a weighted decision among the available neighbors based on the number of storage interactions available in the original social graph.

Sharing network topology. Modeling data sharing across users requires from a realistic description of how users share data in a Personal Cloud. Unfortunately, today there are no public traces of file sharing in a Personal Cloud at large scale, since it would require to access the metadata layer of the system.

For this reason, we made use of the sharing network existing in the NEC Personal Cloud [47]. We built a trace containing the sharing interactions across users that also described the characteristics of the files being shared. Concretely, this trace contained the sharing interactions of 10K users during 2 years.

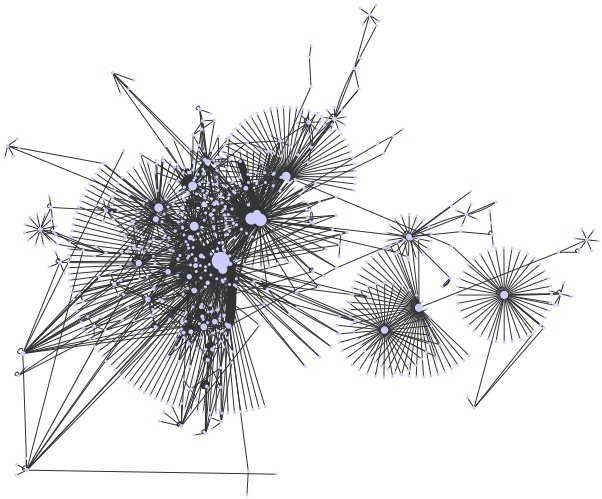


Figure 8: Social graph of NEC Personal Cloud used in our evaluation to model sharing interactions among users. The size of nodes represent their degree, whereas the width of edges is proportional to sharing activity between two users —i.e., the number of storage operations of a user a executed on a shared folder owned by user b.

As a contribution of this work, we make this trace publicly available[16]. The social network used in our experiments is depicted in Fig. 8.

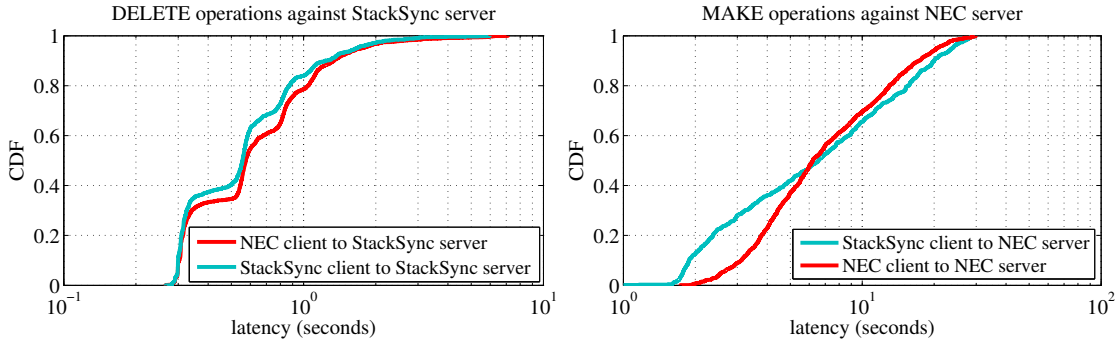
In Fig. 8, nodes represent users in the NEC Personal Cloud and edges represent sharing interactions among them. Furthermore, the size of nodes is proportional to the degree of the node, and the width of edges is related to the strength of the sharing interaction among two nodes (i.e., the number of storage operations that a user a performed on b's shared folder). As we can observe, there are few users that account for a large number of links, whereas most users are only linked with one user. Moreover, despite that there exists some correlation between the degree and the number of sharing interactions, we can observe nodes with few but very active links. All these effects will be subject of further analysis and may impact on the performance of our storage API.

Testbed. To run our experiments, we employed two different platforms for StackSync and NEC. This allowed us to explore the behavior of our protocol implementation not only in disparate Personal Cloud services, but also in heterogeneous hardware.

For StackSync, we used a 9-machine *cluster* formed by 2 compute nodes (2x16GB RAM, 2x1TB 7.2krpm HDD) and 6 storage nodes (4 cores, 12GB RAM, 1TB HDD), plus 1 large node that acted as a proxy(6 cores, 16GB RAM, 2x600GB 10krpm HDD). Compute nodes ran the StackSync metadata service (sync-servers), whereas storage nodes supported a deployment of OpenStack Swift Kilo. Moreover, the proxy node was the machine running the `DataWings` API service. Machines were interconnected via 1Gbit switched network links.

The interoperability prototype of NEC was executed on VMWare virtual machines running in 2 servers (4-core server, 8GB RAM DDR3). One server was intended to run the metadata service (SQL server database, WebDav service), including the interoperability API. The other server executed the graph database used in NEC to manage sharing requests among users (Titan DB). The storage back-end was OpenStack Swift, which was running in Tissat[17], an IaaS provider in the CloudSpaces project. Obviously, the physical distance between the metadata layer that processes all the request and the storage layer negatively impacted the performance.

[16] `http://cloudspaces.eu/results/datasets`
[17] `http://www.tissat.es`



(a) DELETE operations

(b) MAKE operations

Figure 9: Latency comparison of metadata operations executed via the native API and `DataWings` against NEC and StackSync.

6.2. Results

Feasibility of `DataWings`. In what follows, we want to evaluate: i) `DataWings` has been correctly implemented in both Personal Clouds; ii) The client-side interactions of users with a Personal Cloud via `DataWings` do not exhibit additional performance overhead compared to the native API.

First, we clearly observe in Fig. 9 that the clients of our workload generator were able of executing storage operations against both StackSync and NEC. Interestingly, users could make use of `DataWings` to access multiple Personal Clouds. This demonstrates that the `DataWings` API was correctly implemented and deployed in the two Personal Cloud services. Also, both StackSync and NEC also correctly ran the credentials exchange protocol: Workload clients were registered at only one of the Personal Clouds, which means that accessing to the other one via `DataWings` was possible thanks to a prior execution of the credential exchange protocol. We conclude that use-cases such as eyeOS, or even end-users, can now benefit from transparent access to their files stored across several vendors with `DataWings`.

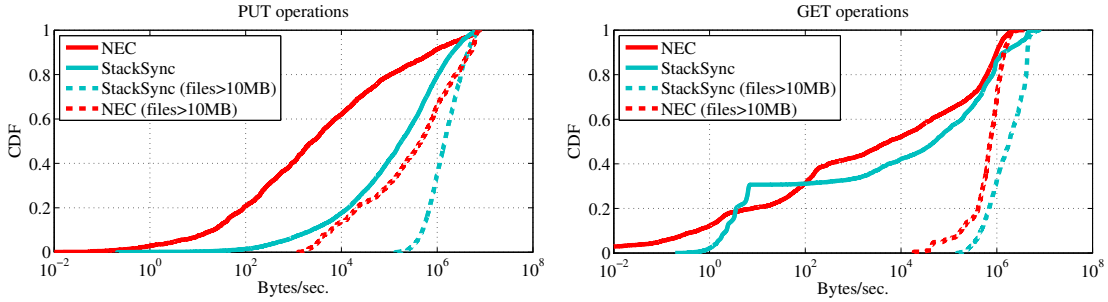
Second, we wanted to analyze if using `DataWings` to manage files induces additional performance overhead from a client's perspective. In this sense, Fig. 9 illustrates the latency of `MAKE` and `DELETE` operations depending on the Personal Cloud to which the client belongs and the Personal Cloud that handles the request. As visible in the Fig. 9a, NEC and StackSync users present similar latency values when deleting files from the StackSync server. StackSync clients can access the service via the native API, whereas NEC users resort to the credentials acquired via `DataWings` to interact with StackSync. A similar result can be observed in Fig. 9b if we consider `MAKE` operations against NEC servers. In this case, our experiments show that NEC clients exhibit a median latency 5.47% higher than StackSync clients when creating files on NEC servers. Therefore, implementing client-side interoperability with `DataWings` does not impose significant performance penalty.

In the case of implementing horizontal interoperability, however, there will be additional performance penalties for clients. The reason is that the user's Personal Cloud will have to execute storage operations against another service to transparently manage her external files. However, as we show next, such penalty is mainly dependent on the actual performance of the interoperating Personal Cloud systems.

API performance under heterogeneity. First, we aim at analyzing the performance of the storage APIs implemented by NEC and StackSync, for both data and metadata requests. Our objective is to compare how heterogeneous providers and workloads impact on the performance of user operations.

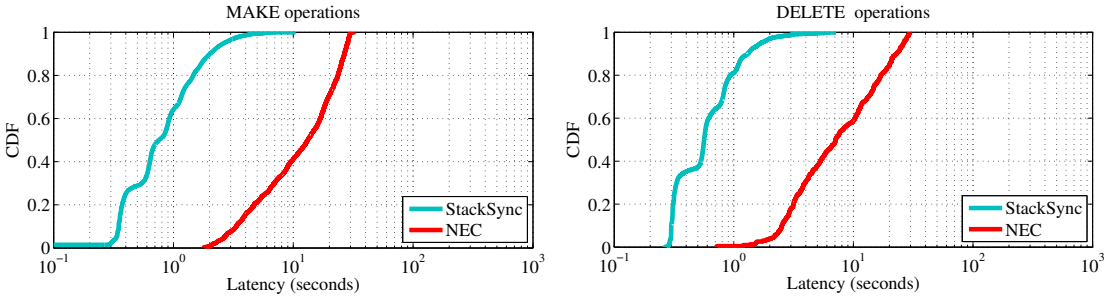
First, Fig. 10 shows the performance of data transfers against the storage API implementation of StackSync and NEC. At first glance, we observe that in most cases data transfers are limited, specially in the case of NEC. For instance, only 20% of file transfers exhibit a mean transfer speed $>$ 1MBps.

One of the main reasons behind this phenomenon is that most file transfers (PUTs, GETs) are related to very small files. To inform this argument, 72.3% of storage operations in our workload belong to files $<$



(a) PUT operations (b) GET operations

Figure 10: Storage operations throughput of StackSync and NEC via our storage API for interoperability.



(a) MAKE operations (b) DELETE operations

Figure 11: Metadata operations latency of StackSync and NEC via our storage API for interoperability.

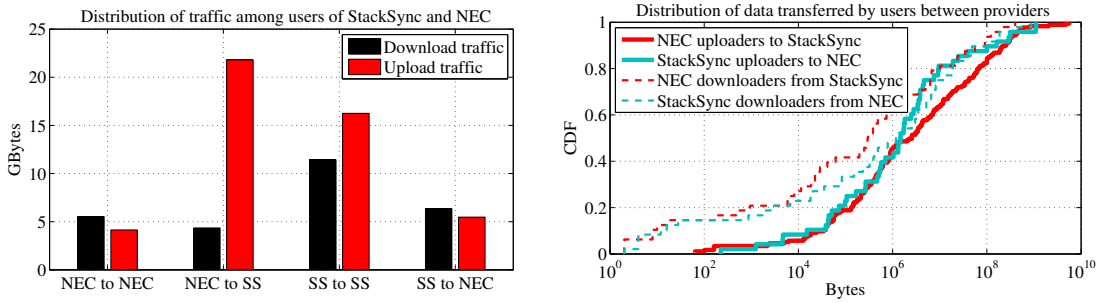
1MB. Clearly, the communication and metadata management overhead of API calls is significant compared to the time needed for transferring tiny files. All in all, as we can also observe in Fig. 10, if we focus on the transfer performance of large files ($>$ 10MB), the performance of transfers is significantly better. Further, the transfer performance of large files is slightly faster for `GET` operations than for `PUT`s. This occurs because writing data can be slightly more costly than reading it in some situations (e.g., concurrency, mixed workloads); for instance, OpenStack Swift —used by StackSync as storage back-end— waits for the data object to be correctly written to at least 2 disks before considering the `PUT` operation as successful.

Second, in Fig. 11 we illustrate the performance of metadata operations (i.e., API operations that do not involve data transfers). As we can observe, StackSync significantly outperforms the NEC storage API deployment. To wit, around 60% of successful delete operations and 40% of successful file create operations (i.e., MAKE) against the NEC deployment take more than 10 seconds to complete, respectively. Furthermore, some of the worst case metadata operation completion times are over 30 seconds, which seems too high for an operation that does not involve data transfers. We identified two main reasons for the poor performance of the NEC deployment: i) The fact that the storage back-end of NEC is located in another datacenter makes the communication between the metadata servers and the storage layer slower, ii) The commodity hardware used to deploy the interoperability prototype was not sufficient, specially considering that it was running in a virtualized environment. Conversely, the StackSync metadata service and its data store were deployed (bare metal) in a single and larger cluster, offering a much higher metadata operation performance (e.g., 90% of delete operations take less than 1 second to complete). These results support our initial intuition: In the case of implementing horizontal interoperability, the heterogeneity of Personal Cloud deployments may be an important factor that impacts on storage performance.

We are also interested on analyzing the storage API failures occurred in our experiments. In Table 3, we

	StackSync	NEC
All operations	47,655 (74.27%)	17,383 (26.73%)
Successful Data operations	23,188 (35.65%)	8,251 (12.69%)
Successful Metadata operations	23,936 (36.80%)	7,746 (11, 91%)
Failed GETs	0 (0.0%)	104 (0.16%)
Failed PUTs	0 (0.0%)	114 (0.18%)
Failed MAKEs	1 (0.001%)	194 (0.30%)
Failed DELETEs	2 (0.003%)	37 (0.06%)

Table 3: Storage and metadata operations performed during our workload (percentages are w.r.t. the total amount of operations).



(a) PUT operations

(b) GET operations

Figure 12: Implications of the social topology and user activity on the traffic exchanged between Personal Cloud services.

provide a summary of the API calls performed by both Personal Clouds, as well as the absolute percentage of failed/successful calls. As visible in Table 3, StackSync exhibits much higher reliability than NEC. In fact, the failures in StackSync represent a $\approx 0.004\%$ of the total, whereas for NEC this percentage is 0.7%. Despite that the number of failures is still limited, the low performance of the NEC deployment may be an important source of failure.

In conclusion, both StackSync and NEC enable users to share data across them. Moreover, the implementation and deployment of StackSync offers higher performance than NEC, which provide us with guidelines to the proper development of REST API for Personal Clouds.

Implications of social topology and user activity. In the following, we inspect the role that the sharing interactions defined by the social graph and the types of user activity play on this scenario. For this reason, Fig. 12 shows the amount of traffic exchanged between Personal Clouds and the distribution of data transfers from external users.

First, in Fig. 12a we illustrate the traffic exchanged among users depending on the origin and destination provider. Clearly, there are profound differences in the amounts of in/out traffic of both Personal Clouds. In our workload, we clearly see that NEC users stored almost 4X more data than StackSync users did on NEC. Similarly, StackSync users consumed 31% more download bandwidth that NEC uses from StackSync. The reasons underlying the asymmetries of inter-cloud traffic consumption are related with the activity of users and the structure of their social interactions.

That is, the amount of download and upload traffic depends on whether users tend to use Personal Clouds for storage (upload-only) or for content distribution (download-only). Furthermore, the individual activity rate of users in a Personal Cloud is also very important. In Fig. 12b, we observe that a small fraction of users consumed most of the traffic [11], causing a significant part of the inter-cloud traffic asymmetries.

For instance, 10% of NEC users account for 83.95% of upload traffic to StackSync shared folders, whereas 5% of StackSync users consume the 68.41% of NEC download traffic.

Moreover, jointly with the activity of users, the social graph plays a critical role on the traffic exchanges across Personal Clouds. To wit, very active users exhibiting strong data sharing interactions with users of other Personal Clouds are the most important factor of inter-cloud traffic exchanges. This, in turn, suggests that Personal Cloud providers should specifically take into account the activity of external and very active users, as they may represent an important source of resource consumption [42].

Although our experiments do not simultaneously replay workload sand sharing networks from different user communities, they give a sense on the potential implications that the activity and interactions of users may have on the traffic exchanged across interoperable Personal Clouds. We conclude that Personal Clouds may need additional mechanisms to control the amount of resources consumed by external users in an scenario with horizontal interoperability among providers.

7. Discussion and Conclusions

Lessons learned. In this work, we presented the first interoperability protocol specifically tailored to Personal Clouds, including a credential management protocol and a storage API. One of the main advantages of our protocol design is the multiple deployment strategies that it offers. For instance, two Personal Clouds may opt by integrating the protocol in their back-end, providing horizontal data exchanges across them. On the other hand, external services may use our protocol to provide transparent access to Personal Cloud data to their own users; we showed an example of this deployment strategy in eyeOS, a cloud operating system that could access to shared folders from both StackSync and NEC.

Moreover, we found that our interoperability protocol is flexible enough to satisfy the storage, synchronization and sharing requirements of heterogeneous Personal Clouds. This is specially true in the case of StackSync and NEC: NEC exhibits a client-server architecture based on WebDav, whereas StackSync resorts to elastic metadata servers totally decoupled from the storage layer. From a technical perspective, such a level of flexibility may encourage other systems, such as onwCloud[18] or big players like Dropbox, to adhere to this protocol with little impact on their running services.

Clearly, apart from technical considerations, incentivinzing existing Personal Clouds to adopt interoperability mechanisms requires the leadership of legal and standardization authorities, as big players may be in a situation in which locked-in customers are a source of profit. But apparently, this horizon seems to not be so far; to inform this argument, recent initiatives such as OpenCloudMesh[19] advocate for building an open federation of Personal Clouds with a unified API that enables data sharing and synchronization services. We believe that the spirit of our work is very aligned with these new kind of initiatives. Moreover, `DataWings` paves the way for leveraging practical Personal Cloud interoperability, which may be object of legislation and standardization in the future.

Beyond Personal Cloud interoperability. Our experiments with StackSync and NEC provided new insights beyond the operation of the interoperability protocol itself.

First, an scenario in which various Personal Clouds provide horizontal interoperability represents a substrate for data management optimizations. For example, one can imagine a user in a Personal Cloud that updates frequently a shared file stored in an external Personal Cloud. In this case, instead of transferring the whole file in each update, Personal Clouds may provide an inter-cloud synchronization mechanism on top of `DataWings`. In particular, interoperable providers may implement chunk-level synchronization, which may greatly reduce traffic exchanges between clouds.

Moreover, we found that active external users may represent a significant fraction of the consumed traffic and storage resources regarding interoperable Personal Clouds. Interestingly, this situation can be alleviated in several ways that are currently unexplored in this scenario. For example, one can imagine

[18] `https://owncloud.org`

[19] `https://oc.owncloud.com/opencloudmesh.html`

specific payment models or agreements among Personal Clouds to balance the resource consumption of external users. On the other hand, novel control mechanisms may arise to enable external user access, but detecting and limiting potential abuses from malicious users.

Conclusions. Today, Personal Cloud services are large data silos that aim at retaining users by offering an all-in-one storage solution. However, the unintended consequence of this model is that users lack from transparent data sharing among multiple providers. This may lead to a vendor lock-in situation in which users get trapped in a single provider due to the cost and complexity of switching to another one.

To solve these problems, we presented `DataWings`: A protocol consisting of a *credential management protocol* and a *storage API* to enable interoperability among Personal Clouds. The protocol enables users to store, synchronize and share data transparently from shared folders in external providers. In turn, the agreement effort for existing Personal Clouds to integrate `DataWings` is affordable: i) Internally, providers can keep their own authentication service, and ii) The storage API covers the fundamental functionality of most existing services and follows the current open standards already adopted by Personal Clouds.

We demonstrated our claims by implementing `DataWings` on heterogeneous Personal Cloud services (NEC, StackSync). We also described different deployment strategies possible with `DataWings`, including the case of eyeOS. Specifically, eyeOS implements our protocol to make it transparent for users accessing data stored either at NEC or StackSync, being a real use case of `DataWings`.

Furthermore, we provided a complete evaluation of our prototype implementations in a real multi-provider deployment (NEC, StackSync). We analyzed the storage API performance on both services, as well as the implications that the activity of users and their social interactions have on interoperable Personal Clouds. The latter aspect has not yet been studied in the literature, and opens the door to the design of novel control and optimization mechanisms in this scenario.

Acknowledgment

This work has been partly funded by the European Union through project FP7 CloudSpaces (317555) and H2020 IOStack (644182) and by the Spanish Ministry of Science and Innovation through project "Cloud Services and Community Clouds" (TIN2013-47245-C2-2-R).

References

[1] F. Research, The personal cloud: Transforming personal computing, mobile, and web markets (2011).
URL `http://www.forrester.com/rb/Research/personal_cloud_transforming_personal_computing\%2C_mobile\%2C_and/q/id/57403/t/2`

[2] F. Research, Pioneer vendors: Personal cloud (2014).
URL `https://www.forrester.com/report/Pioneer+Vendors+Personal+Cloud/-/E-RES105441`

[3] Dropbox, Celebrating half a billion users, https://blogs.dropbox.com/dropbox/2016/03/500-million (2016).

[4] T. Economist, Battle of the clouds, http://www.economist.com/node/14644393 (2009).

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4).

[6] R. Cowan, Tortoises and hares: choice among technologies of unknown merit, The economic journal (1991) 801–814.

[7] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, M. Kunze, Cloud federation, in: IEEE CLOUD'11, 2011.

[8] R. Gracia-Tinedo, M. S. Artigas, A. Moreno-Martınez, C. Cotes, P. G. López, Actively measuring personal cloud storage, in: IEEE CLOUD'13, 2013, pp. 301–308.

[9] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, S. Dustdar, Winds of change: From vendor lock-in to the meta cloud, IEEE Internet Computing (1) (2013) 69–73.

[10] E. Hammer-Lahav, The OAuth 1.0 Protocol, `http://tools.ietf.org/html/rfc5849` (2010).

[11] R. Gracia-Tinedo, Y. Tian, J. Sampé, H. Harkous, J. Lenton, P. García-López, M. Sánchez-Artigas, M. Vukolic, Dissecting UbuntuOne: Autopsy of a Global-scale Personal Cloud Back-end, in: ACM IMC'15, 2015.

[12] ENISA, Cloud computing-benefits, risks and recommendations for information security, https://www.enisa.europa.eu/activities/risk-management/files/deliverables/cloud-computing-risk-assessment/ (2009).

[13] SNIA, Cloud data management interface (cdmi), http://www.snia.org/cdmi.

[14] G. Vernik, A. Shulman-Peleg, S. Dippl, C. Formisano, M. C. Jaeger, E. K. Kolodner, M. Villari, Data on-boarding in federated storage clouds, in: IEEE CLOUD'13, 2013, pp. 244–251.

[15] K. Aberer, P. Cudré-Mauroux, A. M. Ouksel, T. Catarci, M.-S. Hacid, A. Illarramendi, V. Kashyap, M. Mecella, E. Mena, E. J. Neuhold, et al., Emergent semantics principles and issues, in: Database Systems for Advanced Applications, 2004, pp. 25–38.

[16] E. M. Maximilien, A. Ranabahu, R. Engehausen, L. C. Anderson, Toward cloud-agnostic middlewares, in: ACM SIGPLAN'09, 2009, pp. 619–626.
[17] G. Vetere, M. Lenzerini, Models for semantic interoperability in service-oriented architectures, IBM Systems Journal 44 (4) (2005) 887–903.
[18] L. Steels, P. Hanappe, Interoperability through emergent semantics: a semiotic dynamics approach, in: Journal on Data Semantics, 2006, pp. 143–167.
[19] Vision cloud project, http://www.visioncloud.eu/.
[20] mosaic project, http://www.mosaic-fp7.eu/.
[21] Remics project, http://www.remics.eu/.
[22] S. Sotiriadis, N. Bessis, An inter-cloud bridge system for heterogeneous cloud platforms, Elsevier Future Generation Computer Systems 54 (2015) 180–194.
[23] Open cloud computing interface, http://occi-wg.org/.
[24] X.-L. Liu, R.-K. Sheu, S.-M. Yuan, Y.-N. Wang, A file-deduplicated private cloud storage service with cdmi standard, Elsevier Computer Standards & Interfaces 44 (2016) 18–27.
[25] D. Heimbigner, D. McLeod, A federated architecture for information management, ACM Transactions on Information Systems (TOIS) 3 (3) (1985) 253–278.
[26] Q. Jia, Z. Shen, W. Song, R. van Renesse, H. Weatherspoon, Supercloud: Opportunities and challenges, ACM SIGOPS Operating Systems Review 49 (1) (2015) 137–141.
[27] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, et al., The reservoir model and architecture for open federated cloud computing, IBM Journal of Research and Development 53 (4) (2009) 4–1.
[28] D. C. Erdil, Autonomic cloud resource sharing for intercloud federations, Elsevier Future Generation Computer Systems 29 (7) (2013) 1700–1708.
[29] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. M. Sadjadi, M. Parashar, Cloud federation in a layered service model, Elsevier Journal of Computer and System Sciences 78 (5) (2012) 1330–1344.
[30] E. Olden, Architecting a cloud-scale identity fabric, Computer (3) (2011) 52–59.
[31] A. Celesti, F. Tusa, M. Villari, A. Puliafito, Three-phase cross-cloud federation model: The cloud sso authentication, in: IEEE AFIN'10, 2010, pp. 94–101.
[32] H. Y. Huang, B. Wang, X. X. Liu, J. M. Xu, Identity federation broker for service cloud, in: IEEE ICSS'10, 2010, pp. 115–120.
[33] A. Tassanaviboon, G. Gong, Oauth and abe based authorization in semi-trusted cloud computing: aauth, in: ACM international workshop on Data intensive computing in the clouds, 2011, pp. 41–50.
[34] H. Abu-Libdeh, L. Princehouse, H. Weatherspoon, Racs: a case for cloud storage diversity, in: ACM SoCC '10, 2010, pp. 229–240.
[35] A. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa, Depsky: dependable and secure storage in a cloud-of-clouds, ACM Transactions on Storage (TOS) 9 (4) (2013) 12.
[36] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. Anderson, D. Wetherall, Metasync: File synchronization across multiple untrusted storage services, in: USENIX ATC'15, 2015, pp. 83–95.
[37] D. Dobre, P. Viotti, M. Vukolić, Hybris: Robust hybrid cloud storage, in: ACM SoCC'14, 2014, pp. 1–14.
[38] Y. Hu, H. C. Chen, P. P. Lee, Y. Tang, Nccloud: applying network coding for the storage repair in a cloud-of-clouds., in: USENIX FAST'12, 2012, p. 21.
[39] R. Tornyai, A. Kertesz, Towards autonomous data sharing across personal clouds, in: Euro-Par'14 Workshops, 2014, pp. 50–61.
[40] R. Gracia-Tinedo, M. Sánchez-Artigas, A. Ramírez, A. Moreno-Martínez, X. León, P. García-López, Giving form to social cloud storage through experimentation: Issues and insights, Elsevier Future Generation Computer Systems 40 (2014) 1–16.
[41] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, A. Pras, Inside dropbox: understanding personal cloud storage services, in: ACM IMC'12, 2012, pp. 481–494.
[42] R. Gracia-Tinedo, M. Sanchez Artigas, P. Garcia Lopez, Cloud-as-a-gift: Effectively exploiting personal cloud free accounts via rest apis, in: IEEE CLOUD'13, 2013, pp. 621–628.
[43] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory, L. Yerushalmi, Towards an object store, in: IEEE MSST'03, 2003, pp. 165–176.
[44] P. Garcia-Lopez, M. Sanchez-Artigas, S. Toda, C. Cotes, J. Lenton, Stacksync: Bringing elasticity to dropbox-like file synchronization, in: ACM/IFIP/USENIX Middleware'14, 2014, pp. 49–60.
[45] E. J. Whitehead Jr, M. Wiggins, Webdav: Ieft standard for collaborative authoring on the web, IEEE Internet Computing 2 (5) (1998) 34–40.
[46] E. Bocchi, I. Drago, M. Mellia, Personal cloud storage: Usage, performance and impact of terminals, in: IEEE CloudNet'15, 2015, pp. 106–111.
[47] R. Gracia-Tinedo, P. García-López, A. Gómez, A. Illana, Understanding data sharing in private personal clouds, in: IEEE CLOUD'16, 2016, p. In press.