On Grappling with Meta-Information in the Internet[*]

Tom Callahan[†], Mark Allman[‡], Michael Rabinovich[†], Owen Bell[†]
[†]Case Western Reserve University, [‡]International Computer Science Institute
{trc36,michael.rabinovich,oab7}@case.edu, mallman@icir.org

ABSTRACT

The Internet has changed dramatically in recent years. In particular, the fundamental change has occurred in terms of who generates most of the content, the variety of applications used anl d the diverse ways normal users connect to the Internet. These factors have led to an explosion of the amount of user-specific meta-information that is required to access Internet content (e.g., email addresses, URLs, social graphs). In this paper we describe a foundational service for storing and sharing user-specific meta-information and describe how this new abstraction could be utilized in current and future applications.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design

General Terms

Design

Keywords

Meta-information, User-centric networking, Architecture, Abstractions

1. INTRODUCTION

The Internet looks very different today than it did 15–20 years ago. In particular, we highlight three trends. First, user-generated content has outgrown its humble beginning in the form of bulletin boards and has emerged as a significant presence in the Internet landscape. Normal (non-expert) users have become significant content generators alongside traditional content-providing services (e.g., news, weather and e-commerce sites). Among the Internet's most popular services are those that are mere frameworks to be populated with content as the users see fit (photo and video distribution, blogs, social networking services, etc.). Second, rather than simply sending email messages and looking up information, people are now using a myriad of applications to engage in rich interactions with each other, e.g., by sharing information (e.g., via photo or video sharing services), participating in live communication (e.g., audio/video chats, instant message, multi-player games) and finding each other (e.g., social networking sites). Third, users access the network from an ever-increasing number and variety of devices and access points (e.g., computers at work and home, laptops in coffee shops, smartphones from anywhere).

The increased heterogeneity in terms of content providers, applications and access methods employed by users to obtain and provide information on the Internet is a testament to the power and flexibility of the basic Internet architecture. However, the situation has also created a large amount of user-specific meta-information that is required to access the actual content[1] such as URLs for photos, videos and blogs; buddy lists for instant messaging and voice-over-IP systems; cryptographic certificates; email addresses; relationship information; etc. This meta-information explosion is born out of necessity, but leads to a mess of formats and redundant data scattered across applications and devices.

Current applications cope with meta-information in two basic ways. First, applications generally cope with their own information reasonably well (e.g., email programs track email addresses, calendar tools track calendar subscriptions, etc.). Further, applications are sometimes given pair-wise capability to interact with each other (e.g., a social networking site bootstrapping friends from an address book application). Second, applications can utilize cloud computing platforms to aggregate meta-information at some central point in the network that is accessible from any connected device. This centralized approach removes the need individual devices have for information at the cost of trusting a third-party to hold users' data. Given these two suboptimal paths, we view the meta-information explosion as an opportunity to build a new primitive for the Internet that is tasked with coherently, comprehensively and securely dealing with arbitrary user meta-information.

In this paper we describe the Meta-Information Storage System (*MISS*) which is a foundational service that provides users and applications with a new *architectural abstraction* for dealing with meta-information. The system can both deal with the current mess and more crucially enable new functionality based on wide-spread ready access to meta-information. We note that the actual system we describe in this paper uses off-the-shelf components. Our contribution is not in designing some novel information store, but rather in the architecting of a service that offers a better way to organize meta-information within the network. Further, our goal is to design a "thin waist" sort of service (akin to the IP network protocol or the DNS naming system) that is at once general and simple, even if not always optimal. We view our

[*]ACM SIGCOMM Computer Communication Review, October 2011.

[1]In this paper we consider "content" to be both stored information such as a blog entry, as well as ephemeral communications such as an audio chat.

re-use of previously developed mechanisms to be a positive comment on the community's effort to design generic technology that is broadly applicable. We view this paper as representing another step in the process of creating generic, broadly useful services.

2. META-INFORMATION

The *MISS* system is tasked with dealing with *meta-information* which we define as a range of information that is not the ultimate content the user interacts with, but rather the information required to get at that content (names, social graphs, etc.). In the discussion below we describe *MISS* as both extensible and application-agnostic and therefore it is tantalizing to think about using the system to hold actual application content. While perhaps possible our focus is on meta-information and this drives the design and therefore the resulting system may be suboptimal for storing actual content. First, as sketched above, we designed *MISS* with the idea of it being a "thin waist" for meta-information. Therefore, the interface is sufficient for that task, but likely not nearly rich enough to support the general content transactions. Second, the volume of meta-information in relation to content is incomparable. A system for storing a relatively small amount of meta-information is feasible, but a general content store represents a larger and more complex task.

The *MISS* data store calls for each user to have a *collection* of meta-information.[2] Each collection is comprised of a set of *records*, each of which is identified within the collection by a name and type. The *MISS* system itself is agnostic to what is held in these records—except for a few pieces of information needed by the system itself—leaving applications to craft records as needed. As an example to aid the reader's intuition, consider a record in Alice's collection containing her current email address. When Bob sends an email to Alice, his email client can look her address up in Alice's collection at the time of email transmission. Whereas today's situation calls for Alice to tell Bob when she starts using a new email address, the late binding facilitated by *MISS* allows for Alice to arbitrarily change where her email is sent with no impact on Bob. This naming scheme is discussed in more detail in § 4.1.

Before describing collections and records in detail we sketch several requirements for the system. (1) The system should be *extensible* in terms of the types of meta-information stored to accommodate an evolving set of applications. (2) The system should allow the user to *control access* to their meta-information on a per-record basis (allowing for records to be private, public, shared within well-defined groups, etc.). (3) The system should provide for record *integrity* such that anyone with access to a record should be able to validate the record and that it belongs to the given collection.[3] (4) The system should be designed to allow for *portability* of meta-information with respect to where a user's information is hosted. I.e., collections should not be entangled with a particular service provider or meta-information hosting system.

[2]For ease of exposition we frame this paper in terms of each user having *one* collection. However, there is no reason a user cannot have multiple collections within *MISS*—e.g., for different roles they might play (e.g., home vs. work).

[3]Related to integrity is trust. We do not require nor preclude a rigorous trust system. *MISS* can accommodate both externally developed trust, as well as applications that rely on looser notions of trust such as "opportunistic personas" [5] and "leap-of-faith" security [6].

```
<miss_record>
  <name>myemail</name>
  <type>email-address</type>
  <expires>1278597127</expires>
  <signature> [...] </signature>
  <email-address>
    <ex1>alice@somenetwork.com</ex1>
  </email-address>
</miss_record>
```

Figure 1: Example *MISS* record.

Finally, we note that while *MISS* is charged with securely storing and sharing meta-information, that does not obviate the applications from conducting tasks such as access control (as they do now). For instance, a calendar available from a CalDAV server that is named within *MISS* (see § 4.1) will still require access controls at the server to limit who can access the calendar even if the record in *MISS* is encrypted such that only a small number of people can access the meta-information.

Collections: A collection is a container for all of a user's meta-information records. Collections do not have to be strictly tied to a single person, but could be used for organizations or groups, however, for convenience we discuss collections as belonging to individuals. Each collection is associated with and named by a user-generated cryptographic key pair (C_s,C_p) for the secret and public halves, respectively. The collection's name is a fingerprint of C_p. By building *MISS* on a cryptographic foundation we address the bulk of the above requirements. Naming collections with user-generated cryptographic keys allows for portability as collections can be generated opportunistically by users and hosted with any willing service provider and moved at will. Further, the cryptographic foundation naturally allows both access control via encryption and validation via cryptographic signatures. We note that our *key assumption* for *MISS* is that the users' key material remain safe. When this assumption holds the system cannot deliver undetectable fraudulent records. In § 3.3 we discuss coping with the failure of this assumption.

Records: Each record within *MISS* is identified by a type, a name and the collection identifier. We place no constraints on the type and name fields, considering each to be an arbitrary string. The name of each record is chosen by the user (or by an application on a user's behalf). Each record is encoded in XML [9] both (*i*) in pursuit of the extensibility requirement and (*ii*) because libraries for dealing with XML are widely available for a large set of languages, allowing for easy integration into existing and new applications. With the exception of a few standard fields needed to track records, the system is agnostic to the contents of the records. This encoding strategy allows for a high degree of flexibility in the construction of the records. Further, record types can be added and changed as applications evolve.

Finally we note that each record in *MISS* has an associated expiration time which serves two purposes: (*i*) placing temporary records in the system can be done in such a way that no explicit cleanup is required and (*ii*) explicit expiration times allow for consumers to cache records to save querying time and expense on each use of a record (caching is discussed further in § 3.1). This mechanism is analogous to the time-to-live scheme used by DNS.

Figure 1 shows a sample *MISS* record that defines Alice's current email address, per the example sketched above. The first few fields are standard to all *MISS* records and give the record's name ("myemail"), type ("email-address"), expiration time and the signature of the entire record. The collection that holds this record is not explicitly encoded in the record, but is needed to obtain and validate the record (as detailed in § 3.2). The remainder of the record (the "email-address" portion and enclosed address)is defined by an application and is irrelevant to *MISS*.

Since *MISS* is built on top of a cryptographic core the foundation for protecting users' privacy is in place and records can be encrypted such that only certain peers will be able to access the contents. While this allows users and applications to control the information placed into *MISS*, sometimes the mere presence of a record may violate a user's privacy. Therefore, encrypted records are only returned to users with access to decrypt the records.

We note that the *MISS* data model focuses on individual records and there is no index of the contents of a collection.[4] A requester must know what they are looking for (collection, record type and name) to form the request. This allows for record-level access control without carrying that forth to some form of index for enforcement. In addition, it offers a level of privacy in that while public records can be fished from the system, not readily publishing those records in an easy to find form prevents others from surfing through all of a user's meta-information. Finally, our focus on individual records keeps the interface between the meta-information servers and the clients simple (i.e., *get()/put()*) and only places minimal trust in the *MISS* servers themselves (see § 3 for a deeper discussion of the interface).

We also note that like transport protocol port numbers there are two general classes of record types: well-known and ad-hoc. The division allows for well-known rendezvous points and ad-hoc customized entries. Technically, the only difference between the types is that a registry[5] of well-known types and their corresponding record format will be kept (e.g., so two like applications can both understand the format). To avoid clashes in type names it is recommended that types expected to see broad use be registered. Additionally, ad-hoc types should be prepended with the name of the application to reduce the chances of collision (e.g., "Firefox-foo" instead of "foo"). Ultimately, however, applications must verify the record retrieved conforms to expectations and handle cases when it does not as an error.

3. SYSTEM OVERVIEW

We next provide an overview of the system we have designed and prototyped to store users' meta-information. Figure 2 shows a high-level diagram of the system and will be described in the remainder of this section. We note that while we believe the design presented in this section is the best realization of the engineering tradeoffs involved, other designs were considered and are possible. The main contribution of this paper is in the abstraction and not on the particulars of the system which could be engineered differently without impacting the provided foundation.

[4]Of course, various forms of index records could be constructed for various purposes since *MISS* itself is agnostic to the contents of a record.

[5]While we have not yet set up a registry the history of IANA suggests that it is possible and works well.

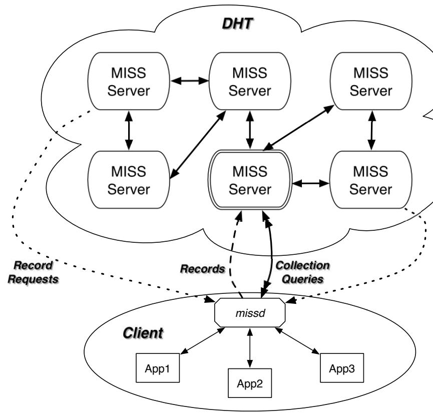

Figure 2: Conceptual diagram of *MISS* system.

3.1 Local Interface

The *MISS* system requires a local interface between applications and the global repository. As shown in Figure 2, this interface is implemented through a local service, *missd*, which runs on a user's behalf on the same device as the applications and provides several key functions and undertakes a number of common tasks, as follows. First, the *missd* provides applications with a general *put()/get()* interface to the meta-information in the overall system without requiring configuration and complexity for each application. Second, the *missd* holds the user's state (their own records, cached copies of others' records, etc.) and provides for meta-information management—e.g., as the user's records need to be added or updated in the global database (e.g., as expiration times near). Also, the *missd* handles common operations, such as signing and/or encrypting records, as necessary. In addition, the *missd* fetches other people's meta-information and can cache and pre-fetch these records to make access to often-used information quick.

We note that the simple *get()/put()* interface *missd* provides has proven sufficient for managing the meta-information in all applications we have worked through and/or implemented (see § 4). Therefore, to keep complexity low we have not constructed a richer interface based on hypothesized needs. That said, the interface may well need to be enlarged as needs arise.

3.2 Global Access

We now turn our attention to the structure of the global meta-information storage system. The system is comprised of the collection of *MISS* servers shown in the upper portion of Figure 2. These servers are presumed to be infrastructure-level nodes run by ISPs or institutions for their respective users (e.g., similar to email servers). Each collection of records is stored on at least one *MISS* server (and possibly several for robustness). Each *missd* is configured to know the *MISS* server(s)—denoted by the server outlined with a double-line in the figure—for its collections and stores its applications' records there. (The figure only shows one *MISS* server—i.e., no backup servers—associated with the *missd* for clarity.)

When querying the system for a record, the *missd* must first determine which *MISS* server holds the relevant collection. The *MISS* servers together form a distributed hash ta-

```
<miss_record>
  <name>collection</name>
  <type>miss</type>
  <expires>1278684469</expires>
  <signature> [...] </signature>
  <miss_collection>
    <owner>
      <name>Joe Smith</name>
      <email>jsmith@foo.com</email>
    </owner>
    <pub_key> [...] </pub_key>
    <server>128.1.2.3</server>
    <server>132.25.30.35</server>
    <server>68.45.100.7</server>
  </miss_collection>
</miss_record>
```

Figure 3: Example `master` record.

ble (DHT) that stores a mapping of the collection identifier to the particular *MISS* server(s) on which the records are stored in a `master` record. Therefore, to retrieve a meta-information record the *missd* first queries the DHT with the collection identifier to retrieve the `master` record—as denoted by the double-arrowed line in Figure 2. An example `master` record is shown in Figure 3. The record starts with the four standard fields (name, type, expiration time and signature). Next, the collection information is enumerated, including the collection owner's name, email address and public key. Finally, there are a list of *MISS* servers which hold the records in the collection. Once the *MISS* server(s) for a given collection has been identified, the client queries one of the *MISS* server directly for particular records within the collection—shown with the two dotted lines on the figure. Each record is identified by a three tuple (c, t, n) where c is the collection identifier, t is the type of record and n is the name of the record. It is presumed the servers will be queried in the given order, however, to support load-balancing the record may include additional information to better direct the *missd*'s choice (e.g., an indication to choose a random server).

As noted, *MISS* naturally supports the use of multiple *MISS* servers for robustness. This is akin to DNS' notion of listing multiple authoritative servers for particular names. When publishing a record the *missd* attempts to publish the same record on each of the *MISS* servers the user employs (and retries failed *put()*s, as necessary). The *MISS* servers independently age information out based on the expiration time and therefore no additional coordination is required to remove stale information.

It is possible that a user updates a record on one *MISS* server but cannot reach another *MISS* server (e.g., a backup) at that moment, which creates an inconsistency. However, (i) the inconsistency will be fixed as soon as the *missd* can re-establish a connection with the unreachable *MISS* server and (ii) while the lagging *MISS* server may not provide queriers with the most recent meta-information, any returned records will not be expired and therefore should still be valid (e.g., the given record could be used from some *missd*'s cache at that point). This system of loose consistency has proven reasonable within DNS and therefore we expect it will also be reasonable for *MISS*.

Since *MISS* queries represent a new lookup layer that must be used before various application operations begin, we must consider the added delay in performing these lookups. To combat this extra delay, the system can heavily use caching and pre-fetching. We expect expiration times to be relatively long in many cases. E.g., recording a name for an email address (as discussed in § 2 and § 4) with an expiration of one week would allow for both reasonable email address changes and record caching to prevent users from constantly looking up a common correspondent's current email address. Further, for regularly used items (which *missd* can easily track since all interactions happen via this component) records can be pre-fetched as the expiration time approaches. Using caching and pre-fetching means that for common records there will be only negligible delay imposed by *MISS*. However, the delay for looking up less commonly used records will remain (see experiments outlined in § 6 for initial data on the extra delay). While we do not have a good estimate of how many records people will utilize, we note that if we assume records are stable and time-out every week, that means pre-fetching one record per minute would allow a *missd* to keep 10K records fresh.

Finally, we note that the system readily supports the portability requirement outlined in § 2, as moving a collection from one server to another merely involves altering the pointer to the *MISS* server(s) in the `master` record.

3.3 Managing Collections

We now delve into three issues the system must deal with in terms of managing the stored meta-information.

Bootstrapping: The first problem *MISS* faces is a bootstrapping issue between users. To access another user's information (e.g., an email address as outlined in § 2) the user's collection's public key (or fingerprint thereof) is required. The collection ID can be transmitted in a number of ways that are already used to exchange contact information, e.g., email header fields, optional HTTP headers, meta-fields in HTML documents [14], email signatures, vCards, etc. In addition, we could setup well-known registries (built from *MISS* collections) such that people could map their collection ID to some human-understandable name in a well-known place. While the names would have to be unique within the registry, this is for bootstrapping only. After learning someone's collection ID it can be readily stored within one's own collection under some context-sensitive name (e.g., "Dad"). This allows for users to interact with the system in a natural way. As sketched in § 4.1 we have built a naming system within *MISS* that can handle such tasks.

Security: Recall that in § 2 we noted our crucial assumption is that the key material associated with a collection not be compromised. As long as this assumption holds, the *information* retrieved from *MISS* can be verified as legitimate by the consumer. An attacker compromising components of the system (e.g., *MISS* servers) can *prevent* (or slow) record retrieval, but cannot provide undetectable fraudulent information given that the consumer must have the collection's legitimate public key to request information.

However, in reality *the key material will be compromised* on occasion. Once an attacker takes possession of the user's private key, records can be fraudulently inserted and modified which could have far-reaching implications (e.g., redirecting email (see § 2 and § 4.1) to an account controlled by the attacker). In the limit, users can recover from this breach by simply populating a new collection and informing their peers out-of-band about the new collection. This is an effective, but high cost solution. Therefore, we design a reasonable—although not fool-proof—procedure to mitigate such events without requiring a start-from-scratch approach.

We extend `master` records in three ways. First, updating a `master` record *adds* a new copy of the record rather than *replacing* the old version (and all non-expired versions are returned in a lookup), preventing an attacker from simply over-writing legitimate information placed in the system. Second, a `master` record can *redirect* the requester to a different `master` record to indicate that the collection is migrating to a new collection ID (i.e., from a compromised collection to a fresh and uncompromised collection). Third, a `master` record advertises a so-called *secondary public key* (SPK) at all times. The SPK of various collections will be cached in the consumer's *missd* as a matter of course when using the system (i.e., before a collection is compromised). A `master` record indicating a redirect to another collection must be signed by the private key corresponding to the well-advertised SPK. Since this key is only needed for constructing the redirect (a rare occasion), the private key can be kept offline (e.g., on a USB stick), making it harder to compromise. Use of the SPK prevents the attacker from transitioning to new key material for the user by inserting a fraudulent redirect. Meanwhile, the legitimate user will possess the private half of the SPK and therefore be able to publish a legitimate redirect to a new and uncompromised collection.

This scheme does not provide iron-clad guarantees. For instance, if the collection's primary and secondary key material are stored together and both compromised, there is little recourse besides starting from scratch—and even that may be fruitless if the attacker still controls a user's device (and hence can readily steal the new key material). An attacker that controls both the primary key and part of the *MISS* infrastructure, e.g., the DHT node holding the user's `master` record, can thwart legitimate key transition by blocking legitimate redirects to keep consumers in the dark. Another attack involves consumers with no prior use of a collection and hence no SPK on file. In this case the SPK itself can be undetectably forged. The presence of legitimate redirects can of course raise a red flag in this case. Overall we note that these key transition attacks require a confluence of several events and/or breaches across the system and therefore raise the bar for the attacker significantly. While we believe that further mitigation of these attacks is possible with additional layers of monitoring (e.g., charging *MISS* servers with announcing key transitions to users' out-of-band or tasking the *missd* with re-requesting `master` records to build a history if none exists), we leave these mechanisms to future work.

Sharing Collections: We finally note that for a user to share their entire collection across multiple devices will require the *missd*'s state (e.g., records, collection IDs of peers the user interacts with, etc.) to be kept within the user's *MISS* collection.[6] Note: the state can be kept in an encrypted form such that no other user can access the data. In addition, the user's *MISS* key pair will have to be distributed to the various devices from which they will access *MISS*. In addition, updating *MISS* from more than one location incurs the risk of update collisions. Our focus on individual record operations enables us to use simple timestamp-based concurrency control. *MISS* returns a timestamp t with each *get()* and *put()* that indicates the timestamp of the current copy of the record on the *MISS* server. When updating a record, the t from the previous *get()* is included with the *put()*. If the corresponding record on the *MISS* server is newer than t an error is returned. The caller can then fetch the current record and update that as needed.[7]

[6]Alternatively a user could keep their state on a portable USB fob, access it via some networked filesystem, copy it to the local device via `scp`, etc. But, for ease of exposition and as a generally pragmatic path we assume the user utilizes *MISS* for this purpose.

4. USE CASES

We now turn to sketching several use cases for *MISS*. The first two (§ 4.1 and § 4.2) leverage *MISS* to provide fundamentally new functionality while the latter two (§ 4.3 and § 4.4) illustrate how *MISS* can help users more easily deal with current meta-information. We stress that the use cases sketched below are exemplars and, per § 4.5, we believe *MISS* has many additional uses.

4.1 Naming

Within the Internet many ad-hoc naming systems have been designed and implemented independently and generally provide application-specific names to uniquely identify a given resource. On the one hand, the ability to name resources as needed by applications in an ad-hoc manner is central to the general flexibility of the Internet architecture, and forcing all applications to use some rigid naming system would likely hinder innovation. On the other hand, the explosion of names, namespaces and artifacts of these makes the system difficult to grapple with for normal users. The two key problems with Internet names is a requirement for uniqueness and a tight coupling with a hosting provider.

At a basic level, names need to be unique such that one name cannot refer to multiple resources. This leads to names that are obtuse, ambiguous to users and difficult to share [12, 21, 3]. Consider the URL [8]. The URL may contain information about the application-layer protocol (e.g., "http", "https", "ftp", etc.), the machine that hosts some resource (either a hostname or an IP address), the transport-layer port number, the filename on the server's disk and/or arguments to some server-side program. The representation is inclusive and offers the flexibility to name a large number of resources, but at the cost of complexity. This complexity in turn makes the names hard for users to cope with directly and share. Further, global uniqueness leads to ambiguity for humans. E.g., does "ou.edu" embedded in a URL indicate the University of Oklahoma or Ohio University?[8] Since both of these institutions are referred to as "OU" in their respective regions the Internet name becomes ambiguous. In sum, while global uniqueness is required, forcing users to use such names comes with a complexity cost.

The second fundamental problem with our current namespaces is that names are tied to service providers [21, 3]. Consider an email address. While a user may get some input on creating the username portion of their email address the hostname portion to the right of the "@"-sign denotes a location within the Internet where email will be delivered. Therefore, if a user wishes to change email providers they

[7]Note: like DNS, *MISS* is a "record store" and not a database. If an application requires multiple semantically linked records, then it must implement the appropriate concurrency control (which can likely be done within *MISS* given its extensibility).

[8]The University of Oklahoma

Alice's Collection:
email *myemail* = alice@mailserver.com
rss *blog* = http://blogserver/alice-blog.xml

Bob's Collection:
pointer *Mom* = [Alice's Collection ID]
email *Mom* = Mom:myemail
URI *vacation07-pix* = http://www.flickr.com/[...]

Figure 4: Example personal namespaces.

must garner a new address and inform their correspondents to use the new version. One problem with this scheme is that changing is costly enough that users are locked into service providers rather than having the real ability to choose providers based on merit, price, etc.

Previously we have proposed a *personal namespace* system that allows users to add *aliases* on top of the Internet's myriad namespaces, hence creating a meta-naming layer that is better suited to human use [3]. While the personal namespace proposal envisioned a system for storing aliases, the *MISS* system is essentially a superset of that system. Two example personal namespaces—for Alice and her son Bob—are given in Figure 4.[9] Each record in the namespace has a type given in **bold**, a name given in *italics* and a value given in normal text which is the standard unique name for the given resource. Alice controls the names for her own resources, e.g., by assigning her current email address to "myemail". Her son can set a pointer to Alice's namespace using the context-sensitive name "Mom". He can further set a pointer to her email address as "Mom:myemail" which will resolve to Alice's current email address. Alice can change her email address—because she wishes to switch providers, gets a new job, etc.—without any impact to Bob who continues to simply use "Mom:myemail". We also note that Bob's namespace illustrates the use of *MISS* to track other users' cryptic collection identifiers, as sketched in § 3.3, in his pointer to Alice's collection.

Using *MISS* to allow users to alias their own resources addresses both fundamental problems with Internet names. I.e., by allowing users to alias unique names we allow for human-friendly, context-sensitive identifiers that are easy to use and share. Additionally, we break the coupling between names and providers by allowing people to operate on aliases that are controlled by users within their *MISS* collections.

Together with our prototype *MISS* system (see § 6) we have implemented our naming system within the Firefox web browser and the Thunderbird email client via plugins (publicly available from [1]). Once enabled this allows users to access web pages and send email using aliases stored in *MISS* that resolve to traditional URLs and email addresses.

4.2 User-Directed Protocols

We next turn our attention to using *MISS* to allow users to more directly control the flow of their transactions through the network. The naming mechanism discussed in the last section provides one aspect of this by decoupling names from service providers and by allowing users the flexibility to quickly change providers. However, such wholesale changes of providers represent only the beginning of the ways a general meta-information storage system can be leveraged to allow users to control information flow. Below we sketch two different ways to employ *MISS* records to give users more fine-grained control.

Adding Redundancy: The first use of *MISS* to allow users to direct traffic involves the user explicitly requesting redundant delivery of data. While we discuss this in the context of email, the general concept of specifying redundant delivery is a more widely applicable notion.

While the Internet's email system generally works well there are two fundamental problems in the overall mechanism. First, email has no end-to-end delivery guarantee, but rather the system is composed of a succession of SMTP servers that are each a potential point of failure. Further, there is no over-arching recovery process to detect and/or recover from delivery failures. Second, each point in the process attempts to deliver each email message even when the next hop is unavailable and this can lead to prolonged delays in message delivery. For instance, consider the case when an institution's email server loses connectivity due to a prolonged power outage. Mail destined for the institution will either be queued up in intermediate nodes and retried periodically or be sent to a backup server as defined in the DNS. Since messages at the backup server are not likely available to users (via IMAP, say) both of these situations leave messages in limbo until power is restored to the recipient's institution.

There are proposals for solving the first problem. In fact, the SMTP specifications call for "delivery status notifications" [16] and "message disposition notifications" [13]. However, the system is still largely implemented using hop-by-hop reliability instead of end-to-end reliability. That is, once a message has been handed off to the next hop, a mail server no longer tracks it in any way. Alternatively, SureMail [2] is a system whereby the sender deposits notations in a DHT that indicate a message was sent to a given recipient. Only the recipient can retrieve and understand these notations. Any email that has been noted as sent but has not been received can be requested from the sender. The second problem of prolonged delivery times has never been addressed to the best of our knowledge.

A straightforward use of *MISS* can mitigate both issues. As sketched in § 4.1 users can give their email addresses aliases within *MISS*. Rather than setting this address to a single address the user can set it to multiple email addresses separated by commas. Such a string will be accepted by the vast majority of email clients with the message being sent to both addresses. In this way a user can force the system to redundantly deliver their email[10] and hence mitigate the reliability and timeliness issues discussed above. Our prototype supports this use of *MISS*.

Hooks: Above we sketched aliasing to facilitate choice in providers and redundancy to increase robsutness and timeliness as mechanisms users can employ within *MISS* to control their transactions. In this section we specify more fine-grained control. Much like CLISP [11] provides "hooks" that are run before or after some activity to customize the process we propose providing such hooks in the Internet. *MISS* provides a natural place for users to set such hooks to inform the system about a user's preferred delivery process. We continue with email as our exemplar. An institutional SMTP server may consult a "pre-delivery" hook in the re-

[9]Note: The format used here is for illustrative purposes. As mentioned previously the information would be encoded in XML for storage within *MISS*.

[10]Note, email clients can effectively use the Message-ID email header to remove duplicate messages so the user is only presented with one copy of each message.

cipient's *MISS* collection before delivering a message to the IMAP server. In this record the user could specify a spam checking service to which the incoming message is submitted and the returned message is delivered to the user (say, after adding headers with the checker's results). This added flexibility gives users choice in terms of picking the best possible technologies rather than a bundle of technologies that is put together by a given provider. While in some sense users can do this now (e.g., by running their own spam filter) that is (*i*) often an impractical path for normal users and (*ii*) sometimes beyond the abilities of a given device (e.g., a smartphone).

In addition to email filtering, hooks could be used to specify services for web proxies that remove unwanted content (e.g., malware or ads). Alternatively a user on a smartphone could direct traffic through a transformation service that reduces picture quality, increases font size, etc. for both better viewing on a small device and better use of the scarce network resources often encountered on a 3G link (say). Another use might be a service that could observe incoming messages (email, IM, etc.), detect particularly important messages and send the user a notification to their phone (e.g., via SMS). We believe that as is the case in programming languages hooks could be a powerful concept in giving users a mechanism to flexibly control network transactions.

4.3 Sharing State

As sketched in § 1, the devices people use to access Internet services are increasing in number and variety. It would not be atypical for a user to access the Internet via multiple devices of differing capabilities in the course of a day (e.g., desktops, laptops, tablets, smartphones, etc.). One of the consequences of this access pattern is at any given time the user does not have access to all their application state. For instance, setting a bookmark in Firefox at the office does the user no good when later accessing the web via their tablet. Likewise, finding and pulling up a web page on one computer is of no use if the user only has time to read the page when they are away from that computer but have access to their smartphone. Ad-hoc solutions exist for various applications (e.g., storing bookmarks in the cloud). However, *MISS* provides a framework for addressing the problem in a coherent and comprehensive fashion. Since applications can easily push custom records into *MISS* they can readily save bits of state in the user's own collection for retrieval from another device. E.g., an RSS reader could record the article IDs associated with articles the user has read such that they do not appear as "new" when a different device presents the list of articles to the user.

We prototyped another example application on top of the *MISS* system discussed in § 6. We have built a Chrome extension and Firefox plugin to save and retrieve the state of a user's browser in *MISS* (both extensions are available at [1]). The plugin takes a snapshot of the URL associated with each window and tab open at a given time[11] and saves this in the user's *MISS* collection as a record that only the given user can access (i.e., decrypt). When the plugin is asked to retrieve the state it queries *MISS* for the given state record and opens windows and tabs as necessarily and loads the corresponding URLs. Exactly what gets stored in such records is determined by the applications. For instance, while we have not yet done so, we could store each tab's history, as well—allowing the user to use the "back" button on the restored state just as they would have on the origin browser. Again, existing solutions such as remote desktops can achieve similar functionality but *MISS* can support it as part of a foundational service.

[11]Currently the user is required to push a button to save their state, but we could trivially make the browser save the state on particular events.

While the previous two use cases of *MISS* have been aimed at utilizing the *MISS* data store to provide new functionality this use case is a bit more mundane. For a number of applications there are in fact existing ad-hoc mechanisms to share state between devices (e.g., bookmarks, address books, RSS aggregaters, etc.). This use case is therefore an illustration that *MISS* can be used for current applications, as well. However, even for existing applications there are benefits in a standard interface (e.g., we can share current browser state across Chrome and Firefox). Further, applications can leverage the *missd* to handle common tasks and therefore do not have to be nearly as complex as point solutions to accomplish the same task.

4.4 Storing Configuration

Similar to storing application state across devices and sessions, *MISS* can be used to store configuration information. Such configuration can be complex and therefore cumbersome to deal with. E.g., consider the configuration for a typical email client. For each incoming email account the user accesses the client will require the hostname of an IMAP server, the TCP port on which the server is listening, a username, a password, possibly an SSL certificate, configuration options (e.g., whether to keep mail on the server or not), etc. In addition, to handle outgoing email the client will also need the name of an SMTP relay, TCP port number and possibly credentials to access the relay. This information needs to be configured into every client a user employs to check their email (e.g., at least once per device they use and each configuration change must be manually propagated to all devices). Similar configuration information is required within many applications (e.g., instant messaging setup, HTTP proxy settings, simple general preferences in myriad tools, etc.).

MISS provides an opportunity to encode this configuration information once and then retrieve it for use on various clients. As with storing application state discussed in the last section, this use of *MISS* moves towards a more seemless integration of devices. I.e., no longer is it a manual burden to setup a new device or access information from a temporary location. In addition, once applications understand how to retrieve configuration information from a standard *MISS* record this will pave the way to remove even more manual configuration by allowing institutions to provide most of the information via a institutional collection. The users would then only have to deal with pointing their application at the institutions configuration record (using some canonical name, say) and conducting user-specific configuration (e.g., credentials). And, this latter only needs done once and then can be used across clients and devices. Such a system provides flexibility for an institution in terms of changing the configuration without requiring manual intervention by users or system administrators on each application instance.

As with the use case described in § 4.3 this application of

MISS is not a fundamentally new capability. Rather, this use case illustrates the ability of *MISS* to help uses cope with a current source of meta-information. Once a system such as *MISS* is in place many incremental benefits such as the examples we have discussed are likely to be straightforward. While not necessarily compelling in their own right, we believe the sum of such benefits will provide a qualitatively better system.

We have developed a Thunderbird plugin to retrieve email client configuration information from our *MISS* prototype (see § 6) and then use that information to configure the application. Our plugin is available from [1].

4.5 Other Uses

We stress that the use cases developed above are illustrative examples that serve to demonstrate the potential benefits of a system to manage meta-information. We envision additional uses, such as:

- Several proposals have advocated removing social network data from social networking sites like Facebook (e.g., Authenticatr [18]). *MISS* would be a natural place to store users' social graphs such that they would be available across applications.
- We have previously proposed the use of "assistants" to which small housekeeping tasks can be delegated such that a host can enter a power-saving sleep mode without losing its standing in the network [4]. Such delegation information could be conveyed within the *MISS* system in a secure fashion.
- *MISS* could be used to encode mirrors to particular data. This could be done to load balance the transferring of some large file (e.g., a Linux distribution) from multiple large servers. Or, it could be the basis of some private file trading service among a small, private group that does not possess fixed infrastructure.

There are no doubt many more examples. Our contribution is in a flexible and extensible foundation that can enable such uses—including those we cannot yet envision.

5. INCENTIVES

We briefly consider the incentives to use and deploy *MISS* for three directly impacted constituencies here.

Users: The entire design of *MISS* revolves around providing users both ease-of-use and flexibility in dealing with networked systems. As sketched in § 4 the uses of *MISS* from the user's point-of-view are numerous and therefore we believe an instantiation of *MISS* that mostly hides the system complexity will have immediate appeal.

Developers: The incentive for developers to use *MISS* in their applications is two-fold. First, there is a natural incentive to add value for users and to the extent that users benefit from *MISS* then developers will be incentivized to use the system. Second, applications naturally have meta-information management needs and so leveraging a general system rather than building and maintaining an ad-hoc mechanism is appealing. This latter may not hold as strongly for existing applications that already implement an ad-hoc mechanism, however, migrating to a general framework that allows for both removal of complexity for the application and benefit for the user is still an incentive.

Operators: Finally, system administrators and network operators—at ISPs and institutions—will be required to run *MISS* servers. *MISS* may well help their own processes. However, the largest appeal of *MISS* will be to add value for their users. There is obviously a track record for such services to be offered to users (e.g., email, web space, etc.).

None of this is meant to suggest uptake of a system like *MISS* will happen overnight. There is still a certain chicken-and-egg property to the system. Our goal in this section is to note that there are tangible incentives for the use and deployment of a *MISS*-like system.

6. EXPERIMENTS

We have prototyped the *MISS* system, including the *missd*, *MISS* server, DHT functionality and several application plugins (as sketched in § 4). Communication between the various components of the system is handled via XML-RPC[12]. We use the Bamboo DHT [7] system to connect the *MISS* servers. Our prototype implementation and the plugins discussed above are publicly available [1].

A correctly working prototype is just a beginning, however. Since storing meta-information in *MISS* results in an added lookup time during application processing we now turn to an initial evaluation of the performance of our prototype. Our experiments concentrate on three aspects of users retrieving information from *MISS*: the capacity of a stand-alone *MISS* server, end-to-end tests of the system across a small LAN testbed to better understand the non-networking costs and finally end-to-end tests across a more realistic wide-area deployment of *MISS* to understand the networking costs. While we briefly touch on record insertion below, our focus is on information retrieval as we believe that will be the most prevalent interaction with the system and information insertion will be relatively rare and can happen behind the scenes and not hinder users' activities.

We note that the workload imposed in the following experiments is in some ways contrived. However, in the absence of a production meta-information system we are not able to leverage a "typical" workload. There are aspects of such a realistic workload that will no doubt matter to performance (e.g., collection popularity, locality, etc.). However, since there is no realistic model of this new system our goal in this section is to get an initial understanding of the order of the imposed delays. In our future work we will endeavor to refine these experiments based on more realistic usage patterns as we involve actual users in our experiments.

MISS Server Load: Our first experiment aims to assess whether a reasonable server can handle a query load from reasonable-sized organization. We setup a *MISS* server running Apache 2.2.14 and used a client machine running ApacheBench 2.3[13] to stress test the server. Both machines were configured with Ubuntu 10.04 Server, each having two quad-core 2.40GHz processors with 8GB of RAM. We varied the load using ApacheBench's concurrency setting. Table 1 shows our results. An unloaded server—answering one query at a time—can handle the requests in under 1 msec. Increasing the concurrency to 500, our *MISS* server can support over 27K requests/second with an average response time of 18 msec. However, further increasing the concurrency to 3,100 yields both a longer delay—550 msec—and a lower aggregate service rate (5.6K requests/second).

Obviously in the absence of an operational deployment

[12]http://www.xmlrpc.org

[13]http://httpd.apache.org/docs/2.0/programs/ab.html

Concurrency	**Sustained Req. rate (K/sec)**	**Avg. Resp. Time (msec)**
1	1.7	< 1
500	27	18
3100	5.6	550

Table 1: Stress-testing of a MISS server.

Operation	**Median (msec)**	**95th perc. (msec)**
Parse/verify	22	26
`master` fetch	3	6
Record fetch	2	3

Table 2: Overhead of *MISS*.

we do not have estimates of a reasonable query rate. However, as a rough approximation we note that at the Lawrence Berkeley National Laboratory (LBNL) one of the main DNS servers received an average of 121 requests/second over the course of one day in July 2010 [17]. While there is not an exact equivalence between DNS lookups and our expectation of *MISS* lookups, the DNS load is suggestive of user activity levels in that web page retrievals, email transmissions, etc. trigger name lookups. Taken together the experiments and the LBNL data suggest that a single reasonable server-class host could handle the load imposed by a large organization without imposing significant additional delays to the process.

Local Network Baseline: Next we turn our attention to assessing the overhead of the basic *MISS* system we have built using a small LAN-based testbed. We deploy three *MISS* servers which also form a DHT. Each *MISS* server holds 100 collections of 100 records each. We next configure three clients to each fetch 30K random records. Each retrieval involves first obtaining, parsing and validating a `master` record followed by retrieving, parsing and validating the actual meta-information desired from the given *MISS* server. The results are in Table 2. The entire process finished in at most 35 msec for 95% of the cases. We note that the client operations of parsing and verification—not the networking operations—dominate the time required in this environment. This experiment shows that obtaining local data—which we expect to be a common occurrence due to people's general habits which tend to have a local focus—is not particularly time consuming compared with many common networking operations (e.g., loading a web page).

In the same three-node setup we also tested the naming plugin we developed for Firefox (sketched in § 4.1). Within Firefox we retrieved ten records from each of three collections hosted on the testbed. Over the 30 tests we find a median retrieval time of 31 msec or 4 msec longer than we find above when using an automated retrieval tool and not an actual application. We therefore conclude that our experiments are useful predictors of real application performance.

Internet-Wide Test: Our final set of experiments involve assessing the time required to fetch meta-information across the Internet within *MISS*. Our experiments are conservative since they do not take into account caching and prefetching as discussed previously. To understand how such mechanisms impact performance we would need a model for user activity that—in the absence of a production meta-information system—we do not have.

For this experiment we picked 100 random PlanetLab

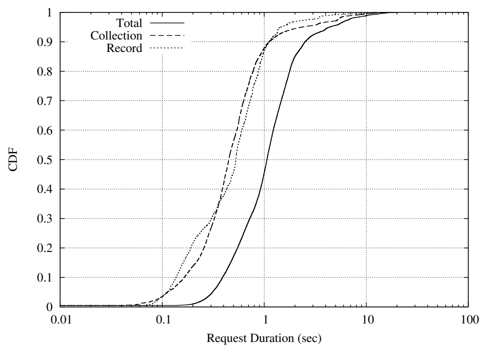

Figure 5: Duration of information retrieval from *MISS* to client at Case.

nodes to act as *MISS* servers and placed 10 collections with 100 records each on each node. The *MISS* servers form the DHT to provide `master` records. Our first set of experiments involve using client machines at ICSI and Case to retrieve meta-information. For each measurement we (*i*) choose a record at random from our corpus, (*ii*) download the associated `master` record from the DHT starting with a random DHT entry point[14] and (*iii*) fetch the given record from the *MISS* server indicated in the retrieved `master` record.

We retrieved 19K records using the ICSI client and 34K records using the Case client. Figure 5 shows the distribution of the retrieval time to the Case client. The distribution from the ICSI client is similar—with times being slightly less. The results show that the total time to fully retrieve a piece of meta-information is over 1 second in nearly 60% of the cases across both datasets. Further, 10% of the retrievals take more than 2 seconds to ICSI and more than 2.5 seconds to Case. The non-networking components of the delay are similar to that shown in the last set of experiments in absolute terms—which is much less as a fraction of the entire process for the wide-area experiments. Finally, we find that the two components of the retrieval process—fetching the `master` record and retrieving the meta-information record itself—take roughly the same amount of time (over 0.5 seconds at median). These results show that meta-information retrieval in the system we have designed consumes a non-trivial amount of time relative to the duration of normal Internet transactions. This is especially so if the meta-information retrieval is part of a task whereby the user is actively waiting on the results. The retrieval times suggest that caching and pre-fetching of commonly used records will be crucial.

Record Insertion: Finally, we assessed inserting information into *MISS* in the context of our PlanetLab experiments. We find that inserting `master` records into the DHT takes a median of 831 msec (with the 95th percentile being 9.2 seconds) and inserting meta-information records to particular *MISS* servers takes a median of 386 msec (with the 95th percentile being 1.2 seconds). The former time is of little concern because inserting `master` records is a rare ocurrence. The latter time is likely a dramatic over-estimate of reality since we expect in general users will be inserting records into close *MISS* servers and our experiment utilizes random

[14]Using a random DHT entry point is conservative and may impose added delay over a situation where the user can simply use a local *MISS* server for this purpose.

nodes throughout PlanetLab. While long, we expect these delays will largely be "behind the scenes" and therefore they will not impede users.

7. RELATED WORK

We elide an in-depth discussion of related work due to the shear breadth of topics covered in this paper. However, for each component and application of *MISS* we have discussed in this paper there is much related work. For example, our notion of indirection—as used in the naming use case—can be viewed as similar to *i*3 [22]. Further, meta-information sharing resembles a number of proposals for Internet-wide user profile or identity management (e.g.,[20, 15]). While these approaches focus on data models, *MISS* attempts to provide a neutral framework for applications to use however they see fit. In addition, our use of cryptography to name principals and encode their relationships has been previously proposed (e.g., [19]). We also simply re-use previously developed DHT technology in our current prototype, but nothing in our design is tied to a specific DHT implementation.[15] Finally, the references given in § 4 as well as others (e.g., [23, 24]), show that in some cases similar tasks have been previously developed in an ad-hoc fashion. Our contribution is not in any one of these technologies or applications, but rather in the development of a pervasive *architectural abstraction* that we believe is useful for an increasing number of networked applications and services. We believe this new abstraction holds promise to both simplify current ad-hoc mechanisms and enable new services due to reducing the burden in dealing with meta-information.

8. CONCLUSIONS

We make several contributions in this paper. First, we describe a new architectural foundation to hold users' meta-information. The system, *MISS*, accommodates arbitrary application-defined meta-information. We build the system on top of a cryptographic foundation such that access control and data integrity are built in at the root of the system. Finally, the *MISS* system allows for portability of meta-information—i.e., a user's meta-information is not tied to any particular server or provider and can be easily moved between hosting services. This gives users flexibility and choices as their situations change. We have sketched a variety of use cases for the system and also illustrate the performance of a prototype we developed. While ultimately it is difficult to try to assess an undeployed architectural framework, we believe the system sketched herein offers potential benefits and also that it continues the community's conversation on user-centric technologies. This paper represents only an initial foray into the space. Our future work on *MISS* will revolve around deploying test setups and integrating applications into the system such that we can then get the system into users' hands in a non-trivial way. This will then allow us to better understand how *MISS* will be used and any limitations that require better system design.

Acknowledgments

Fred Douglas helped us prototype several of the use cases described in § 4. Robin Sommer and Vern Paxson provided the DNS data we used for context in § 6. This work was supported in part by NSF grants CNS-0831780, CNS-0831821 and CNS-0831535. This paper was improved by suggestions from the anonymous reviewers. Our thanks to all!

[15] In fact, we interchangeably ran *MISS* on top of the Chord [10] and Bamboo [7] DHTs.

9. REFERENCES

[1] ICSI/Case User-Centric Networking Project. `http://www.icir.org/user-centric-networking/`.
[2] S. Agarwal, V. Padmanabhan, and D. A. Joseph. SureMail: Notification Overlay for Email Reliability. In *ACM HotNets*, Nov. 2005.
[3] M. Allman. Personal Namespaces. In *ACM SIGCOMM HotNets*, Nov. 2007.
[4] M. Allman, K. Christensen, B. Nordman, and V. Paxson. Enabling an Energy-Efficient Future Internet Through Selectively Connected End Systems. In *ACM HotNets*, 2007.
[5] M. Allman, C. Kreibich, V. Paxson, R. Sommer, and N. Weaver. The Strengths of Weaker Identities: Opportunistic Personas. In *USENIX Workshop on Hot Topics in Security*, Aug. 2007.
[6] J. Arkko and P. Nikander. Weak Authentication: How to Authenticate Unknown Principals without Trusted Parties. In *Security Protocols Workshop*, Apr. 2002.
[7] The Bamboo distributed hash table. http://bamboo-dht.org/.
[8] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL), Dec. 1994. RFC 1738.
[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). Technical report, World Wide Web Consortium, Oct. 2000.
[10] Chord. http://pdos.csail.mit.edu/chord/.
[11] GNU CLISP - an ANSI Common Lisp Implementation. http://www.clisp.org/.
[12] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. Persistent personal names for globally connected mobile devices. In *USENIX Symposium on Operating Systems Design and Implementation*, 2006.
[13] T. Hansen and G. Vaudreuil. Message Disposition Notification, May 2004. RFC 3798.
[14] J. Koskela, N. Weaver, A. Gurtov, and M. Allman. Securing Web Content. In *ACM CoNext Workshop on ReArchitecting the Internet (ReArch)*, Dec. 2009.
[15] Liberty alliance project. http://www.projectliberty.org/.
[16] K. Moore. Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs), Jan. 2003. RFC 3461.
[17] V. Paxson and R. Sommer. Personal Communication, July 2010.
[18] A. Ramachandran and N. Feamster. Authenticated Out-of-Band Communication over Social Links. In *ACM SIGCOMM Workshop on Online Social Networks*, 2008.
[19] R. Rivest and B. Lampson. SDSI—A Simple Distributed Security Infrastructure, 1996.
[20] A. Sahuguet, R. Hull, D. F. Lieuwen, and M. Xiong. Enter once, share everywhere: User profile management in converged networks. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.
[21] S. Singh, S. Shenker, and G. Varghese. Service Portability: Why http redirect is the model for the future. In *ACM SIGCOMM HotNets*, Nov. 2006.
[22] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *IEEE/ACM Transactions on Networking*, volume 12, Apr. 2004.
[23] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: better privacy for social networks. In *CoNEXT*, 2009.
[24] D. N. Tran, F. Chiang, and J. Li. Friendstore: cooperative online backup using trusted nodes. In *Proc. of the 1^{st} Workshop on Social Network Systems*, 2008.