User-Relative Names for Globally Connected Personal Devices

Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas,
Sean Rhea, Frans Kaashoek, and Robert Morris
Massachusetts Institute of Technology

1. INTRODUCTION

Personal devices such as mobile phones, digital music players, personal digital assistants, console gaming systems, and digital cameras are now ubiquitous in the lives of ordinary people. As these devices proliferate, peer-to-peer connectivity between them is increasingly important. For example, a user may copy photos from a camera to a PC for storage, to a web page for publishing, or to a photo iPod to take on the road, and perhaps from there to a friend's iPod. One current transfer mechanism—plugging devices together via USB cable—is both straightforward and secure: the cable itself physically indicates which devices should participate in the transfer, and the isolated physical medium guarantees its security.

As personal devices begin to support wireless networking and Internet connectivity, we would like to extend the simplicity and security of a USB cable to device connectivity on a global scale. Alice should be able to connect her WiFi-enabled iPod to her home PC via a "virtual cable," so that she can browse photos or play music stored there from a WiFi-enabled coffee shop or friend's house. Setting up this "virtual cable" should not require technical knowledge or special configuration on Alice's part, and it should continue working even when the devices it connects are behind firewalls or NATs.

If Alice meets Bob in a coffee shop, she should easily be able to share with him information or services located on any of her personal devices. Bob should be able to connect to Alice's devices even after he leaves the coffee shop, until she chooses to sever their relationship. No one else should be able to impersonate Bob, however, in order to gain access to Alice's shared resources.

The *User Information Architecture*, or UIA, is a peer-to-peer connectivity architecture that provides users a simple, intuitive, and secure way to share information and services between personal devices by assigning ad hoc names that act like "virtual cables." Users assign names by "introducing" devices to each other on a common network. Unlike the ephemeral names used in rendezvous schemes such as Apple Bonjour [1], however, UIA names persist and remain securely bound to the global cryptographic identities of their targets [11,12,16] as devices migrate. Once Alice introduces her iPod to her home PC, her iPod can continue accessing her PC by the same name from anywhere she finds Internet access.

In a network of billions of users, globally unique names would inevitably have to look something like `ipod.alicesm5186.myisp.com`, substantially limiting their conciseness and readability. UIA names are instead *user-relative*: users control their own private namespaces much as they control their mobile phones' address books today. Unlike a conventional address book, however, a UIA namespace is shared across all the devices a user owns: changes made on one device automatically propagate to the others.

Users assign user-relative UIA names not only to their own devices but also to other users. Bob might know Alice as "Alice", her company directory might list her as "Alice Smith, Marketing", and her son might simply name her "Mom". If Alice gives Bob access to some files on her PC, he accesses them via a name analogous to "Alice's PC". In this way, UIA adapts peer-to-peer social networking ideas previously explored for other purposes [3,10,15] to form a secure peer-to-peer naming infrastructure.

The next section presents the goals of UIA's naming system, and Section 3 describes its operation from a non-technical user's viewpoint. Section 4 develops the technical details of UIA's design, and Section 5 summarizes implementation status. Section 6 presents related work, and Section 7 concludes.

2. GOALS OF UIA

The purpose of UIA is to provide users with a convenient and intuitive way to name and communicate with their own and their friends' personal devices. To this end, UIA must satisfy the following goals:

- Names must be *user-relative* so as not to require global uniqueness. If Alice owns only one laptop and has only one friend named Bob, she should be able to refer to them as `laptop` and `bob`, despite the millions of other laptops and people named Bob in the world.
- Names must have *strong bindings* to the identities of the objects named, independent of their current physical location or network attachment point. When Alice refers to her name `laptop`, the name should always resolve to *her* laptop or fail to resolve (e.g., if it is turned off); no other device should be able to impersonate it.
- Assigning names must be a *simple and intuitive* process. If Alice meets Bob at a conference and their laptops share a common WiFi network, assigning a name to Bob should be as simple as looking him up in a local network browser window and clicking "Bookmark".

- A user should only have to manage *one namespace*. If Alice already owns several devices, she should only have to name a newly purchased device once, not once on each existing device.
- Users should easily be able to *share* their personal namespaces and device resources selectively with trusted friends and acquaintances. If Alice gives Bob permission to access some files on her desktop PC, he should have access to them via a name as simple as “Alice’s PC”.
- When physically possible, UIA should *automatically provide connectivity* among devices that have naming relationships, including between Internet-connected private LANs and within ad hoc networks disconnected from the Internet (e.g., among passengers in an airplane).
- Finally, UIA should *coexist cleanly* with DNS, so that a user can use personal names like `laptop` alongside global names like `amazon.com` within the same application.

3. USER EXPERIENCE

This section describes UIA’s key operating principles from the perspective of a non-technical user, demonstrating how it satisfies the goals listed above with the help of an example scenario illustrated in Figure 1. Technical details of how the system provides this user experience follow in the next section.

3.1 Local Introduction, Remote Access

Each UIA device ideally ships from its manufacturer UIA-enabled and pre-configured with a generic name for itself such as `laptop` or `phone` that the user can accept or change as desired. After purchase, the device acquires names for other devices as its user *introduces* it to them on a local-area network. These introductions assign persistent names by which one device can securely refer to the other, and these names subsequently facilitate remote access as the devices migrate to different networks.

Users can introduce UIA devices in two different ways: they can *merge* two or more of their own devices to form a *personal device cluster* sharing a common logical namespace, and they can create named *social links* from their own clusters to other users’ personal users.

3.2 Personal Device Clusters

At Time 1 in the scenario shown in Figure 1, Bob purchases and brings home a new laptop and Internet phone, having default names `laptop` and `phone` respectively. At Time 2 Bob uses UIA’s local rendezvous tool on each device, similar to those already available with Bonjour [1], to find the other device on his home WiFi network and selects the UIA “Merge Devices” command on each. This action merges the two devices into a single personal device cluster, allowing names already known to or subsequently entered on each device to be used on the other device as well.

While working on his laptop, for example, Bob can now refer to his phone by its manufacturer-configured name, `phone`. Furthermore, the merge operation is persistent and secure; when Bob subsequently takes his

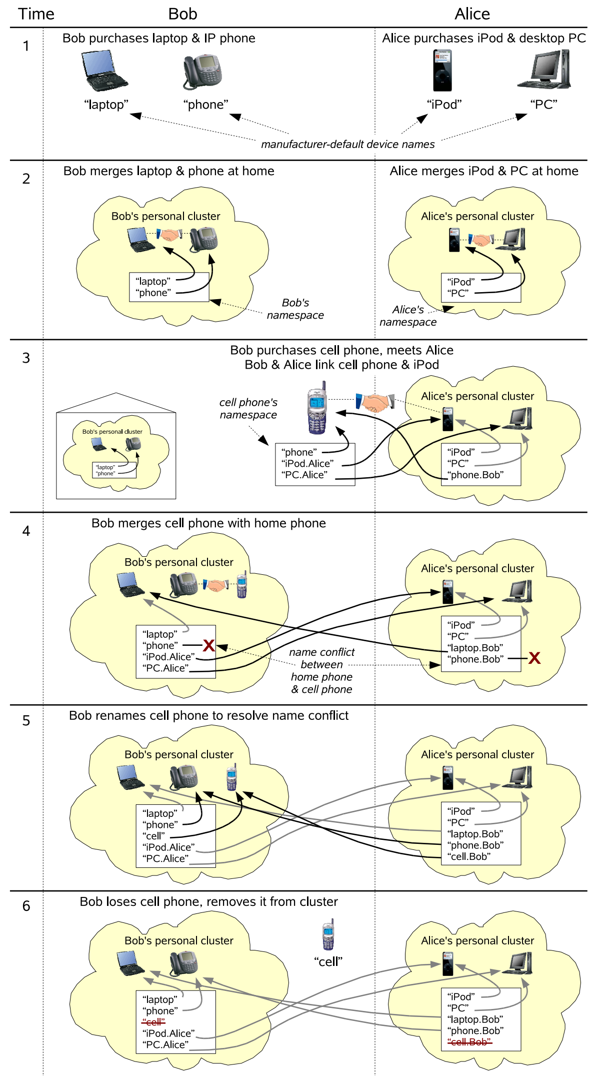

Figure 1: Example Personal Device Scenario

laptop on a trip, he can remotely access his home phone from his laptop anywhere he finds Internet access, to check his voice messages for example, still using the same name, `phone`.

3.3 Name Sharing and Social Networking

Users can also create named links from their personal device clusters to those of other users in order to reflect social relationships and facilitate information sharing. With this second form of introduction, users retain exclusive control over their clusters’ private namespaces, but can selectively grant access to known acquaintances, without making their devices vulnerable to unknown hosts that may inhabit the same physical network.

In the example scenario, Bob purchases a new WiFi-enabled cell phone at Time 3 and meets Alice at a coffee shop on his way home, before he has merged his cell phone with his other devices. Bob finds Alice’s iPod using his cell phone’s local rendezvous tool and selects UIA’s “Link to Another User’s Device” command,

and Alice does likewise. Bob's phone presents Alice's self-chosen user name if available, such as `Alice`, as a suggestion for Bob's new name for Alice, but Bob can override this hint as desired—using an alternative like `Alice Smith` or `Alice from IPTPS` for example if Bob already knows other Alices. If Alice now grants Bob access to some files on her desktop PC at home (which is named `PC` in Alice's personal namespace), then Bob can subsequently access these files from his cell phone under the "Bob-relative" name `PC.Alice`.

Bob then returns home and merges his new cell phone with his home phone, as shown at Time 4 in Figure 1. Bob's laptop transitively learns about the cell phone's membership in the cluster without him having to merge them explicitly, and Alice can now name Bob's laptop on her devices via the Alice-relative name `laptop.Bob`.

3.4 Resolving Conflicts

Unfortunately, both of Bob's phones happened to have identical default names of `phone`, resulting in their names conflicting in his newly merged namespace. UIA notifies Bob of the conflict, and he can continue using the non-conflicting name `laptop` but must resolve the conflict before the name `phone` will work again. Bob resolves the conflict on his cell phone at Time 5, by renaming it `cell` while leaving the home phone with the name `phone`. Bob's other devices learn the resolved name bindings automatically, and Alice's devices now see Bob's phones as `phone.Bob` and `cell.Bob`.

If Bob makes conflicting namespace changes on two of his devices while they are disconnected from each other, UIA similarly detects the conflict once the devices reconnect. Bob can continue using other names while conflicts exist, and he can resolve such conflicts at leisure on any of his devices.

3.5 Removing Devices

When Bob loses his cell phone at Time 6, he uses his laptop to remove the cell phone from his personal cluster. The laptop notifies Bob's other devices as well as Alice's, protecting them from subsequent access via the cell phone. If Bob's phone was stolen and Bob does not remove it from his cluster promptly, the thief could gain remote access to Bob's other devices, illustrating one inherent risk of greater connectivity.

4. DESIGN

This section outlines the elements of UIA's design that are key to realizing the user experience described above.

4.1 Device Identity and State Management

To give UIA devices strong, decentralized identities, each device hashes the public key of a locally-generated key pair to construct its *endpoint identifier*, or EID. As in similar identity schemes [11, 12, 16], EIDs are cryptographically unique, self-configuring, and self-certifying, but not human-readable.

To manage user-friendly names and their associations with secure EIDs, a UIA device generates *change records* in response to user actions affecting the namespace and stores these records in an append-only per-device log. Each device signs the change records it creates, and devices with namespace sharing relationships replicate each other's logs via gossip. To resolve user-relative names, a device examines both the records in its own log and those in its replicas of other device's logs; no network communication is performed during name resolution. Replicating logs in this manner guards users' state against device loss or failure and keeps the namespace available during periods of disconnection.

Table 1 summarizes the essential information contained in the most common types of change records. This table is not intended to be complete or definitive, but merely to illustrate the general design strategy. Additional information and record types may be needed in the future, for example, to support more sophisticated namespace sharing and access control policies.

4.2 Name Resolution

To resolve UIA names, each device maintains a tree representing the names embodied in its log and in its replicas of other devices' logs. The internal nodes of this tree consist of *create namespace* records that have been transitively merged via *merge* records, and the branches of the tree consist of named *link* records that join these merged namespaces. Each device designates a single *create namespace* record as the root of its local tree.

For compatibility with DNS, UIA names follow the same formatting rules as DNS names, consisting of a series of *labels* separated by dots. To resolve a specific name such as `PC.Alice` on Bob's phone, for example, Bob's phone first parses the name into its component labels, `PC` and `Alice`. Starting from its own root namespace, Bob's phone then traverses its namespace tree by following successive links corresponding to the labels in the name, working from right to left as in DNS: from Bob's root namespace to Alice's root namespace via the label `Alice`, then from Alice's root namespace to Alice's PC via the label `PC`.

The remainder of this section clarifies the above rough summary of the name resolution process, by following the sequence of events shown in Figure 1.

4.3 Naming Devices and Building Clusters

To implement the preconfigured name for Bob's VoIP phone in Figure 1, UIA writes a *create namespace* record at the beginning of the phone's log, designates it as the root of the phone's namespace, then writes a *name device* record with the phone's EID, a pointer to the root namespace record, and the name `phone`. In this way, the phone's local name for itself is just `phone`: the name `phone` is contained in the *name device* record, and there are no named *link* records between it and the root.

When users introduce devices to form clusters, each device writes a *merge* record merging its local root with that of the other. *Both* devices must write, digitally sign, and exchange corresponding *merge* records in order either device to consider the merge process complete. If a device encounters an unpaired *merge* record, which could happen due to a hardware failure or loss of connectivity during a merge, for example, the device treats the unpaired *merge* record as a "conflict" that the user can resolve at leisure by completing or canceling the merge.

Record Type	Fields	Purpose
create namespace		creates a new empty namespace
link	*parent pointer*, *child pointer*, *name*	maps *name* to the namespace *child pointer* in the namespace pointed to by *parent pointer*
name device	*parent pointer*, *device EID*, *name*	maps *name* to the device *device EID* in the namespace pointed to by *parent pointer*
merge	*local pointer*, *remote pointer*	merges the namespaces pointed to by *local pointer* and *remote pointer* so that they share device names and social links placed into either one
unlink	*pointer*	unmaps the namespace linked in the record pointed to by *pointer*
remove name	*pointer*	removes the device name created in the record pointed to by *pointer*
stop merge	*pointer*, *stop seq.*, *stop hash*	stops importing new records from the namespace imported at *pointer* after the record with sequence *stop seq.* and hash *stop hash*

Table 1: *The primary types of log records in UIA.* In addition to the fields shown, each record contains the device EID, a sequence number, and the secure hash of the record that preceded it. Pointers to records are implemented as the triple of these three fields.

When Bob introduces his laptop to his phone at Time 2 in the figure, for example, the two devices merge each other's root namespaces. Since their root namespaces are now linked by corresponding *merge* records, they are treated as the same node in the logical namespace tree, and each device can now refer to the other via *name device* records taken from either original namespace. The laptop, for example, refers to the phone simply as `phone`, since that is the name in the phone's *name device* record, and there are no named *link* records between it and the newly-merged root.

4.4 Introducing and Naming Users

UIA uses the *link* record type to provide access from one user's devices to those of another. When Alice and Bob link their devices at Time 3 in the figure, Alice's iPod links Bob's root namespace into her own by writing a *link* record with the name `Bob` and pointers to the two relevant namespaces. Bob's phone similarly mirrors these actions, linking its own root namespace to the iPod's root namespace via the name `Alice`.

Alice can now refer to Bob's phone, for example, as `phone.Bob`. To resolve this name, Alice's iPod follows its new *link* record for the label `Bob`, from Alice's root namespace to Bob's root namespace, then from Bob's namespace to Bob's phone via Bob's original *name device* record for `phone`, originally written on Bob's phone but gossiped to Alice's iPod after the introduction.

4.5 Merging Clusters

At Time 4 in the figure, Bob returns home and merges his new cell phone with his home phone. Bob's logical root namespace, and in effect Bob's "user identity" for UIA's purposes, is now transitively defined according the set of valid *merge* record pairs that link together the root namespaces of Bob's devices: namely the *merge* pair Bob generated earlier at time 2, and the new *merge* pair from Time 4. Bob's laptop and cell phone thus discover each other through gossip with the home phone and begin gossiping together in turn. Likewise, Bob's laptop and home phone learn of Alice's devices through his cell phone, and Alice's devices similarly learn of Bob's additional devices.

Although Alice's existing *link* record for `Bob` only directly contains the EID of Bob's new cell phone, all UIA devices now treat this *link* record as logically referring to all of Bob's merged root namespaces as defined according to his *merge* records, giving Alice a convenient name for Bob's implicit "user identity." Any UIA device now treats records affecting any of Bob's merged root namespaces as affecting all of them, making Alice's *link* record for `Bob` in effect name Bob *as a user* rather than Bob's cell phone. To resolve the name `laptop.Bob` on Alice's iPod, for example, the iPod follows Alice's *link* to the root namespace of Bob's cell phone, then follows Bob's *name device* link for `laptop` to find the laptop's EID, even though the latter *name device* record was originally written with reference to Bob's laptop's root namespace.

By identifying users indirectly via their clusters in this way, UIA avoids imposing on users the burden of having to manage any kind of explicit *user identifiers*, or the per-user cryptographic key pairs that would presumably be needed to generate such identifiers securely. UIA can therefore give logical identities and meaningful names to both users and devices, but only devices actually need to have explicit identifiers, which they can create automatically for themselves.

4.6 Groups

Though not illustrated in the figure, UIA easily supports shared groups as well. For example, the members of a household may list their common devices under the suffix `.home` by creating a new namespace on each device, linking from the device's root namespace to the new namespace via the name `home`, and then merging all of the new namespaces together. Moreover, the devices need not all merge directly to each other; a single spanning tree suffices to join all the devices' respective `home` namespaces.

4.7 Resolving Conflicts

By merging his home and cell phones at Time 4, Bob creates a conflict in his cluster, as the name `phone` is now mapped to two different EIDs, and he renames one device `cell` to resolve it. To accomplish this renaming, UIA logs a new *name device* record for the name `cell`, followed by a *remove name* record pointing to the original *name device* record for that EID. The *remove name* record does not actually delete the original *name device* record, but merely causes all devices that see it to ignore the original *name device* record for purposes of name resolution and conflict detection. Since there is now only one "active" *name device* record for the name `phone` in Bob's root namespace, the conflict is effectively resolved on each device as soon as that device obtains the *remove name* record via gossip.

4.8 Lost or Stolen Devices

To remove the lost cell phone from Bob's cluster at Time 6 in the figure, his laptop logs a *stop merge* record pointing to the last known legitimate entry in the cell phone's log. This record instructs Bob's devices to continue using old names that the cell phone created, while ignoring any subsequent records that may be created by the phone, if the cell phone falls into the hands of a thief for example. To prevent the cell phone from introducing conflicting versions of old records, each UIA log record contains the secure hash of the record that preceded it. The secure hash of a single record thus secures an entire prefix of the log.

Bob's laptop also logs a *remove name* record to delete the cell phone's EID from his namespace. The pointer to the initial creation record in the remove record allows Bob to reuse the name "cell" later with no ambiguity as to which one of the *name device* records the *remove name* record actually refers to.

4.9 Routing

In order to gossip, devices must be able to communicate. To this end, they keep track of the IP addresses of others to which they have been introduced in a local table. To find a peer that is no longer reachable at its last known IP address, a device uses a Gnutella-like expanding ring search through the network of its reachable peers. Each device also remembers old IP addresses to which a peer might have returned in case a search is unsuccessful.

A successful search result includes the full path taken by the search. If a device cannot establish a direct IP-level connection to a peer itself—perhaps because the peer is behind a NAT or firewall and cannot accept incoming connections—the device asks the next-to-last node in the search path to forward traffic to the peer on its behalf. The devices then use this channel to establish a direct connection by "punching a hole" through the NAT [8] if possible; otherwise they continue communicating through the intermediary.

Since UIA devices can route opportunistically through their social neighbors, the ability of any two frequently-moving devices on the Internet to locate each other reliably does not depend on any centralized or manually-configured servers, but only on the existence of *some* UIA device somewhere in the two devices' common "social neighborhood" with an accessible and relatively stable IP address. This "rendezvous device" could happen to be a public server of some kind, perhaps even one specifically set up to help other UIA devices rendezvous, but UIA does not depend on this being the case; the rendezvous device could just as well be a friend's home PC attached via DSL a "well-behaved" or suitably configured NAT.

Each UIA device by default actively monitors the IP addresses only of its *immediate* social neighbors—i.e., the user's own devices and those of his "first-degree" contacts. Since most users are expected to have tens or at most a few hundred immediate contacts, each node's routing traffic burden should be manageable, and thus the system should scale well even if the total size of the interconnected UIA network is orders of magnitude larger. The time required to locate an arbitrary device naturally increases with "social distance," and its chance of success similarly decreases, but this property is appropriate since we expect people to use UIA mostly to communicate within their immediate social neighborhood. Nonetheless, efficient routing to arbitrary EIDs is an important direction for future work.

5. IMPLEMENTATION

A protoype UIA implementation currently runs on Linux and Mac OS X. This prototype is divided into separate routing and naming layers, both of which run as user-level daemons to which UIA-aware applications on the device can directly interface via Sun RPC. Through these interfaces, a UIA-aware application can send packets to EIDs of its choice, listen for packets on its own EID, discover potential peers on the local-area network, enter new names into its namespace, and resolve UIA names to EIDs. The prototype uses Apple's Bonjour library for local-area device discovery and SSL for secure communication between peers.

Furthermore, the UIA prototype provides support for legacy applications that support IPv6. We have successfully used Apache, Firefox, and OpenSSH over UIA, without modification or even recompilation, via this legacy interface. For routing, UIA uses the *tun* device to disguise EIDs as IPv6 addresses. In this way, applications can bind a socket to the local device's EID or connect to a remote device by EID. For naming, UIA provides a local DNS proxy that sends each lookup request to both the UIA naming layer and the device's normal DNS server and returns a combined result.

Currently, users perform device discovery and introduction using command-line programs. A graphical user interface is under development.

6. RELATED WORK

Existing Internet mobility mechanisms require configuration effort and technical expertise that deters even many sophisticated users. Dynamic DNS [19] supports automatic IP address updates, but devices still become inaccessible when behind a NAT [9]. Mobile IP [14] gives a mobile device the illusion of a fixed IP address, but requires a dedicated forwarding server at a static, public IP address. UIA in contrast relies on user-relative names, self-configuring EIDs, and opportunistic use of peer devices for rendezvous and traffic forwarding to support mobility.

Ad hoc naming schemes such as Bonjour [1] allow devices to choose their own names on local-area networks, but these names are insecure and ephemeral: any device joining a network can claim any unused name, and a device's name becomes invalid as soon as it moves to a different network. UIA uses the Bonjour libraries to discover new devices on the local network, but UIA names persist and remain securely bound to the original target device despite later migration of the devices involved.

UIA's user-relative naming model may be useful to other systems that use cryptographic host identifiers, such as HIP [12], $i3$ [17], and SFR [2]. Though UIA takes advantage of global infrastructure when available, UIA does not depend on it and can continue providing naming and communication among local devices even when disconnected from the Internet.

UIA's user-relative naming model is inspired in part by SDSI/SPKI [4, 5, 16]. Like SDSI, UIA identifies devices by their public keys, and allows users to define relative names. SDSI's model for designated certificate servers does not adapt well to disconnected mobile devices, however. UIA also simplifies key management by identifying users implicitly via their personal clusters instead of requiring them to manage per-user public/private key pairs explicitly, and UIA handles lost or stolen devices without rekeying and thus losing the user's identity.

UIA's relaxed consistency model is partially inspired by Bayou [18] and Ivy [13]. The semantics of UIA's log records do not require devices to converge on a single total ordering of their logs, however, simplifying conflict detection and resolution.

Finally, UIA is a continuation of work begun with the Unmanaged Internet Protocol [6, 7]. Unlike the earlier work, however, UIA routes along overlay links that mirror the social trust relationships of the devices involved, as in Turtle [15] and SPROUT [10].

7. CONCLUSION

UIA is a connectivity architecture that facilitates global peer-to-peer sharing of information and services between personal devices by giving technically unsophisticated users a simple, intuitive, and secure way to name both their devices and other users. By combining local device introduction and user-relative naming with self-certifying global device identities and globally distributed personal namespaces, UIA represents a unique application of social networking concepts to the problem of ad hoc peer-to-peer naming.

ACKNOWLEDGEMENTS

This research is sponsored by the T-Party Project, a joint research program between MIT and Quanta Computer Inc., Taiwan, and by the National Science Foundation under Cooperative Agreement No. ANI-0225660 (Project IRIS).

REFERENCES

[1] Apple. Bonjour. http://developer.apple.com/networking/bonjour/.

[2] Hari Balakrishnan, Scott Shenker, and Michael Walfish. Semantic-Free Referencing in Linked Distributed Systems. In *IPTPS*, 2003.

[3] G. Danezis, C. Lesniewski-Laas, F. Kaashoek, and R. Anderson. Sybil-resistant DHT routing. In *ESORICS*, 2005.

[4] C. Ellison. SPKI Requirements, 1999. RFC 2692.

[5] C. Ellison et al. SPKI Certificate Theory, 1999. RFC 2693.

[6] Bryan Ford. Scalable Internet routing on topology-independent node identities. Technical Report MIT-LCS-TR-926, MIT Laboratory for Computer Science, October 2003.

[7] Bryan Ford. Unmanaged Internet protocol: Taming the edge network management crisis. In *HotNets*, 2003.

[8] Bryan Ford. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference*, 2005.

[9] M. Holdrege and P. Srisuresh. Protocol complications with the IP network address translator, 2001. RFC 3027.

[10] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina. SPROUT: P2P routing with social networks. In *P2P&DB*, 2004.

[11] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *SOSP*, 1999.

[12] R. Moskowitz and P. Nikander. Host identity protocol architecture, April 2003. Internet-Draft (Work in Progress).

[13] Athicha Muthitacharoen, Robert Morris, Thomer Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *OSDI*, 2002.

[14] C. Perkins, Editor. IP mobility support for IPv4, August 2002. RFC 3344.

[15] B. Popescu, B. Crispo, and A. Tanenbaum. Safe and private data sharing with Turtle: Friends team-up and beat the system. In *Proc. of the 12th Cambridge Intl. Workshop on Security Protocols*, 2004.

[16] Ronald L. Rivest and Butler Lampson. SDSI – A Simple Distributed Security Infrastructure, 1996. http://theory.lcs.mit.edu/˜rivest/sdsi10.html.

[17] Ion Stoica et al. Internet indirection infrastructure. In *ACM SIGCOMM*, 2002.

[18] Douglas Terry et al. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP*, 1995.

[19] P. Vixie, Editor, S. Thomson, Y. Rekhter, and J. Bound. Dynamic updates in the domain name system (DNS UPDATE), April 1997. RFC 2136.