Personal Namespaces

Mark Allman
International Computer Science Institute

Abstract

In this paper we propose an over-arching namespace that serves to abstract away the Internet's current and obscure naming schemes from *users*. We argue for users to have *personal namespaces* that are not concerned with unique naming of resources, but rather focused on aiding user's interactions with the system. This additional namespace does not replace any of our current (or future) naming systems. Rather, our vision calls for adding a naming layer that provides the ability for users to meaningfully alias network resources (especially their own). These aliases become context-sensitive, provider independent names for objects that can be easily shared among people. In addition, we sketch a strawman system—called *pnames*—in high level terms as a starting point in the discussion of how such a system might be built.

1 Introduction

Currently, Internet applications use a significant number of systems to name resources—from the DNS to URLs to email addresses to a multitude of other application layer naming conventions. These naming systems have been designed independently as needs arise and generally provide application-specific context-insensitive methods to uniquely identify a given resource. Therefore, current names are largely dictated by where a given resource is located and how it is provided. For instance, consider the URL [3] for a web resource. The URL may contain information about the application-layer protocol (e.g., "http", "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 URL representation is inclusive and offers the flexibility to name a large number of resources at the cost of complexity. Our current naming schemes have a number of drawbacks, such as:

Names are obtuse and "ambiguous". URLs, email addresses, etc. can be quite obscure to users and do not lend themselves to manual use. Additionally, many of our current namespaces require the names users interact with to be globally unique and this can lead to ambiguities *for people*. E.g., is "ou.edu" the domain for Ohio University or the University of Oklahoma? The latter—even though in different parts of the United States and to different groups of people both are referred to as "OU". While, at some level, uniqueness is obviously a requirement, we argue that *people* should not be forced to interact with the system in a context-insensitive manner. Some applications assist users trying to grapple with obscure naming for common tasks. For instance, web browsers have bookmark facilities and email programs have address books such that resource names are abstracted from users, who are simply presented with context-sensitive aliases (e.g., "Dad", "Bob's blog", "Alice's budgets", etc.).

Names are hard to share. Similar to the last point, obscure resource names are difficult for users to share with others. For instance, most people will likely not be able to rattle off the URL of the flickr.com web page containing their summer vacation pictures to a friend at a chance meeting in the grocery store. Applications have again employed some features that help with some common tasks. For example, web browsers often allow users to easily email links to others. In turn, email readers can feed URLs in email messages to web browsers. While these sorts of processes remove the need for users to directly deal with some obtuse resource names in some contexts, the techniques are somewhat clunky, not generic and do not help in the grocery store[1].

In addition, users often have occasions to share resource names with themselves. For instance, sharing a collection of web page bookmarks between a desktop machine and a laptop is not generally straightforward even though there have been attempts to make the task a bit easier (e.g., sharing bookmarks via RSS). There are also ways to synchronize address books across hosts (say a desktop and a PDA). However, there is no standard way to do this and any particular instance of this sort of sharing works only because some specific system has been put into place for the devices in question. While useful, such schemes are far from a coherent way to arbitrarily share high-level names.

[1] Actually, these schemes do indirectly help in the grocery store because what inevitably happens is for the party with the pictures to promise to email a URL to the other person when they get home. While this *does* help users a bit it is not as convenient as being able to directly provide the name of a resource in the grocery store while both people are thinking about it.

Names are intolerant of location change. As noted above, most current naming includes some notion of where a resource is currently located. For instance, an email address explicitly includes the location of a mail server (or, at least a forwarder). Therefore, when a person graduates, changes jobs or just switches ISPs they need to distribute a new name to their social network such that they can be reached. There are ad-hoc mitigations such as using institution-independent email systems (e.g., GMail) or forwarding (e.g., acm.org), however such per-application mechanisms are not available across all current namespaces and may not be ideal for many use cases (e.g., because of economic or privacy concerns).

Naming is a one-way street. There are generally three parties involved in accessing resources: the *consumer* who is accessing the information, the *content provider* who provides the information being sought by the consumer and the *service provider* who runs the servers on which the content resides. Namespaces generally include portions specified by both the service provider and the content provider. For instance, a content provider may name a file while a service provider will generally have named the server, with both pieces being required to obtain the given resource. Consumers can assign arbitrary local aliases to a given resource. However, when the actual unique name of the resource changes the user of the local alias will have to track down the real name again and update the alias. Finally, neither content providers nor service providers can provide the context-sensitive naming that the consumer can impose (e.g., "Mom's web page").

This paper sketches a higher level notion of naming that provides for a *personal namespace*. The vision for this new namespace, *pnames*, is that it is a *user-centric* meta-namespace that can provide an abstraction to the myriad of other namespaces in current and future networks. The goal of *pnames* is not to replace any current namespace, but rather to add an abstraction to make interacting with these namespaces easier for users. *Pnames* is not application specific and cannot itself be used to access a resource. Rather, the names will simply be aliases to resource's current unique identifiers (or, those created in the future).

Each personal namespace is in some sense a "root" namespace. That is, some given name "*foo*" can have a different meaning inside each namespace. The names in a personal namespace can be selectively shared across hosts and users. Individuals can manage any number of namespaces for themselves. For instance, one could use different namespaces for home and work (e.g., to have context-sensitive names like "calendar" bring up the appropriate resource in a given setting). Further, while we use the name "personal" as a touchstone for the notion that these namespaces are user-centric, such namespaces could also be used for a group. For instance, a institution may wish to run their own namespace that names internal resources.

This paper describes a vision for a possible system that enables *user-centric* naming for network resources. Our goal for this paper is to start a discussion about whether an over-arching naming abstraction that is focused on users is desirable, as well as getting feedback on the initial system design we present.

The remainder of this paper is organized as follows. § 2 discusses related work. § 3 outlines the proposed system and discusses several design decisions. § 4 comments on using the *pnames* system for more than small-scale personal naming. Finally, we conclude in § 5.

2 Related Work

The Internet and its applications utilize a large variety of naming schemes. A number of proposals share ideas with the system sketched in this paper. This paper is not breaking new technical ground, but trying to develop an architectural vision for an abstraction that encompasses many ideas from previous work. The following is a discussion of the various classes of related work.

Personal, Application-specific Naming. Many applications provide ways for users to create aliases or shortcuts to often-used names. Examples include email address books and desktop icons that provide ready access to a shared disk. These facilities unquestionably aid users in their interactions with the network (and are suggestive of a need for user-centric names). However, many of the drawbacks given in § 1 remain.

Private Naming Realms. A number of high-level user-oriented naming systems have been developed on a small scale. For instance, administrators can create aliases for hosts in the /etc/hosts file on most Unix machines. Anyone using the given machine can then take advantage of the alias. America OnLine (AOL) (among others) defines keywords to help users navigate their system. These keywords are simply translated into the real name of the given resource. These naming schemes have utility for users. However, the scope in which these names are valid is limited and these schemes also lack the flexibility for *users* to name resources themselves.

Naming Hosts. Much work has gone into naming hosts. We discuss several classes of mechanisms. First, the Domain Name System (DNS) [6] provides the ability to map human-understandable names into an IP address. Second, [4] outlines the notion of user-centric naming of hosts, noting that users want to interact with hosts on their own context-sensitive terms (e.g., to get a picture from "Alice's phone"). [4] describes a way to introduce and track devices by gossiping unique host identifiers and storing these as local user-defined names such that there is no centralized database. Lastly, we note that the Semantic Free Referencing (SFR) scheme [9] abstracts out the hostname and port number portion of URLs, replacing them

with an SFRTag (which points to IP and port information). The SFRTag provides a level of indirection that makes the tag independent of the host and port on which a resource actually resides and therefore can be updated as services change network locations. These host-centric schemes address some of the pitfalls discussed in § 1, but others remain because the schemes only focus on a portion of the name required to access a resource.

Naming Services. The DNS has also been used—both formally and informally—to name services. DNS SRV records can be used to lookup a service in addition to IP addresses. More informally, hostnames are often defined to have meaning to users (e.g., "maps.google.com"). However, often resources do not have concise names that can be readily inserted into the DNS. This is especially true of resources normal users make available. [8] introduces *permafind* to address the issue of service portability by relying on protocol redirection, indirection and relaying. The *permafind* system adds an indirection point to the network. For instance, HTTP redirects are used to allow a user-defined *permafind* name to point to an arbitrary web location. Also, the *permafind* system could simply forward email to the proper service provider. This allows users to define and change the names and actual locations of various resources without changing the globally used name to reach that service. The *permafind* system well addresses many of the problems we outline in § 1. However, the system relies on relaying through a central service or protocols that support redirection. Further, to address the naming problem *permafind* must also deal with data transactions, bringing up issues of scalability and privacy (since *permafind* would be offer a possibly large-scale view of user behavior). On the other hand, *pnames* is decentralized and deals only with the naming aspect of network activity (see § 3). The *pnames* proposal in this paper is similar in many ways to the spirit of *permafind* (and, in fact, [8] hints at a system more like *pnames*). However, we have designed *pnames* without regard to how current protocols work and/or how they would need to be designed in the future.

Generic Group Naming Schemes. SDSI [7] offers a mechanism to generically name principles, identify those principles by their key fingerprint (as we do with *pnames*; see § 3) and allow principles to name other principles in higher-level terms (e.g., "Alice's friend Bob"). In generic terms the SDSI techniques are what we use in our proposed framework. In this paper, however, we are focused not on generic naming of principles, but on adding an *abstraction* to the *network architecture* that is focused on users being able to easily name network resources.

Summary. The previous work described above (and many others) has led us to a different architectural approach. Rather than focusing on particular aspects of various namespaces in use, we take the view that we need an *additional naming layer* that is focused on meeting the needs of *users*.

3 System Overview

The *pnames* system is conceptually simple: a daemon process runs on every host and tracks various *pnames* namespaces on their user's behalf. Each namespace is identified by a *pnames* namespace identifier (NID) which is a cryptographically unique identifier derived from a hash of the public key of a locally generated public/private key pair for the given namespace. Basing the system on keys instead of names handed out by some central authority also keeps the system decoupled from any required infrastructure or provider. This approach is similar to that used in [4]. The NID will be difficult for people to use directly. This clearly runs contrary to the entire goal of the *pnames* system. However, the *pnames* system itself can be used to provide a human-readable alias that users can employ when using someone's NID (as discussed below). The following sub-sections explore various aspects of the *pnames* architecture.

3.1 Contents

We envision three types of names to be accommodated in the *pnames* system.

Simple Names. These entry types are the backbone of the *pnames* system and will be simple aliases for network resources. For instance, there would be a type for mapping hostnames to IP addresses or adding aliases to existing hostnames. This would allow for naming private address space or providing meaningful names (e.g., "laptop") for machines that have obscure names (e.g., "laptop-dept43-building2-3421324.foo.bar.com"). In addition, names could be added for URLs, allowing names like "calendar" to point to the appropriate resource for a given person or group. The intent is for the system to be extensible to allow aliases for arbitrary existing and future namespaces (e.g., names that map to NFS or SMB shared directories, port numbers, DHT keys, email addresses, etc.).

Pointers. A namespace can include pointers to other namespaces for ease of reference. For instance, Alice may have a pointer to Bob's namespace such that she can easily access available items in Bob's namespace (e.g., by using something like "Bob:calendar"). Not only is the naming scheme easy for people to understand, but the names associated with a particular namespace (e.g., "Bob") are context-sensitive within a particular namespace (e.g., Alice's, in this case). Given that NIDs are not human friendly (as discussed above) pointers will be *crucial* to accessing other's namespaces.

Fallbacks. A namespace can define one or more "fallbacks" for a particular entry type. As an example, consider hostname lookup. If the given hostname is not found in the user's namespace the system may simply request the name from the DNS. This is akin to the normal procedure that hosts use to consult a local hostname database before querying the DNS.

3.2 Example

Before discussing additional aspects of the system we pause to discuss an example of two *pnames* namespaces to aid the reader's intuition behind the system. Figure 1 shows namespaces for Alice and her son Bob. Alice's namespace defines two names for resources she makes available. The first name is "email", is of type "email" and points at Alice's email address ("alice@mailserver.com"). The second name is of type "rss", has the name "blog" and points at the URL of Alice's blog.

namespace: *[Alice's NID]*
email: *email = alice@mailserver.com*
rss: *blog = http://blogserver/alice-blog.xml*

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

Figure 1: Example namespaces.

The entry for Alice's son Bob is shown next and illustrates the power of the *pnames*. The first entry in Bob's namespace is a pointer that binds Alice's namespace to the name "Mom" within Bob's namespace. Using this pointer, Bob can access names in Alice's namespace without dealing with Alice's obscure NID. The second entry shows this by defining a name of type "email" and name "Mom" to point at "Mom:email"—which in turn resolves to Alice's email address. This shows that Bob can now deal with the user-friendly and context sensitive "Mom" when writing a note to Alice, instead of Alice's actual email address. In addition, if Alice changes her email address she can update her "email" name and Bob will not be required to change anything. Finally, the first two entries of Bob's namespace show that the names in a namespace are sensitive to the type of name they are defining and so can be defined for each type. I.e., "Mom" is both a pointer to Alice's namespace and the name of Alice's email address depending on the context in which the name is used. The third line in Bob's namespace sets a name for his vacation pictures. Once set, Bob can inform friends about these pictures using his "vacation07-pix" name.

We note that Bob could add an email address for his Mom such as "email: Mom = alice@mailserver.com". In the case where Alice does not make a namespace available this is what Bob would no doubt do (to be able to use the name across his various computer's, say). However, when Alice makes a namespace available it is more advantageous to put links to her namespace such that when Alice changes her email address (and, hence her namespace entry) Bob does not have to update his namespace. In other words, the *pnames* system works best when the *content providers* name their resources.

3.3 Bootstrapping

One problem with using NIDs based on cryptographic keys is that they are in fact more obtuse than any current naming scheme and so at odds with the very problem the *pnames* system is trying to attack! There are several ways we might cope with this deficiency.

First, we note that determining and exchanging NIDs is a *one-time cost* between two users[2]. Per the example in the previous subsection, after finding a NID, a pointer can be inserted such that the namespace no longer needs to be identified by the real (and obtuse) unique NID. Rather, the namespace can be referred to by something context-sensitive such as "Bob" or "Mom" making the use of names within the namespace straightforward and easily understood by people (after the initial pointer insertion).

In addition, well-known namespace registries could be setup that would use pointer records to associate people's personal namespaces with an easy to communicate name. The *pnames* daemon could know about popular registries such that they could query the "MasterRegistry1" (for instance) namespace for "MarkAllman" and receive the NID for the author's namespace. The user could then record the NID of the namespace as a pointer using whatever context-sensitive name the user chooses (e.g., "Mark", "mallman", "King", "dork", etc.).

Another bootstrapping technique is to glom on to traditional ways users convey naming information. A NID could be included in an IM profile, a vCard or a header or signature in an email message. Savvy applications could interface with the *pnames* daemon to allow users to easily record pointers from such entries with a simple mouse click (or, more speculatively entries could somehow inserted into a namespace automatically if they are carried in some standard way like a known email header).

3.4 Sharing

Sharing *pnames* is key component of and motivator for the system. Local storage of name mappings can clearly be done in arbitrary ways at the discretion of the local

[2]To carry our grocery store example from § 1 further, we see that if two people meet in the grocery and have previously exchanged NIDs then sharing names is straightforward. However, if they have not shared NIDs previously then we have likely made the problem more difficult by introducing quite obtuse NIDs into the mix. Our model is that people will develop lists of NIDs for their social network such that adding new NIDs to their namespace will be far more rare than using the names within these namespaces.

system (e.g., using some database system). When sharing names a standardized form will be needed (e.g., an XML schema). The system will then accommodate two ways to share names: (i) resolution from a centralized database or (ii) direct exchange with another individual.

The normal way to share *pnames* will be through a distributed hash table (DHT) that would be built from dedicated infrastructure-level nodes (similarly to the DNS system). While the DNS system functions reasonably well, such an arrangement would likely not work as well for *pnames*. First, hostnames follow a natural hierarchy and so a tree of servers can be constructed to match this hierarchy. *Pnames*, on the other hand, is a mostly flat naming system with many namespaces, but no depth in each and therefore lends itself to the flat naming provided by DHTs. Second, the aggregate load the *pnames* system may impose on a central service will likely far out-pace the load placed on the DNS. The number of *resources* on the Internet dwarfs the number of hostnames. Further, a single network resource may be named by a large number of users in the *pnames* systems, thereby increasing the aggregate storage and retrieval costs.

Pnames records could be maintained and requested at will by users. For instance, the data item requested from the DHT could be stored and retrieved using a hash key of a SHA-1 of the name being looked up, the name's type and the NID that contains the name. *Pnames* inserted into the global database could be public or private. All records will be signed by the owner of the given namespace. When looking up a record the resolution process should include validating that the name is properly signed by the owner of the corresponding namespace. Private records would additionally be encrypted such that only authenticated users could decrypt and use the given names. Since the system requires a public/private key-pair to setup each namespace all the infrastructure required to sign, verify, encrypt and decrypt records will be required to be in place.

The above scheme works on a per-name basis not a per-namespace basis. This has two main implications. First, there is no easy way to list out all the names in a given namespace[3]. Second, just because Alice has access to some name within Bob's namespace that does not imply that she has access to Bob's entire namespace. Alice could fish for names within Bob's namespace quite easily, however, and see where any found names point (if they are public) or at least that they exist if they are not public and Alice does not have access to the given record. We offer two ways to address this issue (although, there are no doubt others):

- One way to combat the sort of fishing given above is for private records to include some shared secret in addition to the name, NID and type used to construct the key used to access a particular name. In this way, arbitrary queries for private names would not be possible. The cost of this approach is in terms of additional mechanism to establish shared secrets between users.

- A second way to combat the sort of fishing described above is to make the DHT return a bogus record for every non-existent name accessed that looks like an encrypted record that the given user does not have access to decrypt. The DHT then adds this bogus entry to the DHT so that the same bogus entry is returned to every query for the given name (until the entry expires or is replaced by an actual entry by the user). In this case, it will appear that every namespace has every possible name, which makes the fishing scheme moot. While this scheme requires no effort or interaction among users it requires additional work from the DHT.

A second way records could be shared would be outside the DHT via whatever method users employ to exchange data. For instance, records could be attached to email messages, sent via instant messaging sessions or swapped via USB sticks. One particularly interesting way to share names is for using WiFi or Bluetooth on user's mobile devices when the user's are in close proximity (e.g., when meeting in the grocery store), as sketched in [4]. Once obtained the user could import the records into the local *pnames* system. This scheme for exchanging *pnames* may suit some circumstances, but does not lend itself to updating names or accessing new names as readily as sharing through a DHT since accessing the global database will be well supported by the system as opposed to ad-hoc importing from arbitrary sources which will likely be a manual process.

3.5 Reliability

The added layer of indirection provided by *pnames* introduces a new failure case when accessing services. Therefore, the system needs to be designed to be robust. Our design uses two primary methods to increase robustness. First, sharing is done through a DHT, which is a robust and distributed data structure that gracefully deals with server failure, connectivity issues, etc.

A second robustness mechanism is the heavy use of caching. Each record in *pnames* will be tagged with a time-to-live (TTL) field, which represents the valid lifetime of the given name. We expect most names will be slowly changing and hence TTLs will be on the order of days. When names time out the local *pnames* system is encouraged to simply lookup the name again such that it can be cached with a new lifetime. If the name is not actually used for a long period of time the name may be

[3]Directories of names could be created for various purposes, of course.

actually removed from the local cache. This caching and pre-fetching of names is similar in spirit the notions in [5, 2] for massively distributing copies of the DNS NS records and using stale cache entries to aid robustness. Somewhat permanently caching *pnames* should not be a burdensome requirement since hosts already track bookmarks, email addresses, etc. for the most used resources. Simply using the local *pnames* daemon to cache a few thousand non-local entries and keeping them up-to-date is not a resource intensive task on modern computers. This local caching will both increase the robustness to lookup failures for common resources, and also reduce the performance penalty that another level of indirection naturally adds to the overall process of accessing some resource. That said, caching would not be a required component of the system (and there may be good reasons not to cache, for instance when using a constrained mobile device).

3.6 Security

The *pnames* system is designed with security in mind along three axes:

Validating records. Since namespaces are setup and named with a public/private key-pair all records inserted can be easily signed and then validated as belonging to the given namespace as they are looked up and used.

Sharing records. Records can be privately shared by simply encrypting the record such that only authorized users can access it. The entire system forces the use of keys and so encrypting records for only a given set of people is straightforward[4]. Further, records that are to be shared with nobody else can be privately held in the local name database and not inserted in any fashion into the global database.

Robustness to attack. A final possibility that we deal with up-front is attack on the infrastructure required to share *pnames*. A number of proposals have been put forth to help the DNS system in the case of targeted attacks against key DNS servers (e.g., [5, 2]). Since the *pnames* database is built on a DHT it is naturally distributed and less vulnerable to targeted denial-of-service attacks than a centralized service like a particular DNS server or a web server. Further, the aggressive use of caching sketched in § 3.5 reduces the reliance on the central infrastructure for common lookups. On the other hand, DHTs may well be vulnerable to different kinds of attacks that involve leveraging homogeneous software on tightly coupled hosts.

3.7 User Interface

As discussed at the beginning of this section, the *pnames* system requires a daemon to run on every host to resolve *pnames*. As we have sketched, this daemon is responsible for providing names to applications, fetching and caching names from the global database, pushing local names into the database, etc. We envision the user's interface into the system to come in two forms. First, the daemon would provide an interface for arbitrary applications to lookup, add, delete and change names. We note that all applications that wish to utilize *pnames* will have to be changed. This seems like a steep price, but the intent is for the benefit to outweigh the extra cost over the long-term. An additional factor to keep in mind is that since we are not *replacing* a naming scheme, but *adding* a meta-namespace, applications can naturally evolve to support *pnames*. A second way users could interact with *pnames* would be through a dedicated tool that would let them manage their namespace much like they manage an address book.

3.8 Situational Names

An additional content type that *pnames* could support is "situational" names, or names that depend on where a given host (and, hence the resolver) is connecting to the network. For instance, when a laptop connects to the network behind a NAT accessing some host's SSH server might require sending traffic to port 22 on 192.168.1.76, whereas outside the NAT the same server might be accessed via IP address 69.222.45.91 and port 2222—which represents the external address of the NAT and a port number that has been configured to be forwarded to the given host's sshd. Given the prevalence of such addressing realms it may be useful to incorporate the *location of the information consumer* into the resolution process. Various simple primitives could be used to allow users to configure their situational names. For instance, the *pnames* daemon could use the next hop router's IP or MAC address, the SSID of the access point, etc. as input to the resolution process. More speculatively a more full-featured programming environment could be provided such that plug-ins could be written to aid name resolution.

4 A Different Namespace

One natural consequence of adding a meta-namespace to the Internet's architecture is that DNS names could become less important in a non-technical way. With *pnames* anyone could provide a "top-level domain" by simply creating and populating a namespace. For instance, one could envision an organization like Major League Baseball providing a namespace that would allow for easy access to each baseball team's resources. Or, a television domain providing links to various stations and networks. Anyone could create these namespaces, populate them as they see fit and then allow users the choice of using them or not. Of course, this could create inconsistency because namespace A may resolve some name to resource X, whereas a different and competing namespace B may

[4]Note, while the system requires users to possess keys it does not require them to understand or interact with those keys. See [1] for a discussion of using opportunistically generated keys for such purposes

resolve the same name to resource Y. This leaves users in a quandary about which name to trust—if either of them. This is similar to registering the DNS name $Z.net$ in the hopes of getting traffic meant for $Z.com$, however on a much broader scale, since the number of top-level domains is unbounded and uncontrolled. This problem is fundamental and is caused by the desire to have labels that are both human-friendly and globally unique—-which inevitably leads to name clashes.

5 Summary and Future Work

We offer two main contributions in this paper. First, we outline a new naming abstraction that is *user-oriented*. This naming layer has nothing to do with the nitty-gritty protocol details of accessing a resource, but rather provides a service that will help *users* deal with the myriad of naming systems on the Internet and the obscurity of globally unique names. We think that, apart from the actual system we sketch, this abstraction is useful as we strive to further evolve networks. In addition, we sketch a strawman design for such a naming system. While we have not yet worked out a detailed specification, we have attempted to consider the high-level aspects of such a system.

There is much future work to be done to realize a working personal namespace system. There are many details to consider and refinements to make before even an initial system would be operational. One of our goals in writing this paper is to engage the community in a discussion of the architecture and proposed system.

Acknowledgments

This paper benefits from useful discussions with Ethan Blanton, Vern Paxson and Scott Shenker, in addition to comments from the anonymous reviewers. This work was funded in part by NSF grants ITR/ANI-0205519 and NSF-0626539. My thanks to all!

References

[1] 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.

[2] H. Ballani and P. Francis. A Simple Approach to DNS DoS Mitigation. In *ACM SIGCOMM HotNets*, Nov. 2006.

[3] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL), Dec. 1994. RFC 1738.

[4] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. User-Relative Names for Globally Connected Personal Devices. In 5^{th} *International Workshop on Peer-to-Peer Systems*, Feb. 2006.

[5] M. Handley and A. Greenhalgh. The Case for Pushing DNS. In *ACM SIGCOMM HotNets*, Nov. 2005.

[6] P. Mockapetris. Domain Names - Implementation and Specification, Nov. 1987. RFC 1035.

[7] R. Rivest and B. Lampson. SDSI—A Simple Distributed Security Infrastructure, 1996.

[8] S. Singh, S. Shenker, and G. Varghese. Service Portability: Why http redirect is the model for the future. In *ACM SIGCOMM HotNets*, Nov. 2006.

[9] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *Usenix/ACM Symposium on Networked Systems Design and Implementation*, Mar. 2004.