Building Universal Profile Systems over a Peer-to-Peer Network

Shi-Cho Cha
csc@mba.ntu.edu.tw

Yuh-Jzer Joung
joung@ccms.ntu.edu.tw

Yu-En Lue
eric@yuenlue.com

Department of Information Management
National Taiwan University
Taipei, Taiwan

Abstract

We propose Personal Data Backbone (PDB) to provide universal profile services over peer-to-peer networks. The main objective is to bring the control of personal data back to their owners. By using peer-to-peer technology, people can collaborate with one another to establish the services without resorting to a centralized mechanism or corporation, thereby removing concerns such as privacy, security, and monopoly. The peer-to-peer technology also achieves better trust, availability, accountability, and reliability, as compared to the centralized ones.

1 Introduction

Many information services and applications need personal data (such as ID, passwords, gender, birthday, address, e-mail, credit card information, etc.) for authentication and other application-specific purposes when users sign on their systems, or when users open/close transactions in the systems. As more and more Internet services are offered, a user may have to maintain many copies of his personal data at different places, and sometimes need to manually enter their data over and over again. To relieve users from such troublesome tasks, the so-called *Universal Profile Systems (UPSs)* have been developed to allow a user to access different information services with only a single action of authentication and authorization. Such service also reduces service providers' cost to maintain their own databases for personal information.

Generally speaking, such service can also be implemented as a component in a person's computer (e.g., integrating with the person's browser). For example, current Web browsers can remember IDs and passwords of users. When a person wants to login a Web service, he needs only to input a few alphabets, which then invoke a local procedure to interact with the Web service to auto-fill the user's identity and password for the Web service. One obvious disadvantage of this is that the procedure cannot be accessed from a remote site from which the user wishes to access the Web service. Even if the technology allows the user to do so, the data the procedure manages still cannot be accessed remotely if its hosting computer is powered off or unreachable from the Internet.

In light of these problems, remote resources can be utilized to host the users' data and provide services while the users' home computers are not reachable. Currently, several service providers have provided their UPS services (e.g., the Microsoft's *Passport* [14], the Liberty Single Signon of Sun and IBM [20], and AOL's *Magic Carpet* [19]). Take Microsoft's Passport service for example. A person can use a single Passport account to traverse all Passport participating services seamlessly. However, some concerns may arise when a UPS service is maintained and provided by a single authority or organization:

- Other service providers may not wish to share their customers' data with the UPS service provider (especially when the latter may be or may become their competitor). Furthermore, while the operation of such service can bring up commercial profits and competition advantages, several similar services may emerge. Consequently, we could end up again in having to maintain personal data at different UPS services.

- The centralized UPS service may become a threat for privacy and a clear security target. This of course does not mean that other kinds of services do not have the same concerns. However, because a UPS can touch more complete personal data of its users than other kinds of services do, these concerns pose a more serious problem than to the others. For example, a UPS service is able to track a user's behaviors secretly while the user requests other services. Moreover, the centralized data source may provide an easy way for an unauthorized third party to gather large amount of information [17].

- People may be forced to accept an extra cost, e.g., to receive a lot of advertisements and to use some companioned software packages, added by the UPS service provider.

In light of these concerns, we propose *Personal Data Backbone (PDB)* to provide UPS service using peer-to-peer (P2P) technology. The advances of personal computers and broadband networks have made it possible for people to easily collaborate with one another to establish services. For example, the legendary P2P company Napster [31] that once allowed millions of users to share MP3 files has opened the epoch of P2P applications. Although the kernel of Napster still relies on a centralized server to index and manage resource information, later development in P2P technologies [6, 9, 27, 23, 12] has completely removed this restriction. Therefore, by using P2P technologies in UPS services we can decentralize the management of personal profiles to achieve better availability, accountability, reliability, and trust.

Unlike other general purpose peer-to-peer storage systems (e.g., the Chord File Systems [27] and OceanStore [23, 12]), PDB is designed particularly to tolerate a certain degree of failures and to offer accountability of personal data access. *Accountability* enables users to know who have accessed their personal data so as to discover potential security threats. To do so, we decompose and encode a user's data into r chunks such that at least m out of the r chunks must be retrieved to reconstruct the original data. The placement of the r chunks ensures that they will be physically at r different peers (provided of course that there are at least r peers in the network). Therefore, a person's profile data can still be accessed even if $r - m$ peers to which the data are distributed are unavailable. Access to a data chunk is logged at each peer for accountability. So by letting a request to retrieve at least m chunks of the encoded partial information of a person's profile, at least m sites will have a log recording the access. So even if $m-1$ out of the m peers from which the request has been recorded have crashed, we can still correctly identify the requester.

The next section provides a quick overview of our system. Sections 3-5 then describe this system layer by layer. Finally, conclusions and future work are offered in Section 6.

2 System Overview

The structure of PDB, shown in Figure 1, is composed of four layers: *UPS Service Layer*, *Personal Data Object Layer*, *Chunk Storage Layer*, and *Base Layer*.

The Base Layer provides services for forming and accessing to the physical resources. It maintains a given topology for the participating nodes in the network, provides distributed name lookup services, and manages the shared physical resources in each host.

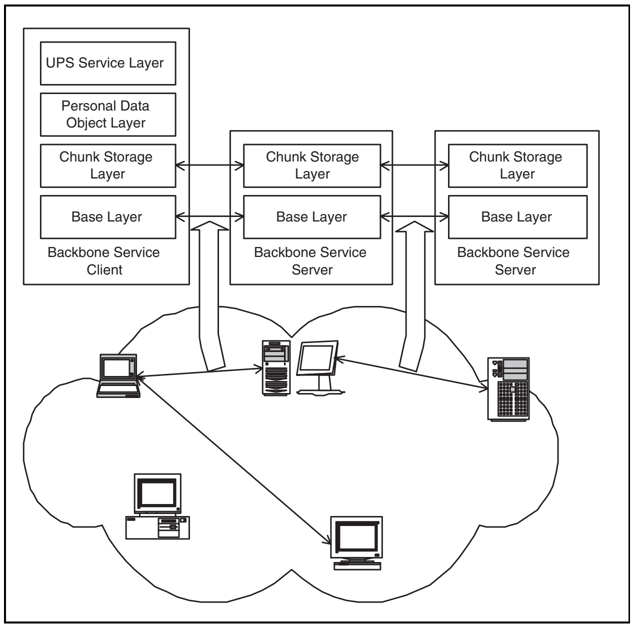

Figure 1. Layers of PDB Service.

The Personal Data Object Layer and Chunk Storage Layer maintains the physical personal data objects. We use the term 'chunk' rather than 'block' to refer to the basic data unit because it is slightly different from that used in traditional file systems. File systems usually divide a large file into several small blocks to reduce seek and processing cost. In contrast, a chunk here represents only a partial and encoded information of a data object, and different chunks of an object may contain overlapping information to cope with fault tolerance. The bottom three layers can also be replaced by other peer-to-peer storage systems. However, as shall be clear shortly, some modifications are needed to ensure fault tolerance and accountability.

The division of labor between the Personal Data Object Layer and the Chunk Storage Layer is as follows: The Personal Data Object Layer provides an interface for UPS Service Layer to gain access to individuals' personal data objects for backbone operation. It decomposes and reassembles personal data objects to satisfy the requests from the UPS Service Layer. On the other hand, the Chunk Storage Layer provides the main functions of chunks management. It contains the following components: *Chunk Access*, *Chunk Allocation*, and *Chunk Discovery*. Chunk Access provides an interface to access chunks. The Chunk Allocation mechanism allocates other nodes (by communicating with the corresponding mechanism in the destination hosts) when chunks need to be replicated and migrated. Fi-

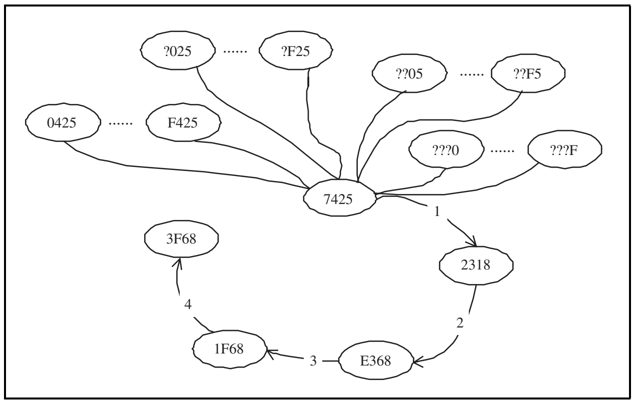

Figure 2. Routing on a Plaxton mesh.

nally, the Chunk Discovery mechanism searches for target chunks.

The UPS Service Layer provides the following four main functions for UPS services: The *opt-in* procedure is invoked when a person wants to use an ISA (Information Service and Application) through PDB *at the first time*. Thereafter, he needs to *login* the ISA for authentication every time he wishes to use its service. The *administration service* allows a person to set and modify his personal data. The *data service* allows ISAs to request data from the backbone. Details of this will be provided in Section 5.

3 Base Layer

Like many other P2P systems, naming and routing are combined together to provide route discovery and name resolution at the same time. Objects and nodes are given unique identifiers separately. Each object is then placed on a certain node by associating their identifiers. Thus, resolving an object can be done by routing to its corresponding node. Below, we introduce the basic idea of naming and routing used in our P2P system; more details can be found in [13].

PDB modified the scheme proposed by Plaxton et al. [22], which is also used in Tapestry [32] and Pastry [24]. Each node is labeled with a unique identifier that is generated by hashing its MAC address with a random time when it joins the backbone. Together, they form an overlay network which we refer to as *Plaxton mesh*. An application can thus reach a node of a specific identifier by routing on the Plaxton mesh.

Routing on the Plaxton mesh is done in a suffix-routing fashion as shown in Figure 2. The specified identifier is matched digit by digit along the routing path. For example, to send a message to some node x, say 3F68, from node y, say 7425, y first sends the message to an arbitrary node with identifier that ends with **8**, say 231**8**. Node 231**8** then matches one more digit by forwarding the message to any node that ends with **68**, for example, node E3**68**. Node E3**68** then matches one more digit by forwarding the message to any node that ends with **F68**, in this example 1**F68**. Node 1**F68** then locates node **3F68**, and we are done.

From the above example we see that locating a data object from any node in the mesh can be done in $O(\log N)$ steps, where N is the total number of nodes in the network. We can also see that a node $x = d_{c-1} \dots d_1 d_0$ (in digit representation, and assuming a hexadecimal system) needs to know nodes with identifier $*0$ (i.e., any node with suffix '0'), $*1, \dots, *\mathrm{f}$, $*0d_0$ (any node with suffix '$0d_0$'), $*1d_0, \dots, *\mathrm{f}d_0$, $*0d_1d_0, *1d_1d_0, \dots, *\mathrm{f}d_1d_0$, and so on. So each node needs to maintain a routing table of size $O(\log N)$.

To cope with node failures, the following modification is made to the original Plaxton mesh: We partition the whole backbone into r clusters as follows: The residue of the node's identifier x divided by r (that is, $x \mod r$) is used to decide which cluster a node belongs to. In each cluster, a node is routed based on the new identifier $x \div r$. So the nodes in a cluster form a Plaxton mesh. Recall that all nodes in a Plaxton mesh can be reached starting from any given node in the mesh. So to reduce routing table overhead, a node x maintains a routing table that is needed to resolve the routing information within the Plaxton mesh x belongs, plus $r - 1$ additional pointers, each of which pointing to an arbitrary node in a different cluster (see Figure 3). Routing to a specific node should first locate the proper cluster and then proceeds as described above. Therefore, starting from any node in the network (which is composed of r pairwise disjoint Plaxton meshes), it is possible to locate any other node in the network, regardless of whether they are in the same cluster or not. Moreover, locating objects in the network can be done in parallel, and so searching r objects in the network still requires only $O(N)$ time.

Node failures are remedied in two ways: backup links and random node search (these two schemes apply to both inter and intra cluster topology maintenance). Recall that at each hop in Plaxton routing, we have multiple choices. For example, in Figure 2, to find a node ending with 05, we have 0005, ..., 1005, ..., FF05. We reserve k linkages out of these choices for each hop. Therefore, we can swap to other links once we have a link failure. In case all backup links have failed, a node initiates a random node search: it selects a random neighbor and tries to route to a node that has such suffix.

4 Personal Data Objects Management

In this section, we first discuss how a personal data object is divided into chunks. We then propose a chunk place-

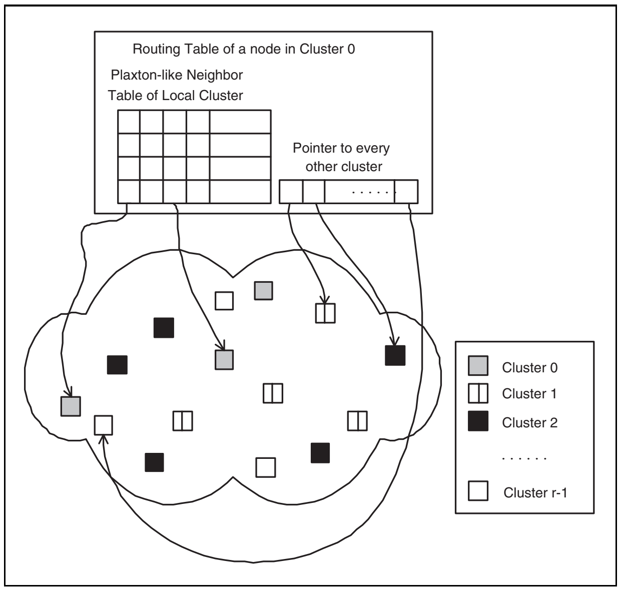
Routing Table of a node in Cluster 0
Plaxton-like Neighbor Table of Local Cluster
Pointer to every other cluster
Cluster 0
Cluster 1
Cluster 2
Cluster r-1

Figure 3. Neighbor table of clustered Plaxton meshes.

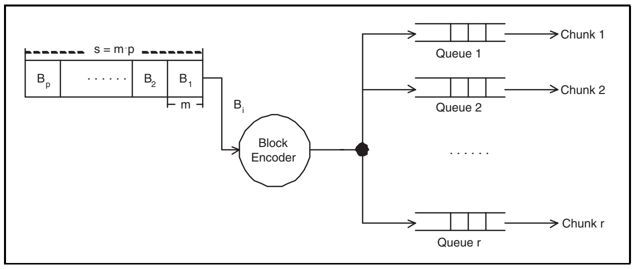

Figure 4. Decomposition of objects to chunks.

ment scheme that facilitates availability and accountability under a certain degree of failures, and then the composition of the chunks.

4.1 Object Decomposition

As shown in Figure 4, to decompose a data object (an encoded personal profile provided by the UPS Service Layer), we first divide it into blocks of size m bytes. Each block is encoded through a *Block Encoder*. The Block Encoder here uses the Reed Solomon scheme. In the areas of communication and data storage, Reed Solomon encoding scheme is often used to cope with errors and to achieve availability and data durability. It uses less bandwidth and storage than simple replication schemes in the application of distributed file system [30]. It also has the lowest theoretical possible value for correcting t errors with only $2t$ parity checks [28]. Each encoded block with k bytes parity checks of Reed Solomon codes can correct at most $2f + e \leq k$ errors, where f is the number of bytes being modified (*manipulation* errors), and e is the number of bytes being erased or lost (*erasure* errors).

Suppose the data object is of size $s = m \times p$, and suppose that k bytes parity checks are used in the encoding scheme. Each m-byte block is encoded to be an $(m + k)$-byte block. So the original data object of size $s = m \times p$ is encoded into p blocks of size $(m + k)$ bytes. Let $r = m + k$, and assume that there are r queues of data streams, each of which is to produce a data chunk. Each of the r-byte encoded blocks then is split and added to each queue, one byte per queue. So, in total there are r chunks of size p bytes to be produced by the queues.

According to the Reed Solomon encoding scheme, any m out of the r non-corrupted chunks can be used to reconstruct the original data object, but no less. In the presence of errors, more than m chunks may be needed to reconstruct object.

4.2 Chunk Placement

Every personal data object x has a unique identifier, say $\mathrm{ID}(x)$. Assume that it is of size c bits. To insert the r chunks of x into PDB, each chunk must be assigned a unique identifier, too. The i th chunk of x is given the identifier composed of $c + \log r$ bits, with the c most significant bits obtained from $\mathrm{ID}(x)$, and the $\log r$ least significant bits obtained from the binary representation of i. For notational convenience, we will use $\mathrm{ID}(x, i)$ to denote the identifier of the i th chunk.

As described in Section 3, the whole backbone is partitioned into r pairwise disjoint clusters. The nodes in a cluster forms a Plaxton mesh. A node S with identifier $\mathrm{ID}(S)$ belongs to the j th cluster if $\mathrm{ID}(S) = j \pmod r$. Assume that the identifier of a node has c' bits. Within each cluster, only the left most $c' - \log r$ bits are effectively used in determining the routing information of the Plaxton mesh of the cluster.

To place chunks to nodes, like the original Plaxton hashing scheme, a chunk $\mathrm{ID}(x, i)$ is placed in the node whose identifier matches the right most c' bits of $\mathrm{ID}(x, i)$. If the node does not exist in the Plaxton mesh, then a unique surrogate within the same mesh will be summoned to assume the responsibility of the missing node; that is, to host the chunk $\mathrm{ID}(x, i)$. As such, the placement scheme guarantees that each cluster contains at most one chunk of a data object. For surrogate choosing, the surrogate of S is the node in the mesh whose ID is the closest to S. In case there are two such candidates (when $|\mathrm{ID}(S) - \mathrm{ID}(R_1)| = |\mathrm{ID}(S) - \mathrm{ID}(R_2)|$), then the one with the smaller ID is chosen.

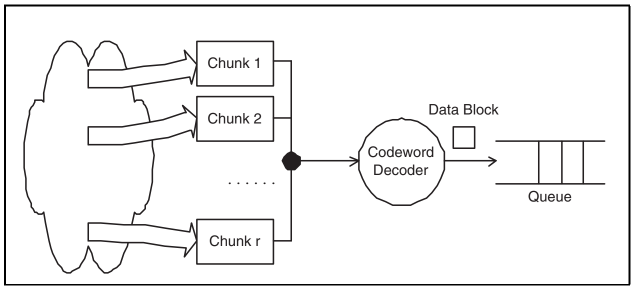

Figure 5. Reconstruction of Data Object.

4.3 Object Reconstruction

As shown in Figure 5, a personal data object, which has been decomposed into r chunks, can be reconstructed as follows: The requester attempts to collect the r chunks from the network. Then, each chunk is dealt with byte by byte. For example, the first bytes of each of the r chunks are concatenated together to form a codeword to generate the first block of the original data object. The second bytes of the r chunks are used to generate the second block, and so on. If a chunk is unavailable, its corresponding index in a codeword is marked.

By using the information of a codeword and what bytes of a codeword are unavailable, the decoder tries to reconstruct the original data block. Recall that k parity checks are used in the Reed Solomon encoding scheme. Assume that e bytes in a codeword are unavailable, and f bytes are corrupted. Then these errors can be discovered and corrected if $2f + e \leq k$, and so a correct m-byte block (cf. Figure 4) can be reconstructed and placed to a queue, which is used to generate the original data object.

If $2f + e > k$, there are two possible cases:

- The decoder can discover the error but cannot correct the error. In this case, the decoder will report the error.
- An error block may not even be discovered by the decoder, and so an error data object could be generated. The error is intended to be discovered in the UPS Service Layer where a signature is embedded in the object for verification.

There are two schemes to embed signatures into objects: (1) whole-object signature, and (2) signature per chunk. The choice of the two trades off between performance and fault tolerance. It is clear that embedding a signature into each chunk costs extra space and time. However, recall from above discussion that k parity checks can tolerate up to f manipulation errors and e erasure errors, provided that $2f + e \leq k$. If manipulation errors can be discovered, they can be treated as erasure errors by discarding the error chunks. So fault tolerance increases as now the system can tolerate up to $f + e \leq k$ errors.

To summarize, the object decomposition method (where a data object is decomposed into r chunks and the size of each processing block is m) and the chunk placement scheme can tolerate the following types of errors:

- A personal data object is guaranteed to be available while twice the number of manipulation errors plus the number of erasure errors about the object does not exceed $r - m$.
- When a chunk is accessed, the node hosting the chunk will record the request (which is signed by the requester so that it is non-repudiable). Because a requester must obtain at least m chunks of an object to access its content, a user can obtain information about who has accessed his data object by composing the logs of the hosting nodes. The access information will not be lost if the number of crashed nodes does not exceed $m - 1$. An interface is provided to the UPS Service Layer to retrieve the access logs, as well as to integrate them from different sites.

4.4 Related Work in Global Storage Services

From the infrastructure point of view, PDB can also be viewed as a global storage service, which enables people and applications to access required data anywhere and anytime efficiently. This concept is originated from distributed storage systems. For example, NFS [25] and AFS [16] provide transparent remote access to shared files across networks. These systems are developed to provide services in local area networks. Some modifications are needed to extend them to a global scale. For example, WebNFS [3] aims to use NFS in wide-area networks and Internet applications. Coda [26, 11] extends NFS to disconnected computing so that portable and mobile computers can still be used when disconnected from the network.

To allow data to be efficiently accessed in the Internet scale, a large number of servers may be needed to store replicated data. Consequently, loads can be distributed among these server and at the same time data can be accessed from a nearby server to reduce transmission delay. It may be too expensive or uneconomical for a single company to deploy these servers on its own. Therefore, content delivery networks (CDNs), which can also be viewed as a kind of global storage service, have emerged to help data distribution [10]. For example, Akamai deploys more than ten thousands servers over more than sixty-two counties [1]. Content providers can then use Akamai's CDN to distribute their content.

Instead of deploying thousands of servers, several researches have considered to utilize existing numerous personal computers with peer-to-peer technologies. People who are willing to share their storage space can cooperate with each other to provide global storage services. For example, MojoNation (MN) [15] and IBP (Internet Backplane Protocol) [21] offer CDNs formed by connecting individuals' personal computers. Similarly, OceanStore [23] is designed to maintain global data storage with millions of cooperated computers.

An important issue in global storage services is availability. A simple and perhaps the most adopted approach is to replicate each data object. Our approach in this article, however, is to decompose an object into r chunks such that at least m out of the r chunks must be obtained in order to reconstruct the object. The r chunks are placed at different sites so that a single server error can only affect one of the chunk. This approach increases reliability, facilitates accountability, and achieves censorship resistance [29, 8], which makes it difficult for a third party to make changes or force the deletion of objects in the system.

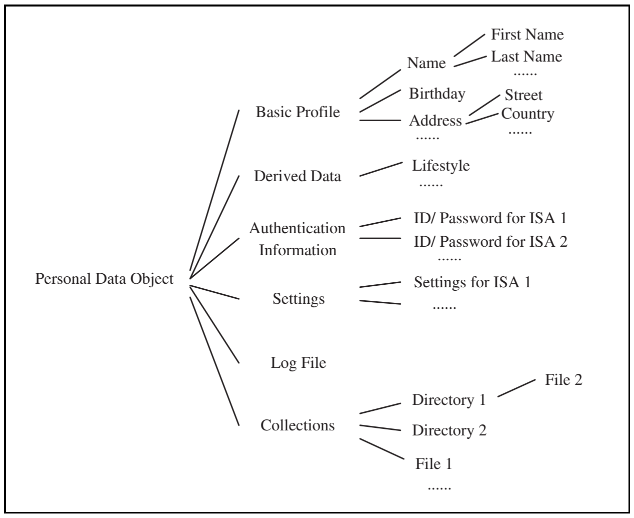

Figure 6. Logical View of Personal Data Object

5 UPS Service Layer

This section presents the UPS Service Layer used to implement the service of PDB. We begin by presenting in Section 5.1 the main functions of PDB service, and illustrate how the layer can support these functions. The detailed design and the main components are presented in Section 5.2.

5.1 Main Functions

This section details the main functions of PDB as introduced in Section 2. Because we are dealing with personal profiles, we first present our logical view of a personal data object in Figure 6. Generally speaking, a personal data object is composed of several sub-objects. For example, *basic profiles* are information that can be used to describe an individual; *derived data* are dynamically generated from other "raw" data (e.g., "the person's total amounts of online purchase last month" can be calculated from the person's behavior logs); *authentication information* includes users' id and passwords, and other information that can be used to identify who the user is; *preferences and settings* are data set by people to represent how they wish associated application services to be personalized (or customized); *behavior logs* are logs that record a person's interaction with a service application; *collections* refer to other data collected by a person, such as images, videos, documents, mailbox and other kind of files. A sub-object may further contain other attributes as its sub-objects. For example, the data object 'Name' in Figure 6 contains 'First Name', 'Last Name', etc., as its sub-objects.

Before a user can use the service of PDB, he must install a PDB service client in his local host. In order for the client to know whom the user is, the user needs to 'sign in' the backbone. We call this procedure 'sign-in' to distinguish it from the login function (to ISAs) of PDB. In the sign-in procedure, the client requests the user to input his account name, some permanent key/attributes, and password for authentication. The account name and the permanent key/attributes are used to produce a globally unique identifier for the user's personal data object. This identifier is then passed to the Personal Data Object Layer to locate and retrieve the (encrypted) personal data object. After the object is obtained, the client uses the password to decrypt the object and to authenticate the user. Note that after the user has signed in PDB, the system has retrieved his profiles and so he can thereafter access the profiles. Because our view of the profiles is a hierarchy of data objects, they can be stored and retrieved separately to increase system performance. As we shall see in next section, this logical view also allows us to enforce different security levels to different types of data objects.

The opt-in function can be depicted in Figure 7. Take Web services for example. After receiving a user's opt-in request, an ISA asks the user to request his local PDB service client to complete the opt-in process. This step can be done transparently by redirecting the ISA's request to the client. When the client receives the opt-in request, it checks if the user has signed in the backbone. If not, the client requests the user to sign in the backbone as described above. Then, the client retrieves an opt-in proposal from the ISA. The proposal is based on the *Platform for Privacy Preferences* ($P3P$) [7], which is designed to let a Web site disclose its privacy preference about what personal data is required by this site and for what purpose. An example of

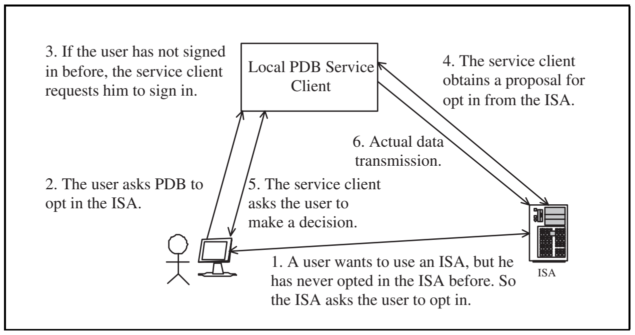
3. If the user has not signed in before, the service client requests him to sign in.
Local PDB Service Client
4. The service client obtains a proposal for opt in from the ISA.
6. Actual data transmission.
2. The user asks PDB to opt in the ISA.
5. The service client asks the user to make a decision.
1. A user wants to use an ISA, but he has never opted in the ISA before. So the ISA asks the user to opt in.
ISA

Figure 7. An opt-in scenario.

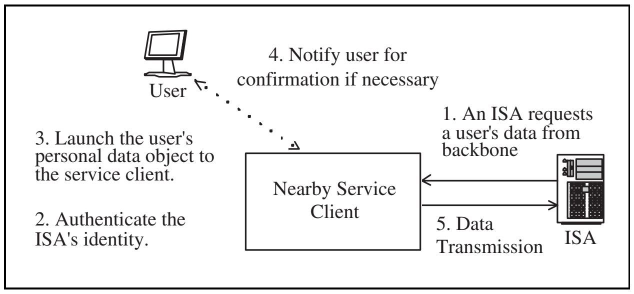
User
4. Notify user for confirmation if necessary
1. An ISA requests a user's data from backbone
3. Launch the user's personal data object to the service client.
Nearby Service Client
2. Authenticate the ISA's identity.
5. Data Transmission
ISA

Figure 8. A scenario for an ISA to request a user's personal data.

proposals can be found in [5]. The client can generate an opt-in page from this proposal, and sends the page to the user to let him confirm the opt-in process. By reviewing the opt-in page, the user can decide whether or not to opt in the ISA. If he decides to opt in, the personal data required to complete the opt-in process will be sent to the ISA by the client, and thereafter he can start using the services provided by the ISA. Note that, if needed, the client may complete the opt-in process automatically if the proposal sent by the ISA matches the preferences set by the user in his personal data object, and the user has authorized the client to skip over the confirmation step in this situation. Furthermore, licenses of using the user's data can be issued to the ISA [4, 5]. Therefore, the licenses can be checked while the data are used or been audited to ensure that the data are used in the same way that the ISA proposes in its proposal.

The login process of PDB is similar to the opt-in process, but simpler. When a user wishes to login an ISA, the PDB service client need only tell the ISA about the identity of the user so that the ISA can authenticate the user with the identity.

Moreover, when a user needs to set or modify his personal data and preferences, he can invoke the administration function of PDB. After verifying the identity of the person, the client modifies the data stored in the person's data object to serve the user's update requests. Note that, when data stored in a personal data object are modified, consistency of the object needs to be taken care of. We will return to this issue later in Section 5.2.1.

In the above functions, the PDB service client uses personal data stored in a user's personal data object to fulfill the user's request. ISAs, on the other hand, may request a person's personal data actively from PDB even if the person is not online. This function is clearly useful when some personal data may be updated frequently and the latest information is required. The scenario of the function can be illustrated in Figure 8. Similar to the opt-in process, the ISA can send requests to a nearby PDB service client, which will then locate the person's personal data object for the requests. After verifying the identity of the ISA, the client may decide whether or not to grant the request. For example, the person may stipulate that only the ISAs to which he has opted in can obtain his latest information. Moreover, the person may ask the client to notify him with the requests or even to confirm him with the final decision.

5.2 Main Components of Service Layer

As shown in Figure 9, the Service Layer contains the following components: When the *Request Processor* receives a request, it calls the *Proposal Processor* to deal with the proposal the request may attach. Depending on the request, a proposal may specify the privacy policy of the requester, or the data to be requested and the purpose of the request. The Proposal Processor compares the proposal with the person's preferences stored in the person's *Personal Data Object*, and decides whether or not to accept the proposal. If the person shall be informed to make the final decision, a notification form is generated from the proposal and is sent to the person through the *Notifier*. When the Notifier receives the person's response, it reports to the Request Processor the decision. If the proposal is accepted, the *Data and Preferences Manager* obtains the requested data from the personal data object, and transmits it to the requester.

Security is always one of the main concerns in personal data management. The components in the gray area take the responsibility of security management in the UPS Service Layer. There are several types of security threats, including data interception, data manipulation, revocation resistance (e.g., a malicious node skips update requests to its local data chunks, or stores a stale copy of a chunk), and masquerade.

To prevent data interception and manipulation, personal data objects are encrypted so that they cannot be accessed without the key at any time or any place. The data are also grouped and encrypted using different keys for different groups (see Figure 6). This allows different security level to be enforced based on the type and the confidential level

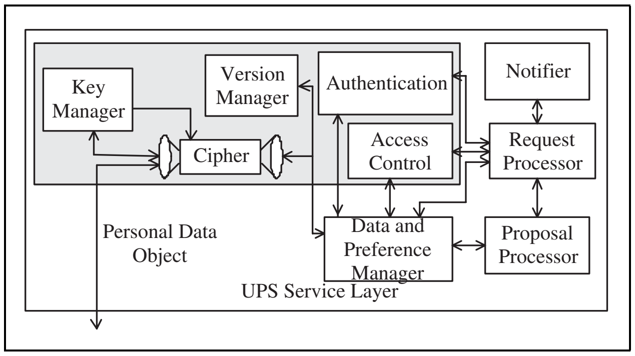

Figure 9. Components of UPS Service Layer.

of the data, as well as avoiding the key to be figured out by table lookup (because some data such as birthday have very simple format and pattern). Cryptographic solutions in [2] can be used to manage this hierarchical access control.

Nevertheless, when a user signs in at a client, personal data may then be decrypted with the keys/passwords provided by the user for operations. Although we can require the key and the content to be purged out when the user signs out, we cannot guarantee that a malicious host will follow the procedure. For a successful operation of PDB, some trust must be assumed. To a person, we say that a computer is trusted if he is convinced that the computer is not hacked and all backbone components function correctly in the computer.

Based on the above discussion, we make the following assumption in the system:

> **Trust Assumption:** *When a user is using the service of PDB, to the user there is at least one trusted computer in the network (e.g., the computer he signs in).*

Likewise, the user can always find a trusted PDB service client to request for service. He need not worry about his personal data being accessed by other unauthorized person while the raw data are decrypted because operations of a PDB service client only occur in his trusted computer.

Finally, to prevent masquerade, certificates [18] can be exchanged between an ISA and a PDB service client for verification. In addition, an SSL communication link can be built to prevent eavesdropping. Our error correction coding scheme also helps to prevent some degree of data interception, manipulation, and revocation resistance.

Table 1 summarizes the main security threats to PDB and our strategies.

Threats	Strategies Against the Threats
Data interception	Data encryption and secure channel in the UPS Service Layer Trust assumption in the UPS Service Layer
Data manipulation	Digital signature the in the UPS Service Layer Error correction coding scheme in the Chunk Storage Layer and Personal Data Object Layer
Revocation resistance	Digital signature in the UPS Service Layer Error correction coding scheme in the Chunk Storage Layer and Personal Data Object Layer Version control in the UPS Service Layer
Masquerade	Digital signature in the UPS Service Layer

Table 1. Security threats and strategies against the threats.

5.2.1 Data Consistency and Version Control

The problem of consistency arises when data can be modified and read concurrently. For personal profiles, although we allow concurrent access, we believe that it is rarely needed to allow concurrent updates to the same profile, as typically only the owner of a profile is allowed to modify the profile. This observation greatly simplifies our consistency maintenance policy. In PDB we allow only sequential updates to the same personal data object; that is, a new update cannot be issued unless previous updates have been completed. This can be easily done by using acknowledgment.

Recall that in the Chunk Storage Layer a data object is decomposed into r chunks stored at different sites. So update to a data object may still involve concurrent accesses to the chunks of the object. To enable a data object to be still available while it is being updated (in the case that strong consistency is not necessary), we let each chunk bear a version number. In addition, two schemes can be adopted: either set r to be greater than or equal to $2m - 1$ (where m is the number of chunks that are necessary to reconstruct the object), or let each site responsible for a chunk maintain the latest two versions of the chunk.

In the first scheme, when a PDB service client obtains the r chunks of an object, the r chunks must either all have the same version number, or some of them have the most recent version, while the others have the same previous version (because updates to a data object are sequentially synchronized). Since $r \geq 2m-1$ and since m chunks are sufficient to reconstruct the object, a complete version of the object can always be reconstructed (provided no other manipulation and erasure errors occur). If the majority of the chunks have a version number less than the others, then the client knows that the version it reconstructed is stale. Depending on the applications, some obsolete data object (e.g., derived data and behavior logs) may still be useful and so re-request of the object is not needed. Otherwise, the application can re-issue a request for the object.

In the second scheme, when a PDB service client requests a data object, each site responsible for a chunk sends the client the latest two versions it has. Again, because updates to a data object are sequentially synchronized, the chunks collected by the client can have at most three different version numbers, and all the r sites responsible for the r chunks of the object must provide the client with r chunks of a common version number. Therefore, the client will be able to reconstruct a complete version of the object, and can also know if the version is obsolete. Clearly, the second scheme requires more storage and band width, but is more flexible in choosing r and m.

6 Conclusion and Future Work

We proposed Personal Data Backbone (PDB) to provide UPS services over peer-to-peer networks. The main objective is to bring the control of personal data back to their owners. By using peer-to-peer technology, people can collaborate with one another to establish the services without resorting to a centralized mechanism or corporation, thereby removing concerns such as privacy, security, and monopoly. The peer-to-peer technology also achieves better trust, availability, accountability, and reliability, as compared to the centralized ones.

The current prototype of PDB we have implemented provides only a simple request and reply architecture in the exchange of personal data between a PDB service agent and an ISA. More complex features can be added in the future. For example, a negotiation mechanism can be designed for an ISA and a user (through a PDB Service agent) to reach an agreement both parties can accept. A pricing mechanism can also be investigated for ISAs to pay for the data. For example, people may charge commercial e-mail senders for using their e-mail addresses to send advertisements. Finally, like 'push' technology, update notification can be included in the framework so that when a person updates his personal data, ISAs that have requested the data before can be informed to ensure that they have up-to-date information of the data.

Acknowledgments. We thank the anonymous referees for their useful comments.

References

[1] Akamai Inc. A distributed infrastructure for e-business—real benefits, measurable returns. In *Akamai White Paper. Retrieved from* `http://www.akamai.com`, 2000.

[2] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, 1983.

[3] B. Callaghan. RFC 2055: WebNFS server specification, October 1996.

[4] Shi-Cho Cha and Yuh-Jzer Joung. Online Personal Data Licensing. In *Proceedings of the 3rd International Conference of Law and Technology (LAWTECH2002)*, pages 28–33, 2002.

[5] Shi-Cho Cha and Yuh-Jzer Joung. From P3P to OPDL. In *Proceedings of the 3rd Workshop on Privacy Enhancing Technologies (PET2003)*, Dresden, Germany, March 26-28, 2003.

[6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity*, 2000.

[7] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. Platform for Privacy Preference (P3P). In *W3C Recommendations*, 2002. Retrieved from `http://www.w3c.org/TR/P3P/`.

[8] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 67–95, Berkeley, CA, USA, July 2000. Springer-Verlag, Berlin Germany.

[9] KaZaA file sharing network. KaZaA. http://www.kazaa.com, 2002.

[10] K. L. Johnson, J. F. Carr, M. S. Day, and M. Frans Kaashoek. The measured performance of content distribution networks. *Computer Communications*, 24(2):202–206, 2001.

[11] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[12] John Kubiatowicz, David Bindel, Patrick Eaton, Yan Chen, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Westley Weimer, Chris Wells, Hakim Weatherspoon, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 35(11):190–201, November 2000.

[13] Yu En Lu and Yuh-Jzer Joung. Mystry: Distributed name resolution in dynamic networks. Submitted for publication, 2003.

[14] Microsoft. Microsoft Passport. Technical White Paper, March 2001.

[15] Mojo Nation. Retrieved from `http://www.mojonation.net/`, 2003.

[16] J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal, and F. Donelson Smith. Andrew : A distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[17] D. Mulligan and A. Schwartz. Your place or mine? privacy concerns and solutions for server and client-side storage of personal information. In *Proceedings of ACM Computers, Freedom, and Privacy*, pages 81–83, 2000.

[18] M. Myers, C. Adams, D. Solo, , and D. Kemp. RFC 2511: Internet X.509 certificate request message format, 1999.

[19] C. Newell. AOL quietly launches magic carpet. In *eWeek.com*, 2002. Retrieved from `http://www.eweek.com/ article2/0,3959,113131,00.asp`.

[20] Birgit Pfitzmann. Privacy in enterprise identity federation—policies for liberty single signon. In *Proceedings of the 3rd Workshop on Privacy Enhancing Technologies (PET2003)*, Dresden, Germany, March 26-28, 2003.

[21] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swany, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, pages 50–58, 2001.

[22] Plaxton, Rajaraman, and Richa. Accessing nearby copies of replicated objects in a distributed environment. *MST: Mathematical Systems Theory*, 32, 1999.

[23] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, pages 40–49, 2001.

[24] Antony Rowstron and Peter Druschel. Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[25] Russel Sandberg. The design and implementation of the sun network file system. In *Proceedings of the USENIX Summer Conference*, pages 119–130, Berkeley, CA, USA, June 1985. USENIX Association.

[26] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file systems for a distributed workstation environment. *IEEE Transactions on Computers*, C-39(4):447–459, April 1990.

[27] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[28] Peter Sweeney. *Error Control Coding—From Theory to Practice*. John Wiley and Sons, Ltd., 2002.

[29] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant Web publishing system. In USENIX, editor, *Proceedings of the Ninth USENIX Security Symposium, August 14–17, 2000, Denver, Colorado*, Berkeley, CA, USA, 2000. USENIX.

[30] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. *Lecture Notes in Computer Science*, 2429:328–339, 2002.

[31] Business Week. Inside napster. Retrieved from: http://www.businessweek.com/2000/00_33/b3694001.htm, August 2000.

[32] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, UC Berkeley, Technical Report UCB/CSD-01-1141, 2001.