Wireless Pers Commun (2015) 80:1449–1473
DOI 10.1007/s11277-014-2093-4

Secure Personal Content Networking Over Untrusted Devices

Uichin Lee · Joshua Joy · YoungTae Noh

Published online: 14 October 2014

Abstract Securely sharing and managing personal content is a challenging task in multi-device environments. In this paper, we design and implement a new platform called personal content networking (PCN). Our work is inspired by content-centric networking (CCN) because we aim to enable access to personal content using its name instead of its location. The unique challenge of PCN is to support secure file operations such as replication, updates, and access control over distributed untrusted devices. The primary contribution of this work is the design and implementation of a secure content management platform that supports secure updates, replications, and fine-grained content-centric access control of files. Furthermore, we demonstrate its feasibility through a prototype implementation on the CCNx skeleton.

Keywords Content-centric networking · Personal content sharing · Content-centric access control

1 Introduction

Today, people carry various consumer electronic devices such as digital cameras, smartphones, and laptops. These Internet-enabled smart devices enable both *consumers* of published content and *producers* of user-generated content. Content creation has become very easy because anyone can post content using Web 2.0 tools, e.g. YouTube, Flickr, Twitter, etc. As a result, personal content is exploding: content is shared and stored in multiple places ranging from personal devices to cloud storage. A recent report estimated that by 2015, terabytes of data will be in a person's pocket and petabytes of data in a person's home [37]. Under these circumstances, it is very important to have a system that seamlessly enables networking of personal content such that users can manage personal content across multiple devices (including cloud storage) and selectively share content with intended groups (e.g. family members, friends, and colleagues).

U. Lee · J. Joy · Y. Noh (✉)
UCLA, Los Angeles, CA, USA
e-mail: ytnoh@cs.ucla.edu

The first step toward this goal is to introduce single persistent naming over personal content across multiple devices. Because the current generation of personal devices maintains individual namespaces in each device, content is closely tied to a device (i.e. it is location dependent). As the amount of content increases, content management becomes more difficult because users tend to lose track of what files are located where. A unified view with persistent naming will allow users to make location-independent (or content-centric) queries where there is no need to specify which device has the requested content. For example, Alice can access her favorite songs via a name: *Alice/Music/My Favorites*. Similarly, she can share the collection to Bob by simply telling him the location name.

This content centric approach is considered to be a key feature of the future Internet through content-centric networking (CCN), which replaces the conventional host-to-host conversations with name-based communications and provides secure binding between the name and data in order to thwart security attacks. The name-based routing of CCN enables content retrieval over a fully distributed network without specifying where the content is located because any nodes that have the requested data locally answer the request [21]. While CCN was originally designed for large-scale content dissemination (or potentially replacing the existing IP network), its key principles (i.e. name-based routing and secure binding) are also applicable in the realization of secure personal content networking. However, due to scalability and performance reasons, CCN transfers data and also caches data on *untrusted devices* that forward/store data properly and yet do not necessarily keep the data confidential. In addition, CCN lacks the essential component of personal content networking of secure content management, such as content updates and access control over untrusted nodes.

While secure content management is an active area of research in the field of distributed file systems, existing work has primarily focused on host-centric trusted file systems; a trusted file server handles user authentication and access control authorization, and then provides data confidentiality through securing the communication channel (e.g. SFS [27]). When managing untrusted storage, the files must be encrypted in order to assure data confidentiality, e.g. Cryptographic File System (CFS) [4] and Plutus [24]. However, such cryptographic storage systems cannot generally provide fine-grained expressive access control; for example, in Plutus, a file can only be encrypted using a single key. If a user wants to share the file with more than two groups, it is not clear which key should be used for the encryption. A simple solution is to use a common key for file encryption and to encrypt this key using each user's public key as in SiRiUS [19]. However, this approach is limited because the metadata size linearly scales with the number of users, and supporting more expressive access control is difficult because it only uses a single file encryption key. Moreover, the existing cryptographic systems do not support *secure binding* between the name and data and, as a result, the channel must be secured in order to prevent man-in-the-middle attacks.

In this paper, we propose the personal content networking (PCN) platform and it provides a secure content management mechanism over CCN, which enables secure replication and updates, as well as fine-grained content-centric access control. We extend CCN to build a framework for distributed content management with replication and updates. Then, we propose and implement a *secure content-centric access control* mechanism using the recently proposed cryptography tool called attribute-based encryption (ABE), which permits secure sharing of content within a group over untrusted devices [3]. ABE supports fine-grained expressive access policies called attribute-based access control (ABAC). An owner can define a set of attributes (e.g. college friends, CS219 team, family members, etc.) and then they issue a secret key for the assigned attributes to an individual. Each file is encrypted based on the access policy over the attributes using the owner's public key. For a given encrypted file

and access policy, any user can decrypt the file as long as they have the secret key with the attributes that satisfy the given policy.

While ABE was designed and has been used for selective *read-only* content sharing over untrusted storage [49,50], this is the first attempt to build a fully distributed personal storage system that supports ABE-based fine-grained access control with *read-write* operations over untrusted devices and *secure-binding* between the name and data. The primary contribution of this paper is twofold.

- We design the PCN platform through significantly extending CCN in order to realize a secure content management mechanism that supports secure replication and updates, as well as fine-grained content-centric access control.
- We build a PCN prototype through integrating the whole system using FUSE [16], which is a user level file system, and we demonstrate that a user can seamlessly access and manage content using PCN.

The remainder of this paper is organized as follows. We present the design goals of PCN (Sect. 2). We provide PCN's basic framework (Sect. 3) and secure content management methods (Sect. 4). Then, we present the prototype implementation (§5) and the preliminary evaluation results (Sect. 5.2). Next, we discuss some of the remaining issues (Sect. 5.3). After reviewing the related work (Sect. 6), we conclude the paper (Sect. 7).

2 Target Scenario and Design Goals

We use the following example to motivate the needs of personal content networking. Bob has a number of smart devices (see Fig. 1): Internet TV, desktop computer, iPhone, WiFi-enabled digital camera, Internet fridge, and network attached storage (NAS). He has other devices at school (e.g. desktop computer, laptop) and also maintains a few cloud servers (e.g. Amazon EC2). His personal content is currently stored across these places, and Bob had a difficult

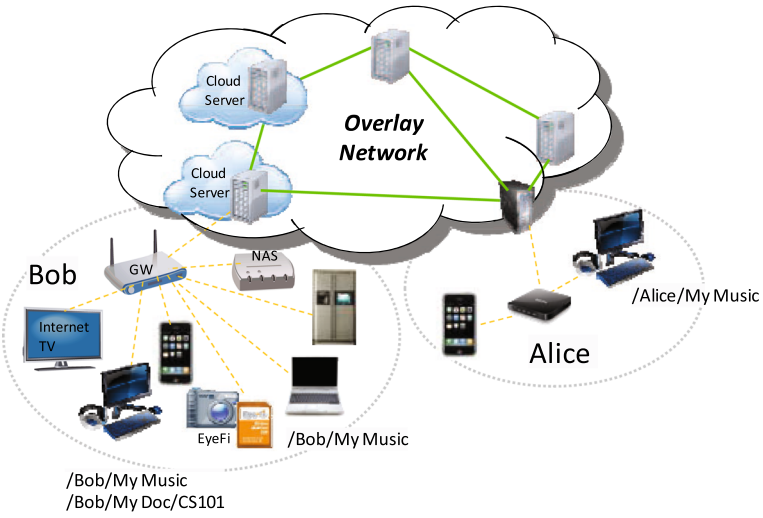

Fig. 1 Personal content networking scenario: Alice and Bob manage their personal content over multiple devices using single, hierarchical, persistent naming. Bob can share the content with Alice by simply passing the link (say, */Bob/My Doc/CS101*)

time tracking all these files. For example, his friend Alice asks him to send the lecture material of the course that they took last year. He only remembers that it is located in a document directory, but he forgets which device he put it on. He searches through his devices one by one: laptop, servers, desktop computer, and NAS, and finds that it is stored in his NAS at home. After locating the lecture material, Bob feels a bit frustrated because the size is over 1 GB (even after compression) and he cannot send it via email. He calls Alice saying he will give the files to Alice using a USB stick.

This example clearly illustrates the needs of personal content networking (PCN) such that users can manage personal content stored across multiple devices and selectively share content with intended groups. The design goals of PCN can be summarized as follows:

- *Single persistent hierarchical namespace* Single persistent naming of personal content will guide users to have a unified view of their personal content stored across multiple devices. A hierarchical namespace is essential because it has been reported that hierarchical naming significantly lessens the cognitive overhead of locating files [10,20,23,26].
- *Social networking* Users often want to share content with their friends. PCN should leverage social networking aspects through establishing and managing trust relationships among friends and through building an overlay network based on these trust relationships for content sharing.
- *Fine-grained access control* PCN must provide fine-grained expressive access control to enable secure content management over distributed untrusted servers that store/transfer data properly and yet do not necessarily keep the data confidential.
- *Disrupted operations* Because devices can go offline at any time, users should be able to replicate files and files must be automatically synchronized whenever the devices come online again as in existing distributed file systems [33,38,42].
- *Security guarantee* Because PCN manages distributed untrusted devices connected over the Internet, it must be resilient to well-known security attacks such as a denial of service attacks and false data injection attacks.

In the above scenario, PCN will allow Bob to easily locate the material (e.g. */Bob/My Doc/CS101*) and to pass this link to Alice. Bob does not need to examine each device, but simply needs to browse his namespace on any machine. Alice can download the content using the link provided. In the following, we review CCN, which is a building block of PCN (Sect. 3) and demonstrate how PCN's basic framework can be built over CCN (Sect. 4).

3 Basic PCN Framework With CCN

In this section, we review the core components of CCN: (1) naming, (2) content reachability, (3) content retrieval, and (4) content-centric security, which are the basic building blocks of PCN's underlying content retrieval. We discuss how PCN builds a web of trust and an overlay network based on social relationships, because the key functions of personal content networking is to share content among friends.

3.1 Background: CCN Review

3.1.1 Naming

CCN names a file with a user friendly, structured, effectively location-independent name. Each file is divided into multiple segments. Consider the following example name:

/parc.com/music/abc.mp3/v3/s0. Here, parc.com is a globally routable name (called a prefix), */music/abc.mp3* is a local name in *parc.com*, *v3* is a version name (represented using a timestamp), and *s0* denotes the segment number.

3.1.2 Content Reachability

In CCN, a prefix owner announces their prefix to the entire network. For example, Alice from *parc.com* announces her files as */parc.com/Alice/* from her laptop. Each node in the network broadcasts the incoming prefix to its neighboring nodes. Whenever a node receives the prefix, it establishes a backward pointer to the sender in its forwarding information base (FIB) for that prefix. Content is reachable to the prefix announcer as any content request can be routed by following the backward pointer in the FIB.

3.1.3 Content Retrieval

Content retrieval is pull-based as in HTTP (i.e. get and response). A user sends a request (via an Interest packet), and any node that has the requested content in its local storage (or cache) can respond to the request. For a given prefix, the Interest packet is forwarded along the reverse path toward each data source through following the backward pointer in the FIB. Whenever a node receives an Interest packet, the breadcrumb information (i.e. the backward pointer to the previous forwarder) is stored in the pending interest table (PIT). Then, the corresponding data packet will be delivered according to the reverse path in the PIT.

When forwarding an Interest packet, CCN uses the longest prefix matching algorithm; i.e. in the FIB, a node finds the longest prefix entry that has the largest number of leading letters matching those of the content name in the Interest packet. For example, Alice's desktop has */Alice/my music/pepper/*, and Alice's laptop has */Alice/my music/*. When Bob accesses */Alice/my music/pepper/abc.mp3*, it matches the prefix entry of */Alice/my music/pepper/* and the Interest packet will be delivered to Alice's desktop.

3.1.4 Content-Centric Security

CCN supports secure binding between the name and content (called content-based security) where protection and trust travel with the content itself [21]. To this end, CCN uses asymmetric cryptography: the content is authenticated with digital signatures, and private content is protected with encryption. Each data packet contains the owner's signature P, $Sign_P(N, C)$, which covers the name (N) and content (C). This content-based security is critical because content can be cached in untrusted intermediate nodes. For key management, CCN can use a traditional certificate-based public key infrastructure (PKI) or a Web of Trust (e.g. PGP).

3.2 Naming in PCN

The current generation of personal devices use rigid and weak naming of the form "hostname:path". The key problem is that content is tied to a host, which makes personal content management non-trivial, particularly when a user interacts with a number of devices (e.g. laptop, desktop computer, smartphone) including cloud-based storage services (e.g. Dropbox). A user must track what files are located in which devices/services and decide how to migrate/replicate/update the content.

In PCN, we define a single persistent hierarchical namespace for each person. It is known that global namespaces are politically and technically difficult to implement (e.g. X.509 [47],

PEM [36]). Thus, we use the local decentralized namespaces of SPKI/SDSI [8]. Each person has a public-private key pair to verify the identity of the sender (sign/verify) and to ensure privacy (encrypt/decrypt). The relationships among users in PCN are considered to be flat, and it is sufficient to use the public key as an identity. Nonetheless, there are cases where hierarchical naming is useful, e.g. a group of users has a set of sub-groups. In SPKI/SDSI, a user can define a local namespace as follows: a user's key K followed by a single identifier (which is distinct within the local namespace). For example, Alice has a key K_a and makes her own name as "K_a Alice". A study group with key K_g can name its sub-groups as "K_g sub1" and "K_g sub2". If a sub-group has multiple smaller groups inside, that group can name those groups similarly; for example, sub1's two internal groups (ssg1 and ssg2) can be named as "K_g sub1 ssg1" and "K_g sub1 ssg2". Note that a local name is globally unique because the name contains the public key of the user. Moreover, each user can make signed statements of these *local names*, which allows anyone to certify a key via a web of trust [8].

Given this, the data name has the form $N = P : L$, where P is the user name (or its cryptographic hash), and L is the label representing the location of data in the hierarchy, e.g. Alice's music can be denoted as */K_a Alice/music/*. PCN's naming can be used in CCN with minimal modification as CCN uses hierarchical naming (e.g. */parc.com/test.txt*). As in CCN, each device advertises the content reachability information through broadcasting the name prefix of the content that is stored in the device. For example, Bob's laptop will advertise */K_b Bob/my doc/*, and his iPad will advertise */K_b Bob/my music/Beatles/*. For the sake of brevity, hereafter we will use *abbreviated names* without a public key, e.g. */Bob/my music/*.

3.3 Trust Management in PCN

Each PCN user has a private-public key pair that is used to define their name. When a new device is purchased, this information must be securely installed in order to initialize the PCN service. Moreover, for content sharing with others, a user must establish a trust relationship through securely exchanging public keys (e.g. how does Bob make sure that the key belongs to Alice?). Nonetheless, trust relationships do not necessarily guarantee data confidentiality. For both device initialization and trust establishment problems, secure key distribution is the critical issue. Users can use USB sticks or local/wide area networks for key exchanges. The latter is less secure than the former, because it is vulnerable to man-in-the-middle attacks.[1]

A simple method of avoiding the attack is to use another secure channel. Alice can show (or read) her public key to Bob (e.g. via physical presence, SMS, email, voice communications, etc.). She can ask Bob to verify whether his key matches the received key. Given that verifying a large number is laborious and can be erroneous, Ellison et al. [11] proposed an approach where the keys are represented in color bars so that users can more easily verify the key. In Unmanaged Internet Architecture (UIA), multiple choice questions are used to reduce the user's burden [15]. For example, Alice sends her multiple choice question to Bob, and Bob sends his question to Alice. After solving each other's question, they exchange the hashed values of their answers (and both keys), hoping that the attacker cannot solve the questions and thus fail to control the conversation.

However, this approach is also vulnerable to man-in-the-middle attacks because a malicious user can perform a dictionary attack. The attacker knows both keys and the multiple choice questions. They can easily find the answer through computing a hashed value for each answer choice and comparing this value with the received answer. Like UIA, we use

[1] An attacker can eavesdrop on the channel and make independent connections with the victims and then relay messages between the victims making the victims believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker.

multiple choice questions, and yet we solve the man-in-the-middle attack using Ellison's approach [13], which is based on Pedersen's interlock protocol [34]. Given that the secret (i.e. the answer of a multiple choice question) is a, one chooses a random value u and then computes $x = g^a h^u \mod p$, where p is prime, and g and h are generators of the group mod p. Alice and Bob generate their own numbers and exchange these values, i.e. Alice generates $x_A = g^{a_A} h^{u_A} \mod p$, and Bob generates $x_B = g^{a_B} h^{u_B} \mod p$. The attacker cannot infer the value of u and must use a random value to finish the transaction, which thus effectively thwarts the dictionary attack.

3.4 CCN Overlay Construction

Trust management among friends can be used to form a social network. This social relationship is used to create an overlay network for content-centric networking (CCN). Whenever an identity introduction occurs, the corresponding personal devices also exchange IP addresses and join the overlay network. Each device maintains a peer list that contains the IP addresses and port numbers of other devices. For a given user, the list includes the user's own devices and direct friends' devices. For example, Alice's laptop has a list of all her devices and a list of Bob's devices. These devices periodically check the availability of neighboring devices in order to maintain the overlay network. This is further discussed on a device hidden behind a NAT in (Sect. 5.3).

4 Secure Content Management

In this section, we first illustrate file replication and synchronization, and we justify the need for prefix protection in replication. Then, we present details about content-centric access control, followed by an illustration of remote content management and a discussion of key revocation.

4.1 Replication

PCN supports both file- and directory-level replication services. Replication is straightforward because a user simply needs to republish the fetched content into the local CCN client's repository. Then, the prefix of the file is announced so that the other nodes can also fetch the file. For example, Alice has her favorite song in her laptop and it is located at */Alice/my music/pepper/abc.mp3*. She simply downloads the file from her desktop computer and asks the local CCN client in her laptop to replicate the file. Now, both her laptop and desktop computers announce the prefix */Alice/my music/pepper/*. However, directory replication requires more attention because it contains a set of entries each of which associates a name with a pointer to a file or subdirectory. If directory replication is requested, the local client recursively fetches all associated files/subdirectories to the local repository. For example, if Alice replicates */Alice/my music/* in her laptop, the local client downloads all files from her remote desktop computer and then announces the prefix */Alice/my music/*.

Note that in PCN, applications can access files without replicating them. Recall that a CCN node has a two-tier data hierarchy: a local cache (in memory) and a local repository (on a disk) [21]. If a requested file is not present in the local cache, its local repository is examined. If that fails, an Interest packet will be issued and the file will be fetched from a remote node. The fetched file will be temporarily stored in its local cache from which applications can access it. Through doing this, locality can improve the accessibility. Note

also that a user should be able to navigate the namespace (e.g. the UNIX *ls* command). By treating a directory as a special file, PCN collects the directory entries from the devices using a procedure similar to that of finding the latest version of a file in CCN.

4.2 Synchronization

PCN provides "eventual consistency" in that all replicas eventually converge to the same version given sufficient messages exchanged among the participating devices (i.e. a file with the freshest timestamp) [33,38]. Eventual consistency is a widely used consistency model in disruption-prone mobile environments.

Whenever a replicated file is updated, a new version is created thereafter (timestamp). Each replicated file has an associated version vector that tracks its update history [32,38]. In order to create an alert for this event, the node that makes the update will re-announce the corresponding prefix with a modification mark, which is a special type of prefix announcement that is used for the update notification. The prefix announcement also contains detailed information of the updated file, including its name, the current version, and the version vector. For synchronization, the local client compares the version vector of the local replica with that of the updated file. If the updated file is strictly newer than the local one, its version vector will dominate; the local client fetches the updated file and replaces the local file in the repository. If two version vectors are not equal and neither one dominates, an update/update conflict occurs. If automatic merging fails, PCN notifies the user that a conflict has been detected. The user will be presented with a revision history including authors, dates, and versioned content. It is left to the user's discretion to resolve the conflicts and mark the content as merged. Note that whenever intermediate nodes hear the modification announcements, their local caches are examined in order to determine whether there are matching files, and the matched files (or data packets) in the caches are invalidated.

Synchronization of a replicated directory needs a special care. Although the modification operations are limited to adding new entries or deleting/changing existing entries, a directory replica can be modified from multiple places, which causes several well-known synchronization issues such as insert/delete ambiguity, remove/update conflicts, and name conflicts [2,32]. In PCN, we adopted the existing solutions used in the Ficus file system [32,38].

When a node re-joins the PCN network after a disruption, it first checks its neighbors to find any missing prefix announcements. As will be seen, a PCN user has a reserved namespace for devices, namely */dev*, and devices are accessible through this name, e.g. Alice's iPod is named */Alice/dev/iPod*. For prefix announcement synchronization, each device stores the received prefix announcements in a designated place, e.g. Alice's iPod has */Alice/dev/iPod/received_prefix*. This allows the node to search for the updates of the files located in its local storage. If the node finds a prefix with a modification mark, it performs file synchronization as described earlier.

Note that in PCN, nodes fetch the updated file for synchronization. If the size of a file is large and only a small part of the file is updated, fetching the entire file will waste the bandwidth. A simple solution to this problem is that a node generates/publishes a delta file (e.g. using a diff file) and includes a link to the delta file in the prefix announcement.

4.3 Prefix Protection

Thus far, we have assumed that any node can replicate the content and announce the name prefix. After replicating the content, however, malicious users can launch an attack through inundating the network with fake update announcements. PCN nodes could waste consider-

able resources managing these fake updates. Given that CCN does not manage updates, this problem is unique to PCN. In order to manage this, we impose a restriction that the prefix announcement must be signed by the prefix owner. This technique is a reasonable approach because people typically want to have full control of their namespace and the locations of files in a multi-device environment. A similar technique is used in a secure BGP where each prefix is signed in order to prevent prefix hijacking where an attacker has partial or full control of the named prefix [25].

In PCN, a prefix announcement is augmented to include a signature that certifies the prefix ownership. Furthermore, we implement the ownership delegation such that an owner certifies that a named user is allowed to announce the named prefix through issuing a prefix certificate. For example, Alice can issue a certificate to Bob that he can announce the prefix */Alice/my doc/proj/*. Intermediate nodes can verify that the certificate is valid and that the prefix announcement originates from Bob (similar to data packet validation). Bob can update his local replica of Alice's file, and the update will be automatically propagated. Note that it is possible for the attackers to perform a replay attack where a CCN speaker replays a prefix that it has previously heard. This problem can be mitigated through adding an expiration timer as in S-BGP [25].

4.4 Content Centric Access Control

Access control in personal devices is primarily host-centric. In identity-based access control (IBAC) [41], a user first logs into the system (authentication) and then accesses files based on the permissions in the access matrix (authorization). SPKI/SDSI supports Role-Based Access Control (RBAC) where permissions in the access matrix are tied to roles [8]. SPKI/SDSI is also host-centric because it assumes trusted servers and insecure channels, i.e. an individual must first set up a secure channel (using SSL) to prevent man-in-the-middle attacks, and the server verifies whether a requester's key is on the role-based ACL [5]. However, in PCN, host-centric access control is not suitable because the channel is not secured but the data itself is secured.

In PCN, we need *content-centric access control*, i.e. the access control of content is self-contained and is not tied to a host. A simple solution is to encrypt the content using the receiver's public key and to define a specific name for the encrypted data that is meaningful to the receiver. The encrypted content can be placed on untrusted servers because others cannot decrypt the content. If a file needs to be shared with multiple people, a common key is used for the file encryption, and this key is encrypted using each user's public key [19]. Then, the encrypted keys are included in the metadata of an encrypted file, and the entire content (metadata + encrypted file) is published. However, this approach has several limitations. Supporting expressive access control is difficult because it only uses a single file encryption key. If multiple keys are used, the metadata size linearly increases with the number of users/groups. More importantly, once the content is published, the owner cannot give access to other users; that is, the owner must republish the original file and include additional users.

In PCN, we solve this problem using ciphertext-policy attribute based encryption (CP-ABE), which permits secure sharing of content within a group across multiple untrusted servers [3]. ABE is the key enabler for attribute-based access control (ABAC) where access decisions are based on the attributes associated with individuals. First, each user generates an ABE public key (PK) and an ABE master key (MK). A user can define a set of attributes (e.g. college friends, CS219 team, family members, etc.) and an access policy using Boolean formula for the attributes. This allows a user to perform fine-grained, expressive access control. The user assigns a set of attributes to each user and then issues a secret key corresponding to

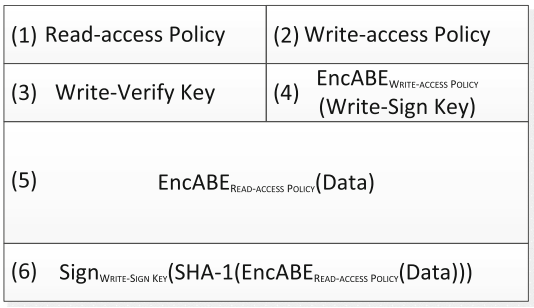

Fig. 2 PCN's file data structure for secure content centric access control

the attribute set, i.e. *Secret Key (SK) Generation(MK, S)*, where MK is the master key and S is a set of attributes assigned to the user. A file can be encrypted using the public key and access policy, i.e. *Encrypt(PK, M, A)*, where PK is the public key, M is a message, and A is an access policy. Here, any user can encrypt the file using the public key. Furthermore, any user who has a secret key with attributes that satisfy the policy can decrypt the content, i.e. *Decrypt(PK,CT, SK)*, where PK is the public key, CT is the ciphertext, and SK is the secret key [3]. In ABE, the metadata size does not linearly increase with the number of users/groups because its metadata only contains the access policy information whose size scales with the number of attributes. Moreover, the owner can still issue attribute keys for published content without republishing the content.

Assume that Alice would like to selectively share her music collection, */Alice/my music/rock*. This content is digitally signed using Alice's publisher key and is published under that namespace. For access control, *Red Hot Chili Peppers* is encrypted with the attribute *college friends*. *Incubus* is encrypted with both *college friends* and *CS219 team* attributes because Alice discussed Incubus with her teammates and wants to share the songs with them only. Bob is Alice's college friend, and Alice issues a secret key for the attribute *college friends*; Cathy is Alice's CS219 team mate, and Alice issues secret keys for the attributes of *college friends* and *CS219 team*. Bob can decrypt *Red Hot Chili Peppers*, while Cathy can decrypt both *Red Hot Chili Peppers* and *Incubus*.

In PCN, an owner of a file can set access permissions of *read* and *write* using separate access policies (as used in ABE). The resulting access modes in PCN are *read-only* and *read-write*; the write-only mode is not suitable for personal content networking. Access modes can be also used with directory files in order to limit access of a directory listing. As shown in Fig. 2, the PCN payload contains the following fields: (1) read-access policy, (2) write-access policy, (3) write-verify key (public key), (4) write-sign key encrypted using ABE with write-access policy, (5) actual data encrypted using ABE with read-access policy, and (6) write signature (optional). For write access control, a file owner issues a private-public key pair that is located in fields (3) and (4). The write-sign key is only accessible to those who have write permission because the write-sign key is encrypted using ABE with the write-access policy. Whenever a file is updated, the file is encrypted using ABE with the read-access policy. This legitimate modifier then reads the write-sign key through which it generates the signature of the updated content, which is placed in field (6). Then, this update event is notified to all nodes that replicate the content via a prefix announcement with a modification mark. The replica nodes will then fetch the updated content and verify whether it is modified by legitimate users who satisfy the write-access policy. Note that in our prototype implementation, we use symmetric encryption in order to reduce the overhead of encrypting/decrypting the content, i.e. the content is encrypted using AES, and ABE is used to encrypt the AES key.

4.5 Replica Management

A user may want to know what files are stored where and wish to replicate files to remote devices. Regular content browsing such as the UNIX command *ls* does not tell users in which device the files are located. For replica management, PCN reserves a special directory for devices, namely the */dev* directory through which a user can freely name their personal devices. For example, Alice's iPad can be named */Alice/dev/iPad*. Furthermore, each device announces its device name prefix in device-to-device communications over the CCN.

Similar to the device communications through files in UNIX, the user can write a replication command to a reserved device file, e.g. */Alice/dev/iPad/.cmd*. After updating the file, its prefix will be announced to the network, and the target device is notified of the update. Then, the device will synchronize the file by fetching the most up-to-date *.cmd* file. The device finally executes the replication command as specified in the file. Note that for security purposes, PCN restricts this function so that only the prefix owner can update the device files, and the files should be signed using the owner's private key. Multiple concurrent requests can be managed using serialized updates based on timestamps. Delayed execution is not permitted and requests expire after a threshold period of time.

4.6 Key Revocation

PCN primarily uses the following keys: a personal public–private key pair, group public–private key pairs, and ABE keys. Secure key distribution can be assured because PCN uses the secure identity exchange mechanism and relies on an SPKI/SDSI-style web of trust. Any intermediate nodes will be able to correctly acquire public keys that are then used to validate the secure binding between the name and data. While we can leverage CCN for secure key distribution, we must be able to appropriately handle key revocation scenarios for a public–private key pair and an ABE attribute secret key.

If a user's public–private key pair is compromised, the existing key pair can be revoked through the prefix announcement with a revocation mark that is similar to a suicide note in PGP [39]. Recall that in CCN, we added two additional prefix types of modification (for update notifications) and revocation (for revocation notifications). Then, the user will generate a new key pair and distribute the public key via the secure identity introduction process, which guarantees that attackers cannot impersonate potential victims. Note that the same procedure can be used to manage the case where a group's public–private key pair is compromised.

If an ABE attribute secret key is compromised or the owner wants to revoke a specific attribute, the owner must revoke both the master key and public key because CP-ABE does not provide a mechanism for revoking an individual attribute. While CP-ABE has a single attribute revocation through the addition of a timer attribute for each attribute, this approach is less practical because it complicates the overall system: (1) the owner must periodically issue keys and all files must be re-encrypted with new attribute sets, and (2) a tamper-proof clock is required to ensure the security guarantee.

Whenever the master key and public key are revoked, the owner must re-encrypt all files. However, this process is very expensive. In order to reduce the overhead, we employ the lazy revocation scheme proposed by Kallahalla et al. [24]. Unlike the compromise of a public-private key pair, that of an ABE attribute secret key is less serious, as long as the revoked user (or attacker) only has read-only access rights: the revoked user cannot remove or update files. In this scenario, it can generally be assumed that the revoked user has read and copied

all files, and it is still acceptable for the user to read unmodified or cached files. However, the lazy revocation ensures that the revoked users are not able to read *updated files*; that is, the updated files will be re-encrypted with the new ABE public key.

Note that it is also possible for users to immediately revoke all keys and re-encrypt all files. In this case, the user must undergo a series of steps: (1) re-generate a new ABE key set, (2) invalidate all cached files via prefix announcement, (3) remove the replicas from multiple devices, and (4) re-encrypt all files and re-distribute the replicas.

5 Related Work

5.1 Distributed Peer-to-Peer File Systems

Research on distributed file systems for mobile environments has been primarily directed toward extending the existing client/server-based file systems to manage node mobility and network disruption [33,38,43]. A common technique is to use optimistic file replication and eventual consistency. BlueFS [31] extends the client/server-based file system through focusing on power management to save energy in mobile devices. Compared with coda (i.e. a distributed file system), BlueFS substantially reduces the file system energy usage and provides up to three times faster access to data replicated on portable storage. EnsemBlue [35] builds upon BlueFS and provides a consistent view of all files located across multiple devices with heterogeneous device capabilities in a self-organized manner. EnsemBlue supports namespace diversity through translating between its distributed namespace and the local namespaces of consumer electronic devices. It further supports extensibility through persistent queries, which is a robust event notification mechanism that leverages the underlying cache consistency protocols of the file system. Ficus [38] uses a flexible peer-to-peer (P2P) model for optimistic replication where all replicas are equal and can propagate updates to all other replicas. It has also been reported that Ficus reliably detects all possible conflicts. Bayou [45] is a P2P weakly consistent storage system where clients are able to connect to any available server to perform reads and writes. In order to support automatic conflict detection and resolutions, it uses anti-entropy for consistent management and supports a database language for data retrieval. Ivy extend a multi-user read/write P2P file system [30]. A detailed survey of P2P file sharing has been presented in this survey paper [7].

Several systems have been designed for multi-device environments. unmanaged internet architecture (UIA) provides zero-configuration connectivity among mobile devices through personal names [15]. Unlike the existing work, UIA assumes that each device has its own persistent namespace, and a user must track all files located across multiple devices. In contrast, Eyo [44] offers a device transparency model in which users view and manage their entire data collection of all devices through periodically flooding metadata everywhere. PersonalRAID [43] supports optimistic replication at a volume level, and a mobile storage device is used to provide the abstraction of a single coherent storage name space that is available everywhere, and it ensures reliability through maintaining data redundancy on a number of storage devices. Footloose [33] is a user-centered data store that can share data and reconcile conflicts across diverse devices. Footloose supports application-specific optimistic replication with eventual consistency (e.g. address books), and yet it uses a persistent flat namespace (called ObjectID).

5.2 Wide Area P2P Storage Systems

Wide area P2P storage can be classified based on the overlay structure: (1) a structured system (e.g. PAST [12], CFS, Ivy) forms a structured overlay network using a distributed hash table (DHT), and (2) a structure-less scheme (e.g. Gnutella and eDonkey) forms a structure-less overlay network where the overlay links are arbitrarily established. Unlike unstructured P2P networks, DHTs provide better performance for searching for items over a large number of distributed nodes, and they have been widely adopted to implement wide area P2P storage. Most P2P storage systems assume wired Internet scenarios and support strong consistency, which is less suitable for personal content networking. As a recent work, Plethora focuses on semi-static peers with strong network connectivity and a partially persistent network state. In a semi-static P2P network, peers are likely to remain participants in the network over long periods of time (e.g., compute servers), and are capable of providing reasonably high availability and response-time guarantees [14].

5.3 Decentralized Access Control

The following concepts are closely related: user authentication, access control authorization, and data confidentiality. Existing access control systems can be classified based on their authentication method. When AUTH_SYS (UNIX's default) and Kerberos are used, systems mostly provide UNIX-style ACL (e.g. Network File System (NFS), Andrew File System (AFS), xFS [48]). When public key cryptography is used, systems typically support either UNIX-style ACL (e.g. SFS [27]) or certificate authorization (e.g. DisCFS [28]). These systems assume that *file servers are trustworthy*, but the network is not secure; thus, data confidentiality is guaranteed through securing the channel (e.g. SSL). If the servers are not trustworthy, we can either rely on other semi-trusted servers as in Cobalt [46] or use cryptographic encryption to preserve data confidentiality as in Cryptographic File System (CFS) [4], Plutus [24], and SiRiUS [19]. A detailed survey of recent decentralized access control has been presented in this survey paper [29].

In the CFS approach, authentication is typically undertaken using public key cryptography where a user's public key is used as an ID, and a digital certificate is used for authentication. CFS uses a single key for encryption (coarse-grained, e.g. directory/volume) and is dependent on the underlying file system for write authorization [4]. Later variants use a lockbox to protect the keys (with more fine-grained access control) and introduce several mechanisms for verifying the write operations without depending on the underlying file system [19,24]. In particular, SiRiUS [19] assumes that the network storage is untrusted and provides its own read-write cryptographic access control for file-level sharing. It permits a file to be shared by multiple individuals or groups using a common file encryption key that is encrypted again using each user/group's public key.

Given that attribute-based encryption (ABE) is designed to provide fine-grained, expressive access control, several existing works have used ABE for *read-only* content sharing over untrusted storage [1,49,50]. In particular, Yu et al. [50] used a key policy ABE (KP-ABE) to provide privacy-aware content sharing over untrusted cloud storage and Proxy Re-Encryption (PRE) to delegate the task of re-encryption on cloud servers. While PCN is considered to be a cryptographic file system, unlike existing systems, PCN provides fine-grained expressive access control using CP-ABE in a fully distributed environment with untrusted nodes and it allows file owners to set up expressive *read and write* access policies based on attributes (e.g.

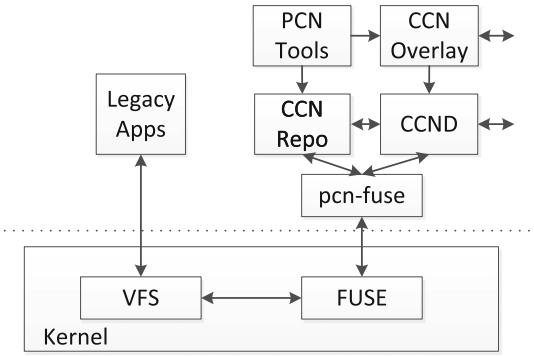

Fig. 3 PCN architecture

college friends, family members, etc.). Furthermore, none of the aforementioned systems provide *secure binding* between the name and data; thus, the channel must be secured in order to prevent man-in-the-middle attacks.

6 Prototype Implementation

We implemented a PCN prototype in the Linux and Android platforms, and further integrated the prototype with FUSE, which is a user-level file system to support legacy applications in Linux. As shown in Fig. 4, the PCN tools include *pcn-init, pcn-intro, pcn-abe-enc*, and *pcn-broswer*. The *ccn-overlay* tool maintains the CCN overlay network. The *pcn-fuse* tool implements the basic VFS file operations, which support legacy applications (Fig. 3).

As depicted in Fig. 4, a PCN user first initializes a personal namespace using the *pcn-init* tool. This tool generates a private-public key pair and prompts the user to name their namespace. The key pair must be securely disseminated to the other personal devices. The tool runs a local area rendezvous tool to locate other devices on the local area network and installs the key pair securely. The device information will be reported to the CCN overlay client that configures its local CCN daemon (CCND) [6]. Because the current CCNx codebase only supports manually configured, static network topologies, we implemented an overlay network client (called *ccn-overlay*) that builds and maintains an overlay network based on social relationships. Whenever the network topology changes, an overlay client uses external commands to reconfigure the local CCND. The prefix announcement is disseminated through the overlay clients because the current CCNx codebase does not fully support the prefix announcement feature. Each client periodically exchanges ping messages to verify whether its neighboring nodes are alive. Recall that we have three types of prefix announcements, i.e. regular prefix, modification, and key revocation announcements. Besides the device initialization, users can establish a trust relationship using the *pcn-intro* tool, which is based on UIA's device management UI tool [15].

In PCN, non-privileged users can mount a namespace into their local user directories, and the legacy applications can seamlessly access the files in the PCN namespace. To this end, we use FUSE, which is a loadable kernel module that allows non-privileged users to create their own file systems without editing the kernel code by running file system operations in the user space while the FUSE module provides a bridge to the actual kernel interfaces. We implemented key virtual file system (VFS) operations in the *pcn-fuse* tool: *getattr, getdir, mkdir, rename, open, release, read*, and *write*. In particular, when an application reads a file,

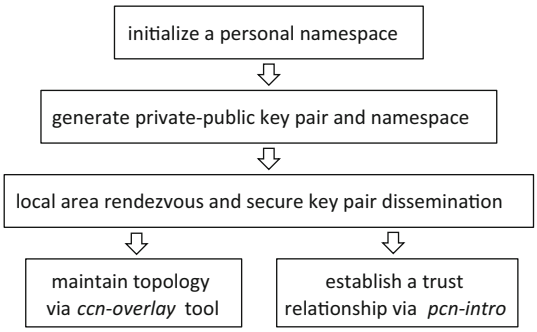

Fig. 4 Flowchart of topology and trust relationship establishment

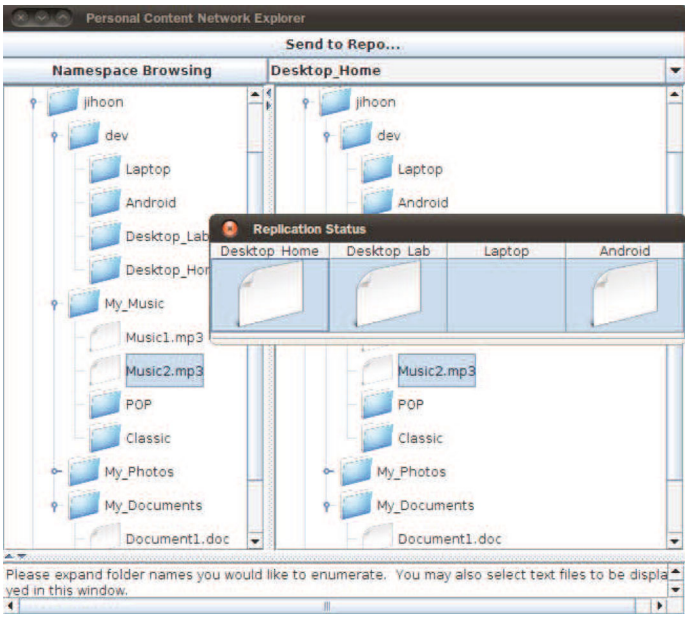

Fig. 5 Content management tool (*pcn-browser*). The left panel is a namespace browser, and the right panel is the replicated namespace at a specific device (*Desktop_Home*). By right clicking the file (*Music2.mp3*), a user can see the replication status across different devices

our user-level module downloads chunks through CCN. When a file is modified, we restrict the modification to being committed to its local repository (a new version is created) only if the file is finally released. Then, the *pcn-fuse* tool sends the prefix announcement with a modification mark via the *ccn-overlay* daemon.

For content management, we implemented *pcn-broswer* through significantly modifying the *ccnbrowser* tool in the original CCNx codebase (see Fig. 5). The *pcn-browser* tool allows users to browse any namespace and also displays the file locations. A user can easily issue a replica management command through selecting a file/directory and a target device.

For ABE support, we used the CP-ABE toolkit [9]. A file published in the local repository can be encrypted using *pcn-abe-enc*. This tool communicates with the local CCN repo daemon, encrypts the named file, and re-publishes the file into the repository. The ABE keys are

Table 1 CP-ABE performance of laptop (L) and nexus one (M) in milliseconds: master key (MK) setup and secret key (SK) generation with k number of attributes

	MK setup	SK: 5	SK: 10	SK: 15
Laptop	166 (±0.2)	531 (±0.4)	913 (±0.2)	1,343 (±1.9)
Mobile	354 (±0.9)	2,068 (±0.5)	3,981 (±0.5)	5,947 (±0.3)

stored in a user's local keystore (e.g. *.ccnx* at home). If a file is encrypted, the *ccn-fuse* tool automatically decrypts the file and returns the plaintext to the reader. It accesses the user's local keystore for decryption. We ported the CP-ABE toolkit to the Android platform via cross-compilation. Because the CCNx codebase supports the Android platform, we integrated the basic PCN tools into the mobile platform.

7 Evaluation

We present our preliminary system evaluation that answers the following questions: (1) What is the overhead of ABE? (2) What is the detailed performance of each component used in PCN? (3) Given realistic user traces, what is the overhead of PCN (e.g. routing table size, update overhead, etc.)?

In order to provide secure personal sharing, we designed our implementation to incur minimal overhead to the existing CCNx codebase. While a complete evaluation of the CCNx method traces is outside the scope of this paper, our experience demonstrates that the CCNx performance improves with every release. We analyzed the performance of providing security using ABE in the three major areas of (1) key setup and generation, (2) encrypting and storing content, and (3) retrieving and decrypting content. In order to model user behavior, we measured the performance using a mobile device (the Android Nexus One from Qualcomm Snapdragon with 1 GHz CPU and 512 MB of RAM), a laptop (Dell Inspirion 9400 with Intel dual core 2 GHz CPU, 2 GB of RAM, and Intel WiFi Link 5300 that runs Ubuntu 10.10 with Linux 2.6.35), and a desktop computer (Apple iMac with Intel i5-2500s, 2.7 GHz CPU, and Broadcom Gigabit Ethernet that runs Ubuntu 10.10 with Linux 2.6.35). The measured device-to-device TCP performance using Iperf is given as follows (over an average of 10 trials with a 95 % confidence interval): Nexus One to laptop over WiFi: 8.21 Mbps (±0.02), laptop to Nexus One over WiFi: 8.00 Mbps (±0.01), desktop computer (wired) to laptop (WiFi): 10.34 Mbps (±0.02).

7.1 Overhead of ABE

Table 1 presents the master key setup delay and secret key generation delay as a function of the number of attributes. The results demonstrate that the delay almost linearly increases with the number of attributes. The master key setup is independent of the number of attributes, and that of the laptop and Nexus One is given as 166 and 354 ms, respectively. Based on our user experience, the key setup delay in the laptop was not noticeable, but that of the mobile device was not ignorable. However, the key setups are not frequent events in PCN and these can be undertaken prior to real time use. Considering the advancements in mobile hardware technology, the key setup delay will be ignorable in the near future.

Table 2 Breakdown of retrieval time (in milliseconds) of a file

	1 KB	10 KB	100 KB	1 MB	10 MB	100 MB
[D2L] CCNx retrieve	1,152 (±0.4)	1,225.8 (±1.0)	1,410 (±2.5)	2,102.2 (±2.3)	11,085.4 (±16.3)	80,593.8 (±139.1)
Local ABE pri-key	8.2 (±0.1)	10.2 (±0.1)	9.8 (±0.)	9.4 (±0.2)	13.6 (±0.1)	16.6 (±0.1)
Remote ABE pub-key	358 (±0.1)	343.6 (±0.1)	346 (±0.2)	348.6 (±0.2)	346.4 (±0.2)	348.4 (±0.1)
AES key decrypt	37.8 (±0.3)	37.7 (±0.2)	37.8 (±0.4)	36.8 (±0.9)	37.3 (±0.5)	37.1 (±0.5)
Content decrypt	1.0 (±0.1)	1.0 (±0.1)	3.2 (±0.1)	35.0 (±0.2)	380.2 (±1.6)	3,946.7 (±0.8)
[L2M] CCNx retrieve	784.8 (±0.5)	973.6 (±0.9)	1,157.8 (±0.5)	2,273.2 (±4.3)	10,751 (±18.5)	1,06,752.6 (±154.6)
Local ABE pri-key	37.6 (±0.1)	39.6 (±0.1)	38.6 (±0.0)	37.6 (±0.1)	39 (±0.0)	39.4 (±0.1)
Remote ABE pub-key	527 (±0.5)	533 (±0.2)	532.2 (±0.2)	536.4 (±0.2)	538.8 (±0.1)	539 (±0.1)
AES key decrypt	425.8 (±0.4)	428.6 (±0.3)	428.1 (±0.8)	427.1 (±1.3)	425.6 (±3.3)	433.6 (±6.1)
Content decrypt	2.1 (±0.1)	19.9 (±0.3)	120.0 (±0.2)	402.0 (±1.1)	3,414.9 (±2.3)	20,713.4 (±5.3)
[D2M] CCNx retrieve	706.4 (±0.1)	914.2 (±0.6)	1,146.2 (±0.3)	2,096.2 (±1.1)	10,259.4 (±26.6)	96,724.2 (±218.5)
Local ABE pri-key	35 (±0.1)	36.2 (±0.1)	37.6 (±0.1)	38.6 (±0.1)	39.2 (±0.1)	39.6 (±0.1)
Remote ABE pub-key	532.4 (±0.1)	425.8 (±0.1)	433 (±0.1)	432.4 (±0.1)	431.8 (±0.1)	431.4 (±0.1)
AES key decrypt	427.1 (±5.3)	419.6 (±7.1)	429.7 (±7.3)	435.1 (±10.1)	429.3 (±6.1)	435.8 (±11.3)
Content decrypt	2.4 (±0.1)	18.1 (±0.4)	128.0 (±0.1)	387.1 (±1.3)	3,371.1 (±2.5)	21,001.4 (±4.1)

Each result is the mean of five trials with a 95 % confidence interval. Each trial was run by setting the CCN cache size to 0 (CCND_CAP = 0) and restarting the CCND between each run in order to reset the local cache

D2L Desktop computer to laptop, *L2M* laptop to nexus one, *D2M* desktop computer to nexus one

7.2 Detailed Performance of Each Component

We measured the performance of the remote content retrieval (single hop). Retrieving/decrypting involves six steps: (1) FUSE open/read call (in laptops only), (2) CCN data retrieval over a remote node, (3) ABE local repository key look-up, (4) ABE public key retrieval from a remote node, (5) CP-ABE decryption of an AES key, and (6) content decryption with AES-256.

As shown in Table 2, the most time consuming operations are the CCN retrieval and CP-ABE decryption, and they have a positive relationship with the file size. Unlike the CCN retrieval and CP-ABE encryption, the ABE local repository key look-up, ABE private/public key retrieval from a remote node, and CP-ABE decryption of an AES key increase linearly with the file size. The cost of the FUSE operations took less than a few milliseconds, and we did not report a delay in the table. Based on the results, the laptop to Nexus One (L2M) and desktop computer to Nexus One (D2M) exhibited similar behaviors in the CCN retrieval and CP-ABE decryption. However, the desktop computer to laptop (D2L) was 2–5 times faster than L2M and D2M. These differences were caused by the lower computation power of the Nexus One.

7.3 PCN Overhead

Given that there is no available realistic trace of personal content management, we collected the recently accessed files from Windows PCs. We drew participants from researchers and graduate students, and collected data from a total of 31 participants (25 male, 6 female). The participants varied in age: 22 were between 21–30 years old; 14 were between 31 and 40 years old; 1 was between 41 and 50 years old. Windows maintains a link to each file accessed in a designated directory (e.g. Windows XP in the Recent directory). A symbolic link file is automatically created when the target file is opened for the first time; also, whenever the target file is accessed, the link's modification time is automatically updated (refreshed). Although this data set does not provide a real-time trace, it provides valuable information for system evaluations (e.g. the average number of prefixes and content access/update patterns). Our investigation demonstrates that the time span of recently accessed files typically ranges from one to two months. There was only one participant who recently erased their access history, and we excluded this participant from the analyses.

For each user, we plot the number of distinct files and the number of distinct directories (Fig. 6). The number of distinct directories is similar to the number of prefixes that PCN needs to announce (assuming that updates occur therein). The number of distinct files provides a rough usage activity level: users 1 and 2 were less active, whereas users 29 and 30 were more active than the regular users. It appears that most participants accessed 120–140 files over the time span of one to two months; this number is conservative in that it only counts the files that were accessed via file browsers. The figure also demonstrates that the number of directories was typically <60. This number is closely related to the number of distinct prefixes announced by the user (which can also be aggregated, e.g. */My Doc/*).

For a given social network with a hop limit of k, the routing table size in an intermediate node is proportional to the number of distinct prefixes. The following is a simple back-of-the-envelope calculation. Assuming that there is an average branching factor of 100 and a hop limit of $k = 1, k = 2$, a node needs to keep the entries of a total of 100 and 10,000 people, respectively. If a user has 60 prefixes, each of which is 100 bytes, the total number of entries is 6,000 and 600,000, respectively, and the total storage demand is 0.6 MB and 60 MB, respectively. The overhead of $k = 2$ can be reduced if we selectively include friends of

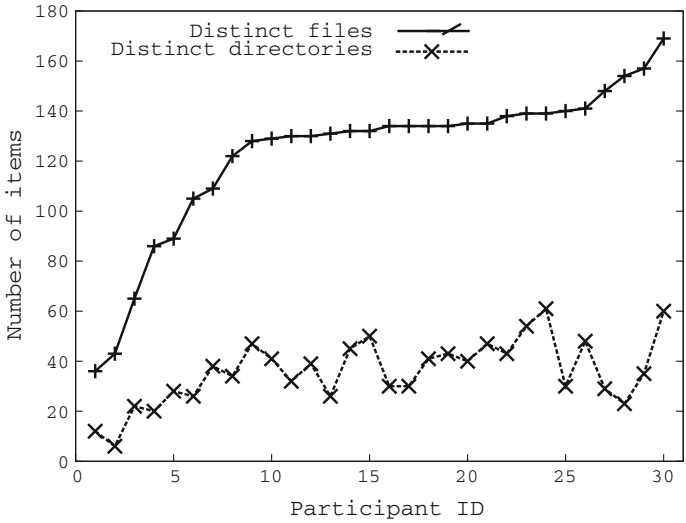

Fig. 6 Recently accessed files and directories

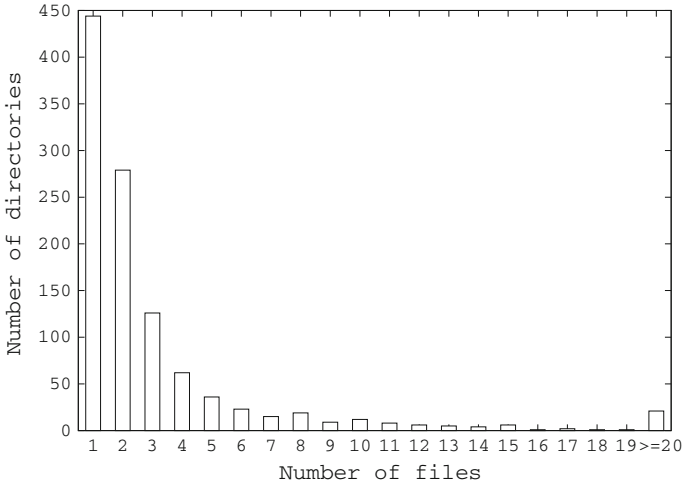

Fig. 7 Number of files accessed per directory

friends because the purpose of extending more than one hop is to limit the impact of network address translation (NAT).

We also analyzed how many files are accessed per directory (Fig. 7). The figure demonstrates that the number of distinct files accessed per directory is highly skewed, and only a small number of files are accessed per directory. For example, in 41 % of directories, only a single file was accessed, and the fraction of directories that had more than five distinct files accessed was only 10.2 %. This result explains why participants have a high number of directories as opposed to a distinct number of files.

Finally, in order to analyze how people interact with files, we measured the time difference between the link creation and modification (i.e. access time span of a file) and plotted the results in Fig. 8. Recall that a link is modified (refreshed) whenever the target file is accessed. Interestingly, the figure demonstrates that the file access patterns of personal content is almost

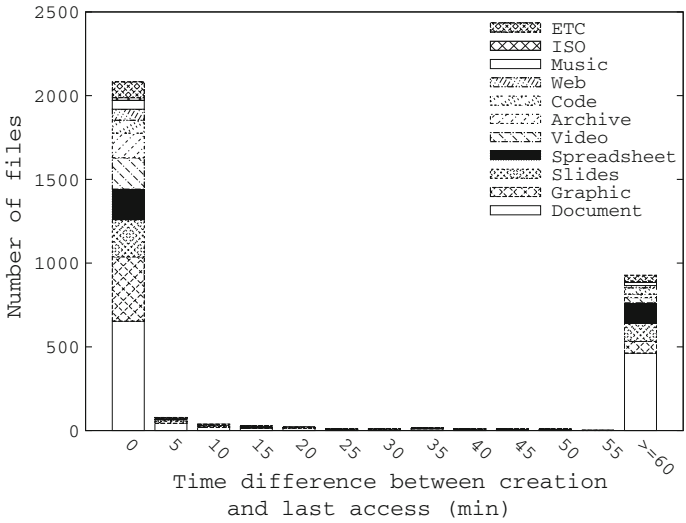

Fig. 8 Distribution of access time span based on file types

bimodal; that is, quite a significant percentage of files are only accessed once and are read-only (i.e. time difference is 0, 64.1 %), and another significant percent of files are repeatedly accessed over the time span of longer than one hour (i.e. the time difference is greater than 60 min, 28.6 %). The remainder of the files (7.3 %) has intermediate access and are likely be repeatedly accessed over time. The files that are repeatedly accessed include both read-only and read-write accesses. It is expected that the percentage of read-write accesses would be significantly smaller than that of read-only accesses. This also indicates that the overhead of maintaining consistency over personal content networking in practice would be minimal (e.g. only a small number of files are updated over the course of a day) (Table 3).

8 Discussion

8.1 Security Attacks

PCN shares the security benefits of CCN because it is pull-based content retrieval and uses secure binding, thereby effectively thwarting distributed denial of service attacks, request flooding attacks, and man-in-the-middle attacks [21]. While PCN introduces new features such as extra prefix announcements (modification and revocation) and content updates, PCN's explicit prefix protection provides a restriction that only authorized users can replicate a named prefix. Moreover, PCN provides a limitation that replicated content can only be updated by users with explicit write permissions. Thus, a user can neither request content replication nor inject updates without explicit permissions from the content owner, as illegitimate requests are automatically discarded by the intermediate PCN nodes.

8.2 Semantic Versus Hierarchical Naming

PCN uses single persistent hierarchical naming. An alternative is semantic naming as in semantic file systems where semantic information is added to file systems and semantic

Table 3 Feature comparison: delay tolerant networking (DTN), single persistent or device persistent (SP/DP), flat/hierarchical (F/H), public-key cryptography (PKC)

	Naming	DTN	Topology	Replication unit	Update	Trust	Access control	Secure binding
Ficus	SP+H	Yes	P2P	File/Dir	Yes	–	ACL	–
BlueFS/ EnsemBlue	SP+H	Yes	C/S	File	Yes	–	ACL	–
UIA/Eyo	DP+H	Yes	P2P	–	–	–	ACL	–
Personal RAID	SP+H	Yes	P2P	Volume	Yes	–	–	–
Footloose	SP−F	Yes	P2P	File	Yes	–	–	–
DisCFS	SP−H	No	C/S	Volume	Yes	KeyNote	Certs	–
Bayou	SQL	Yes	P2P	Volume	Yes	PKI	Certs	–
Plutus/ SiRiUS	SP−H	No	C/S	–	–	PKI	Certs/ Enc-PKC	–
PAST	SP−F	No	P2P	File	Yes	PKI	Certs/ Enc-PKC	–
CCN	SP−H	Yes	P2P	File	No	PKI	Certs/ Enc-PKC	Yes
PCN	SP−H	Yes	P2P	File/Dir	Yes	SPKI	Certs/ Enc-ABE	Yes

attribute queries are used to locate files [17,18,40]. An extreme case would be using a single flat directory where each file has an arbitrary unique name, and a user can search for any files using semantic queries; the user can maintain views across multiple devices [40]. However, extensive human subject studies in the personal information management field have demonstrated that a majority of people want to search by browsing a hierarchical file system (called "orienteering behavior") and use semantic queries (e.g. desktop search tools) as a last resort [22]. This results from browsing relying more on recognition and people use browsing to reduce and distribute the amount that must be recalled [26]. Given that only a handful of applications require semantic naming (e.g. music players), it is more efficient to implement semantic data access as an application layer service over PCN.

8.3 Energy Efficiency

The PCN system includes battery powered personal devices. Battery limited devices need to constantly listen to announcement messages, which prevents them from switching to a sleep mode for power saving. Recall that whenever there are updates, PCN broadcasts the messages to the k-hop neighbors in the overlay network. One solution to this problem is to introduce a *proxy server* in an AC powered device (e.g. desktop computer, laptop, etc.). A mobile device can re-configure the underlying overlay network topology such that messages always travel through the local proxy server. The local proxy buffers all incoming announcements. Then, the mobile client periodically wakes up and pulls the aggregated announcements.

8.4 Interest-Based Push for Synchronization

In our prototype implementation, we used prefix announcements to notify replica nodes of content updates. An alternative to this approach is to use *interest solicitation*, as recommended in the NDN proposal [51]. The node that updated the content sends an interest solicitation

packet to the replica nodes that are interested in receiving the updated content. Then, those interested replica nodes will send an interest packet requesting the updated content. For efficient synchronization, the interest solicitation packet includes detailed information about the updated content because the prefix announcement was augmented in PCN.

8.5 Private PCN

For security reasons, a user could have two different namespaces: one for private access and the other for shared access. The private PCN is not visible to other users; thus, a user can simplify the access control, e.g. just setting a single attribute for content encryption. Given that a large percentage of content is personal use only, it is expected that a private PCN network could lower the burden of content management.

8.6 Offline Devices

If devices are offline, a user cannot browse the content stored in the devices. In order to aid content retrieval from offline devices, PCN can take a similar approach to that used in Eyo [44]. Each device periodically pulls the content lists of the other devices and stores them in its local repository. Given this information, PCN nodes can tell which device has a file and, thus, a user can access the file from offline devices.

8.7 Overlay Construction of the Devices Behind NATs

A device may be behind a NAT, and it cannot actively participate in the overlay network. In this case, the device can be connected through a relay node that is not behind a NAT and is sufficiently stable (e.g. the device is online 90 % of the time). The NAT can potentially reduce the number of peering devices, thus lowering the connectivity among devices. We can increase the connectivity through allowing devices to exchange IP addresses of k-hop friends' devices. For example, when $k = 2$, Alice can connect to Bob and also to Bob's friends. Furthermore, computing resources in cloud systems could be utilized to increase connectivity, e.g. a personal account in Dropbox could serve as an intermediate node in PCN.

9 Conclusion

We designed and implemented the personal content networking (PCN) platform. We extended the CCN to build a basic framework for distributed content management with replication and updates, and then we implemented a *secure content-centric access control* mechanism using the recently proposed cryptography tool called attribute-based encryption (ABE) that permits selective content sharing over untrusted nodes. The primary departure from prior work is that PCN supports ABE-based secure *read-write* operations over untrusted devices and *secure-binding* between the name and data. We built a PCN prototype through integrating the whole system using a user level file system, and we demonstrated its feasibility through performance measurements and trace analysis.

Acknowledgments This work was partly supported by the ICT R&D program of MSIP/IITP [1391104004, Development of Device Collaborative Giga-Level Smart Cloudlet Technology].

References

1. Baden, R., Bender, A., Spring, N., Bhattacharjee, B., & Starins, D. (2009). Persona: An online social network with user-defined privacy. In *SIGCOMM'09*. Barcelona, Spain.
2. Balasubramaniam, S., & Pierce, B. C. (1998). What is a file synchronizer? In *MobiCom*.
3. Bethencourt, J., Sahai, A., & Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *SP'07*. Oakland, CA.
4. Blaze, M. (1993). A cryptographic file system for unix. In *CCS*. Fairfax, VA.
5. Burnside, M., Clarke, D., Devadas, S., Rivest, R. (2002). Distributed SPKI/SDSI-based security for networks of devices. Technical report, MIT Laboratory for Computer Science.
6. CCNx (2012). Codebase http://ccnx.org
7. Chothia, T., & Chatzikokolakis, K. (2005). A survey of anonymous peer-to-peer file-sharing. In *Proceedings of the 2005 international conference on embedded and ubiquitous computing*.
8. Clarke, D., Elien, J. E., Ellison, C., Fredette, M., Morcos, A., & Rivest, R. L. (2001). Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, *9*(4), 285–322.
9. CP-ABE (2012). Implementation. http://acsc.cs.utexas.edu/cpabe/.
10. Dearman, D., & Pierce, J. S. (2008). It's on my other computer! Computing with multiple devices. In *CHI'08*. Florence, Italy.
11. Dohrmann, S., & Ellison, C. M. (2002). Public-key support for collaborative groups. In *Annual PKI Research Workshop*. Hanover, NH.
12. Druschel, P., & Rowstron, A. (2001). PAST: A large-scale persistent peer-to-peer storage utility. In *HotOS'01*. Schloss Elmau, Germany.
13. Ellison, C. M. (1996). Establishing identity without certification authorities. In *USENIX'96*. San Diego, CA.
14. Ferreira, R., Grama, A., & Jagannathan, S. (2005). Plethora: An efficient wide-area storage system. In *High performance computing—HiPC 2004*.
15. Ford, B., Strauss, J., Lesniewski-Laas, C., Rhea, S., Kaashoek, F., & Morris, R. (2006). Persistent personal names for globally connected mobile devices. In *OSDI'06*. Seattle, WA.
16. Filesystem in Userspace. (2012). http://fuse.sourceforge.net.
17. Geambasu, R., Balazinska, M., Gribble, S. D., & Levy, H. M. (2007). HomeViews: Peer-to-peer middleware for personal data sharing applications. In *SIGMOD'07*. Beijing, China.
18. Gifford, D. K., Jouvelot, P., Sheldon, M. A., James, W., & O'Toole, J. (2007). Semantic file system. In *SOSP'91*. Beijing, China.
19. Goh, E. J., Shacham, H., Modadugu, N., & Boneh, D. (2003). SiRiUS: Securing remote untrusted storage. In *NDSS'03*.
20. Henderson, S., & Srinivasan, A. (2009). An empirical analysis of personal digital document structures. In *HCII'09*. San Diego, CA.
21. Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., & Braynard, R. L. (2009). Networking named content. In *CoNEXT'09*. Rome, Italy.
22. Jones, W. (2007). Personal information management. *Annual Review of Information Science and Technologys*, *41*(1), 453–504.
23. Jones, W., Phuwanartnurak, A. J., Gill, R., & Bruce, H. (2005). Don't take my folders away! Organizing personal information to get things done. In *CHI'05*. Portland, OR.
24. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., & Fu, K. (2003). Plutus: Scalable secure file sharing on untrusted storage. In *FAST'03*. San Francisco, CA.
25. Kent, S., Lynn, C., & Seo, K. (2000). Secure border gateway protocol (S-BGP). *IEEE JSAC*, *18*(4), 582–592.
26. Lansdale, M. W. (1988). The psychology of personal information management. *Applied Ergonomics*, *19*(1), 55–66.
27. Mazieres, D., Kaminsky, M., Kaashoek, M. F., & Witchel, E. (1999). Separating key management from file system security. In *SOSP'99*. Charleston, SC.
28. Miltchev, S., Prevelakis, V., Ioannidis, S., Ioannidis, J., Keromytis, A. D., & Smith, J. M. (2003). Secure and flexible global file sharing. In *USENIX*.
29. Miltchev, S., Smith, J. M., Prevelakis, V., Keromytis, A., & Ioannidis, S. (2008). Decentralized access control in distributed file systems. *ACM Computing Surveys*, *40*(3), 10.
30. Muthitacharoen, A., Morris, R., Gil, T. M., & Chen, B. (2002). Ivy: A read/write peer-to-peer file system. ACIM *SIGOPS Operating Systems Review*, *36*(SI), 31–44.
31. Nightingale, E. B., & Flinn, J. (2004). Energy-efficiency and storage flexibility in the blue file system. In *OSDI'04*. San Francisco, CA.

32. Page, T. W., Guy, R. G., Heidemann, J. S., Ratner, D., Reiher, P., Goel, A., et al. (1998). Perspectives on optimistically replicated peer-to-peer filing. *SPE*, *28*(2), 155–180.
33. Paluska, J. M., Saff, D., Yeh, T., & Chen, K. (2004). Footloose: A case for physical eventual consistency and selective conflict resolution. In *WMCSA'04*. Lake District National Park, UK.
34. Pedersen, T. P. (1991). Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91*. Santa Barbara, CA.
35. Peek, D., Flinn, J. (2006). EnsemBlue: Integrating distributed storage and consumer electronics. In *OSDI*.
36. PEM. (1993). http://en.wikipedia.org/wiki/Privacy-enhanced_Electronic_Mail.
37. Personal Content and Home Network Storage. (2007). the Perfect Storm, Tom Coughlin.
38. Reiher, P., Heidemann, J. S., Ratner, D., Skinner, G., & Popek, G. J. (2004). Resolving file conflicts in the ficus file system. In *USENIX'94*. Boston, MA.
39. Rivest, R. (1998). Can we eliminate certificate revocation lists? In *In financial cryptography*.
40. Salmon, B., Schlosser, S. W., Cranor, L. F., & Ganger, G. R. (2009). Perspective: Semantic data management for the home. In *FAST'09*. San Francisco, CA.
41. Sandhu, R. S., & Samarati, P. (1994). Access control: Principle and practice. *IEEE Communications Magazine*, *9*(32), 40–49.
42. Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H., & Steere, D. C. (1990). Coda: A highly available file system for a distributed workstation environment. *IEEE Transaction on Computers*, *39*(4), 447–459.
43. Sobti, S., Garg, N., Zhang, C., Yu, X., Krishnamurthy, A., & Wang, R. Y. (2002). PersonalRAID: Mobile storage for distributed and disconnected computers. In *FAST'02*. Monterey, CA.
44. Strauss, J., Lesniewski-Laas, C., Paluska, J. M., Ford, B., Morris, R., & Kaashoek, F. (2009). Device transparency: A new model for mobile storage. In *HotStorage'09*. Big Sky, MT.
45. Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., & Hauser, C. (1995). Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP'95*.
46. Veeraraghavan, K., Myrick, A., & Flinn, J. (2008). Cobalt: Separating content distribution from authorization in distributed file systems. In *FAST'08*.
47. X.509. (1988). http://en.wikipedia.org/wiki/X.509.
48. xFS. (2012). http://xfs.org/.
49. Yu, S., Ren, K., & Lou, W. (2008). Attribute-based content distribution with hidden policy. In *NPSec'08*. Orlando, FL.
50. Yu, S., Wang, C., Ren, K., Lou, W. (2010). Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM'10*.
51. Zhang, L., et al. (2010). Named data networking (NDN) project. Technical report, PARC technical report NDN-0001.

Uichin Lee is an assistant professor in the Department of Knowledge Service Engineering at Korea Advanced Institute of Science and Technology (KAIST). He received a B.S. in computer engineering from Chonbuk National University in 2001, an M.S. degree in computer science from KAIST in 2003, and a Ph.D. degree in computer science from the University of California at Los Angeles (UCLA) in 2008. Before joining KAIST, he was a member of technical staff at Bell Laboratories, Alcatel-Lucent until 2010. His research interests include distributed systems and mobile/pervasive computing.

Joshua Joy is a first year Ph.D. student in the Network Research Lab at the University of California, Los Angeles (UCLA) under the guidance of Dr. Mario Gerla. He received his M.S. degree in Computer Science from UCLA in 2012. His research areas include future internet architecture, mobile cloud computing, security, and privacy.

YoungTae Noh is a Software Engineer at Cisco Systems, Inc. Prior to joining Cisco Systems, he received his B.S. in computer science from Chosun University in 2005, an M.S. degree in Information and Communication from Gwangju Institute of Science Technology (GIST) in 2007, and a Ph.D. in computer science at University of California, Los Angeles (UCLA) in 2012. His research areas include data center networking, wireless networking, future Internet, and mobile/pervasive computing.