Untangling the Web from DNS

Michael Walfish[a], Hari Balakrishnan[a], and Scott Shenker[b]
IRIS Project
[a]{mwalfish, hari}@csail.mit.edu, MIT Computer Science and AI Laboratory (CSAIL), Cambridge, MA
[b]shenker@icsi.berkeley.edu, International Computer Science Institute (ICSI), Berkeley, CA

Abstract

The Web relies on the Domain Name System (DNS) to resolve the hostname portion of URLs into IP addresses. This marriage-of-convenience enabled the Web's meteoric rise, but the resulting entanglement is now hindering both infrastructures—the Web is overly constrained by the limitations of DNS, and DNS is unduly burdened by the demands of the Web. There has been much commentary on this sad state-of-affairs, but dissolving the ill-fated union between DNS and the Web requires a new way to resolve Web references. To this end, this paper describes the design and implementation of *Semantic Free Referencing (SFR)*, a reference resolution infrastructure based on distributed hash tables (DHTs).

1 Introduction

DNS's original goal was practical and limited—allow users to refer to machines with convenient mnemonics [20, 21]—and it has performed this service admirably. However, with the advent of the Web and the resulting commercial value of DNS names, profit has replaced pragmatism as the dominant force shaping DNS. Legal wrangling over domain ownership is commonplace, and the institutional framework governing the naming system (*i.e.*, ICANN) is in disarray. Commercial pressures arising from its role in the Web have transformed DNS into a branding mechanism, a task for which it is ill-suited.

At a logical level, a linked, distributed system such as the Web requires a *Reference Resolution Service* (RRS) to map from *references* (our generic name for links or pointers) to actual locations. In the current Web, references are URLs with a hostname/pathname structure, and DNS serves as the RRS by mapping the hostname to an IP address where the target is stored. As the Web has matured, content replication and migration have become more important. However, the host-based nature of URLs—which ties references to specific hosts and hard-codes a path—makes content replication and movement hard.[1] Consequently, there have been many sensible calls, most notably in the URN literature [2, 5, 9, 19, 28, 29], to move the Web away from host-based URLs.

Since the Web has imposed the burden of branding on DNS, and DNS has restricted the flexibility of the Web, we believe that both systems would benefit if they were disentangled from each other. However, dissolving this mutually unhealthy union would require a new RRS for the Web. What should such an RRS look like? There has been extensive discussion about this topic, largely within the URN community but among many others as well. While we don't provide a comprehensive review of the commentary, the literature suggests the following two basic requirements for any such RRS (both of which DNS-based URLs do not satisfy):

Persistent object references: *A Web reference, like any abstraction used for indirection, should always be invariant, even when the referenced object moves or is replicated.* This principle has been central to the discussion about URNs. Reference persistence implies that references should not be tied to particular administrative domains or entities, as they are currently in DNS.[2]

Contention-free references: *Reference choice should be free of ownership disputes or other forms of legal interference.* Disputes over human-readable names are inevitable [22], but the reference resolution infrastructure is a poor place to resolve those disputes. Thus, as has been observed in the past [1, 10, 24, 29], references should be *inherently* human-unfriendly; in fact, we believe the infrastructure should enforce this property. Of course, users must be able to associate meaning to references, but the binding between human-friendly names and these opaque references should be done *outside* the referencing infrastructure. Such a separation would (a) free the RRS to focus only on technical concerns and (b) permit multiple, competing solutions to human-friendly naming, thereby allowing the resulting tussle [3] to play out through legal and other social channels.

[1] Because DNS names hosts, not Web objects, it is easy to move and replicate *hosts*. But DNS requires the sophisticated algorithms and substantial infrastructure of content distribution networks (CDNs) to achieve the same goals for individual Web *objects*.

[2] For instance, consider the Web page of someone who first created the page while at institution X but later moves to institution Y. If the reference record is controlled by the X domain (as it is with DNS) then maintaining persistence would require that X allow the author to update the record (if only to provide an HTTP redirect) for all time, even when the author is no longer affiliated with X. This expectation is impractical.

To these two well-accepted requirements, we add a third and less universally accepted design goal (which is similar in spirit to the goal articulated in [33]).[3]

General-purpose infrastructure: *The RRS should be designed to support a wide class of "link-based" applications.* The use of links, or pointers, to refer to objects or content on other machines is not unique to the Web; links are used in a variety of distributed systems for identifying objects and invoking remote code, for locating devices, and for other purposes when one wants to refer to objects by name, not location. The URN literature deals with this multiplicity by having context-specific resolvers [9]. However, since reference resolution is a hard problem that requires delicate design, we believe it should, if possible, be solved once and well.

How does one build a general-purpose RRS for persistent and contention-free references? In our work here we followed two key design principles:

Semantic-free namespace: We believe that the simplest way to achieve persistence and contention-free references is to use a namespace devoid of explicit semantics: a reference should neither embed information about the organization, administrative domain, or network provider it originated in or in which it is currently located, nor be human-friendly. We call such references *semantic-free*.

Minimal RRS interface and factored functionality: A general-purpose RRS should not impose unwanted semantics on applications, implying that the RRS should support a minimal interface limited to reference resolution. Therefore, all other functions required by applications—including mapping between human-friendly names and references—should be handled by auxiliary systems. We believe that the RRS's job is to *provide a platform* that allows for competition and flexibility in application-specific support and not to *solve directly* these higher-level problems.

We used these two design principles to develop both an RRS with **semantic-free references** (SFR) and a version of the Web that uses only SFR. The result is a system decomposition that differs from today's Web: whereas humans today rely on being able to read, and occasionally type, references (URLs), the Web-over-SFR handles user-level naming outside the reference resolution service by enabling a competitive market for *canonicalization services* that map human readable names to semantic-free tags. In the Web-over-SFR, search engines function as they do today, except they return links backed by semantic-free tags rather than by DNS-based URLs. Web browsers in turn use the SFR infrastructure to resolve these semantic-free tags to meta-data like IP addresses, ports, and pathnames that identify Web objects.

[3]The Globe project [32, 33] shares many of the same motivations as SFR, but, as we discuss in Section 7.2, the set of technical challenges addressed is rather different.

In addition to a different factoring, SFR enables new functionality for the Web, including: object-based migration wherein objects can move without requiring referring links to be updated or broken; flexible object replication wherein individual objects can be replicated without heavyweight machinery, administrative control over the hosts of the replicas, or hard-coding the administrative entity responsible for the meta-data; and content location services wherein individuals can provide reliable pointers to objects they did not contribute.

SFR is a "clean-sheet" design; not only does our design, in its pure form, require changes to all Web browsers, it also requires an infrastructure that currently does not exist. The Web-over-SFR is incrementally deployable via Web proxies, and a transition strategy exists, but we will not dwell on these methods. Our goal is rather to investigate, without regard to deployment issues, how one might best support the Web and other applications that require reference resolution. We hope that the lessons learned in this exercise will be useful, indirectly if not directly, in any future evolution of the Web and DNS.

2 SFR Challenges

SFR's advantages do not come without cost. Many of the desirable features of today's Web derive from DNS. As examples, DNS's hierarchical structure enforces URLs' uniqueness and provides fate sharing (a disconnected institution can still access local pages) while the human readability of DNS hostnames gives users some (perhaps misguided) confidence they have reached their desired data. Since SFR has abandoned both hierarchical structure and human readability, the SFR design must explicitly provide for the properties we have mentioned and others like them. Some of these challenges must be met by SFR itself, and some should be left to auxiliary systems supporting the Web-over-SFR. Addressing these challenges is the main focus of this paper. We now briefly discuss them and defer solutions to Sections 3 and 4.

2.1 SFR Infrastructure Challenges

Scalable resolution. Until recently, there was no way to scalably resolve references in a semantic-free namespace, which is largely why the URN literature chooses a partitioned set of resolvers [9]. However, the recently developed DHT technology [15] is designed to do exactly this: at their core, DHTs map an unstructured key from a flat namespace to a network location responsible for the key. But typical DHTs require $O(\log n)$ hops per lookup in an n-node system and would introduce intolerable latency. Thus, SFR must provide, on average, significantly faster lookups than the usual DHT performance bound.

Security and integrity. Any RRS must secure content providers' meta-data and enforce *reference integrity* by preventing two logically distinct objects from receiving the same reference. DNS guarantees these properties by relying on the administrator of each delegated namespace to protect meta-data and avoid local conflicts. A semantic-free namespace, however, has no natural administrative partitioning and thus protecting references—even under network partitions and malicious clients—is non-trivial.

Fate sharing. By delegating its namespace, DNS naturally offers fate sharing: if a domain (*e.g.*, `foo.edu`) becomes disconnected from the rest of the Internet, users in that domain can usually still access content served by that domain (*e.g.*, content at `x.foo.edu`), since the authoritative name server (*e.g.*, for `foo.edu`) is typically on their side of the partition. Since semantic-free names do not reflect objects' origin, SFR must be explicitly structured to provide fate sharing.[4]

Trust and financing. DNS has a very simple financing and trust model: organizations provide the *authoritative* server for their own domain, and DNS nodes need only serve requests *for* hosts or *from* users within the domain. Moreover, DNS requires only a small common infrastructure—the root servers—so all other expenses are incurred by the organization reaping the benefit (by allowing others to access their hosts). References in SFR are not tied to the content provider, so the "serve your own" trust and financing model does not apply.

2.2 Web-over-SFR Challenges

When the Web (or other applications with human interaction) runs over SFR, two important challenges arise: how users will find objects and how users can be sure that the content they see corresponds to the object they are seeking. *Rather than seek a single all-encompassing solution to these problems, we instead factor our system so that multiple, competing solutions can arise.*

Canonical names. There is good reason for the contention over DNS domains; they allow URLs to serve as *canonical names* that are memorable, human-readable, and easily transcribed. In contrast, the SFR approach provides opaque bit strings with none of these useful features. There is great benefit in simple and recognizable URLs, such as `http://www.cnn.com`. Thus, similar sets of canonical names that users can remember, understand, and transcribe must exist in the SFR framework.

Confidence. Humans browsing the Web are usually confident that URLs beginning with `www.nytimes.com` identify content published by *The New York Times* newspaper. While this reliance on the human semantics of a URL is hardly foolproof (as recent scams [4] have demonstrated), it does represent an important user need. SFR must clearly provide an alternative mechanism for giving users confidence in the content they are viewing.

[4]We must address fate sharing because we insist on semantic-free references not because we use DHTs. The SkipNet DHT [12] provides fate sharing, but it encodes administrative domains into references.

SFRTag:	`0xf01212099abcab678ac345ba4d...`
location:	*(ip, port),(DNS name, port), SFRTag*
oinfo:	*App-specific meta-data*
ttl:	*time-to-live: a caching hint*

Figure 1: The `o-record`

3 SFR Design

Our proposed SFR system is a *shared infrastructure* that provides a single service: mapping from a semantic-free tag that references an object to meta-data associated with the object. Content providers insert an object's meta-data into the infrastructure and associate it with a tag. Consumers of the content submit these tags to the infrastructure and receive object meta-data in response. In this section, we focus on this single service and not on auxiliary questions, like how human-friendly names are mapped to tags.

3.1 Essentials

SFR uses a distributed hash table (DHT) to map semantic-free 160-bit strings, `SFRTags`, to `o-records` ("object records"). The `o-record`, shown in Figure 1, contains an object's location and other meta-data. The SFR infrastructure does not store objects, only their `o-records`. Our implementation uses Chord [31] as the underlying DHT routing protocol and DHash [8] to store the `o-records`, but the SFR architecture is modular and permits another DHT protocol to be substituted. In fact, SFR could use any system that supports scalable lookups on unstructured identifiers (such as the location service in the Globe system [32, 33]). For convenience, we will refer to DHTs as the fundamental resolving technology.

The `location` field is set by the application inserting the `o-record` and holds one or more values describing the location of the data corresponding to the `SFRTag`. Each `location` field entry is either an IP address and transport port pair, a domain name and transport port pair, or another `SFRTag` (that in turn resolves to another `o-record`). These latter two options permit additional degrees of indirection so that if many objects migrate together, they can all be updated by a single change to, respectively, DNS or SFR (the SFR option allows objects to move to different hostnames, while the DNS option enables changes to IP addresses for a fixed hostname). SFR's use of DNS to abstract the location of a *host* is not a contradiction; as we noted before, DNS is designed for exactly this function.

The resolving infrastructure imposes almost no constraints on applications since the structure, length, and content of the oinfo field are application-defined; *e.g.*, for the Web application, the field could hold the type of transport protocol (HTTP, FTP, HTTPS), a pathname on the server, etc. The SFR infrastructure does not look at this field. Finally, like DNS's TTL, the ttl field in the o−record is a caching hint instructing entities outside the infrastructure about how long to cache a given o−record. Because SFRTags are persistent references, the copy of the o−record in the infrastructure never expires and so SFRTags cannot be reassigned. As a result, if a content provider wishes to retire an o−record because the reference is no longer valid, the content provider empties the o−record.

The trust and economic model we envision for SFR is quite different from that of DNS because one cannot "serve your own" when the tags are semantic-free. So, instead of a DNS-like infrastructure comprised of "donated" machines dedicated to specific domains, we envision a more uniform infrastructure in which SFR nodes are trusted to serve all o−records. While there may be a startup period during which an "angel" (*e.g.*, NSF, European Union) funds the initial infrastructure (which may require, say, only 1,000 machines), once SFR becomes accepted as a viable service there could be competing commercial offerings. We believe that eventually several competing SFRs could peer with each other (exchanging updates) much like today's tier-1 ISPs, each holding mirror copies of all data. These peered SFRs would together form the global SFR infrastructure.

These SFR structures would be managed infrastructures with good connectivity (*we repeat: even though we are using DHTs, which are a so-called P2P technology, we are* **not** *relying on flaky personal machines connected via cable modems!*), so the SFR infrastructure machines would be relatively stable[5] and bandwidth between them relatively plentiful. Obviously the issue of the economics of such infrastructures is an open question, and our design thus relies on the shaky premise that competitive SFR infrastructures would arise; however, here we hope to convince the reader only that such infrastructures would indeed offer a better solution to the problems in today's Web and other linked services.

Before describing the rest of SFR's design, we emphasize that SFR's challenges derive from its semantic-free namespace, and it is this characteristic, rather than the particular choice of resolving substrate, that identifies SFR. One way to implement SFR may in fact be to use semantic-free DNS names. Indeed, one transition strategy is deploying various nameservers for a .sfr domain that would look up references in an SFR infrastructure. We do not view this as a contradiction: our objective is not to eliminate DNS but to change the way the Web uses DNS

[5] Nevertheless, in Section 6, we demonstrate that lookup performance remains acceptable under node failures.

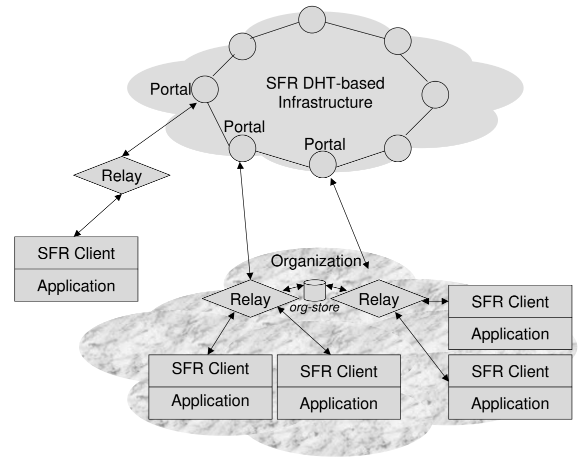


Figure 2: SFR components.

to resolve references. A system of semantic-free names built on DNS would face the same challenges as SFR and would require similar machinery, such as a way to do scalable resolution in an unpartitioned namespace.

3.2 SFR Components

Figure 2 shows the components of the SFR system. At the core is the *SFR infrastructure*, a collection of managed nodes (of the kind we described above) that run *SFR server* software. This software runs on top of a DHT protocol and storage manager implemented at each node.

Applications store and retrieve o−records corresponding to SFRTags using the *SFR client* library. The client interacts with the SFR infrastructure using an *SFR relay*, a software module that intermediates between client requests for storing and retrieving o−records and the SFR infrastructure. The relay handles o−record caching and also ensures that clients can gain access to the o−records for content hosted by the local organization even when the organization is disconnected from the SFR infrastructure. The relay itself does not need to implement the DHT routing protocol or the storage manager; it connects to the SFR infrastructure at an *SFR portal*, which is simply a node in the infrastructure.

If SFR becomes widely deployed, client machines will need to discover a reachable SFR portal or relay. Clients today find out about available DNS servers via DHCP or via hard-coding; we envision identical techniques for SFR. Providing access to an SFR portal or relay would be one of the services offered by an ISP or large institution.

3.3 Security and Integrity

We now describe (a) how content providers who are clients of SFR may create unique, contention-free references without administrative namespace delegation and (b) how the infrastructure secures the content providers' meta-data. SFRTags and their associated o−records have these properties:

- The infrastructure ensures that `SFRTag`s are the output of a hash function and thus have no human meaning.
- Content providers can create unique references without consulting a naming authority or any other entity.
- Only an `o-record`'s creator, or someone who shares his private key, can update that `o-record`.
- Given a reference, the `o-record` is self-certifying.
- Content providers can update their public keys without invalidating references.
- The namespace is too massive for anyone to monopolize a significant chunk of it.

To achieve these properties, the SFR infrastructure first requires that an `SFRTag` is a secure, collision-resistant hash of the content provider's public key and an arbitrary salt. So when a content provider wishes to **create or update** a reference, it sends to its SFR portal (perhaps via an SFR relay), a request with all of the following elements:

- `o-record`, and `SFRTag` = hash(public key, salt);
- public key, salt, and version;
- signature (`o-record`, salt, version);

Before accepting this request, the responsible SFR infrastructure node checks that the `SFRTag` is the correct hash and that the signature is valid. The SFR infrastructure node then stores all of the data listed above. If the `SFRTag` was already in the infrastructure, the responsible node further checks that the request is signed with the current private key. (For clients who do not use public keys—and thus receive no protection—the SFR infrastructure also accepts references that are the hash of a client-chosen salt, only.) Because each reference is the output of a hash function, it is highly unlikely to have mnemonic or branding value, which in turn removes the need for a naming authority or other arbiter. In addition, SFR does not require, or use, a public key infrastructure.

On a **lookup**, a content consumer sends a request for the `SFRTag` to its SFR portal (perhaps via an SFR relay). If the tag is in the infrastructure, the responsible DHT node returns the corresponding `o-record` along with the auxiliary data mentioned above. Returning this data makes the `o-record` *self-certifying* [18]: *i.e.*, without resorting to a public key infrastructure, a retrieving client will be able to tell if a compromised node or malefactor in the middle of the network alters any of the data (since the reference is bound to the public key, and the signature binds the public key to the data). Moreover, SFR clients ensure that they are hearing from *bona fide* SFR infrastructure nodes by verifying the signatures on messages sent to them from the infrastructure nodes. We presume that when a client receives the address of an SFR node (*e.g.*, by DHCP, as with DNS servers), the notification also includes that SFR node's public key.

Public key updates do not invalidate the reference since the SFR infrastructure requires only that the relationship between the `SFRTag` and the public key is satisfied when the tag is *first* inserted. After that, the SFR infrastructure ensures that updates to the public key or to the content have been signed with the existing private key. To maintain the self-certifying property, the infrastructure must store the content provider's signed request to update its public key and must also return these signed requests in response to lookups.[6] To guard against replay attacks, SFR adopts DHash's approach [8]: SFR clients increment, and sign, a version number each time they update their `o-record`, and the infrastructure accepts updates only with increasing version numbers.[7]

The sheer size of the SFR namespace prevents anyone from monopolizing a significant portion. Protecting individual SFR nodes or the DHT as a whole against loading is a different matter, however. Our intent is that these attacks will be addressed by management tools to prevent content providers (where a content provider is defined by its public key) from using too many resources.

3.4 Latency

SFR uses three kinds of TTL-based caching to reduce latency and balance load among the infrastructure nodes. First, each relay caches `o-records` (and the auxiliary data like public keys and signatures), sharing that cache among the clients that use that relay. Because the use of a relay is optional, SFR clients also cache `o-records`.

Second, each DHT node in the infrastructure keeps a *location cache* of identifier-to-IP mappings for nodes it has recently heard about. This reduces the number of hops in certain DHT routing schemes that require $O(\log n)$ hops in an n-node system. In Section 6 we present simulation results showing that location caching can lower the number of hops to two or three in over 99% of lookups. "One-hop" DHT routing schemes [11] are another way to lower the number of hops.

Third, SFR infrastructure nodes also cache `o-records`, which helps balance load and ease hot-spots corresponding to highly popular `o-records`: such `o-records` will quickly be cached by the portals and so should not stress the infrastructure. In addition to each portal's caching the `o-record` retrieved on behalf of a relay or client, our design also permits nodes on the DHT lookup path to cache `o-records`, thereby proactively populating their cache.[8]

[6] Without additional infrastructure, the loss of a key could be catastrophic, but one could imagine auxiliary services that would serve as trusted and secure repositories of such keys. However, if a key is compromised, the situation is more dire; we think the only way the original owner can prevent the takeover of his content is to *break* all current tags (*i.e.*, render them unusable by anyone, adversary and victim alike).

[7] While we don't discuss replication explicitly here, DHTs need replication to provide reliability. Thus, retrieving clients may need to download from several locations to ensure they have the latest version number.

[8] In general, improving the performance of DHT-based systems is

3.5 Fate Sharing and Scoping

In the following, we define an "organization" as a set of machines behind an access link. If an organization corresponds to a single DNS domain, and if the organization's DNS servers are also behind the access link, then, when the link fails, hosts in the organization can continue to reach data within the organization. As described so far, SFR does not provide such organizational fate sharing because an organization's `o-records` are not explicitly associated with, or stored within, the organization.

However, SFR can ensure that clients in the same organization as the *creator* of an object can access the object when an access link fails, thereby replacing domain-based fate sharing with what we call *write-locality*-based fate sharing. The enabling mechanism is a shared *org-store* holding copies of the `o-records` created or modified within the organization. Each time a new `o-record` is created or modified via one of the relay nodes in the organization, the relay stores a copy in the org-store and arranges for it to be stored in the SFR infrastructure.

When retrieving, the relay first checks its internal `o-record` cache. If the `o-record` is in the local cache and the TTL is still valid, the relay returns the `o-record` to the client. Otherwise, the relay contacts its portal to initiate a lookup of the `SFRTag` in the SFR infrastructure. At the same time, the relay contacts the org-store, which returns the `o-record` corresponding to the tag if one exists, *disregarding* any TTL value set in the `o-record`. If the relay does not hear from the SFR infrastructure, it times out and infers that it cannot access any of the persistent copies in the infrastructure. It returns to the client the `o-record` returned by the org-store.

The reason the relay does not directly send the version from the org-store *before* waiting for a response from the SFR infrastructure is that another content provider, that shares the same private key but is located in *another* organization, may have updated the `o-record`. For this reason, whenever the relay retrieves an `o-record` from the infrastructure, it also sends a copy to the org-store so that the versions in the org-store and the infrastructure can be reconciled if necessary. The version number in the `o-record`, incremented on each update, and a UTC timestamp set by the writer indicating the last update time, facilitate this reconciliation.

Updating an `o-record` via a relay within the organization also requires the update to be sent *both* to the infrastructure and to the org-store. If the relay finds that the infrastructure store request does not succeed because of lack of connectivity, it asks the org-store to reconcile the `SFRTag` whenever the organization is reconnected. Updating an `o-record` that was originally created in a different organization does not immediately update the org-store in the creating organization; that update happens when the `SFRTag` is looked up via a relay in the original organization. This level of inconsistency is unavoidable without out-of-band synchronization.

The foregoing scheme improves availability for disconnected organizations but does not ensure that infrastructure nodes hold up-to-date versions of `o-records`. If an organization remains internally connected, the semantics of this write-locality-based cache are:

- For `o-records` created in an organization and never updated from outside the organization, clients within the organization always get the most recent version.
- For `o-records` updated by more than one organization, a client within the currently disconnected organization *may* receive an older version that is no older, and is possibly newer, than the last version written from within the organization.
- When connectivity between the SFR infrastructure and an organization is restored, (1) all subsequent retrievals from within the organization return the `o-record` with the highest version number, and (2) all subsequent retrievals from outside the organization return the latest version after it has been reconciled.
- Reconciliation of `o-records` uses either the later timestamp (which works reasonably well assuming loose clock synchronization between writers and preserves the same semantics as when multiple writers update an `o-record` while connected to the SFR infrastructure), or an out-of-band mechanism (*e.g.*, by discarding one of them, perhaps with human involvement). We believe this approach is reasonable because conflicting updates are likely to be rare and suggest the absence of higher-level human coordination.
- Unlike with DNS, clients in different organizations on the same side of a network partition are not guaranteed to be able to access the other organization's meta-data.

Scoping arises naturally in the org-store framework: if clients within the organization wish to limit their meta-data to the organization, the relay simply stores the `o-record` in the org-store only.

4 The Web-over-SFR

In today's Web, references (*i.e.*, URLs) encode the administrative entity (*i.e.*, the domain) responsible for an object's meta-data. Thus, if an object changes domains, hyperlinks to the object are almost guaranteed to break, and a human browsing the Web might get a "notify the referrer" message. Since references should not have to change

an active area of research, and we expect to use solutions from ongoing work in the community on data replication, load balance, denial-of-service defense, fault tolerance, and protection against compromised nodes. While solutions to these problems are not all currently at hand, we are optimistic that there is no fundamental obstacle to basing SFR on DHTs, and so we focus on the many SFR-specific problems.

SFRTag:	0xf01212099abcd3848123ab38121
(ip_addr1, port1, proto1, path1),	
(DNS name, port2, proto2, path2), ...	

Figure 3: Logical view of the o-record for the Web-over-SFR. The proto field specifies the access protocol (*e.g.*, HTTP, HTTPS, FTP). The path field is the local pathname on the server and identifies the referenced object to the server.

when objects move, we attempt, in the Web-over-SFR, to provide a set of references that cleanly permit object migration and replication.

As we have already noted, the current Web supports object migration only if the original *domain* (which may no longer have any connection with the content creator) issues HTTP redirects for objects it no longer hosts. In the Web-over-SFR, in contrast, *all of the information about how to reach a particular Web object*—the IP address and port of the Web server and the pathname on the Web server—is abstracted by the SFRTag. Content creators (*e.g.*, individuals, organizations, research groups) insert this reachability information into an o-record and store the o-record in the SFR infrastructure. To take advantage of these persistent references, Web authors embed hyperlinks like:

sfr://f012120.../optional_path

where f012120... is an SFRTag resolving to a set of tuples identifying the object, as shown in Figure 3.

To retrieve objects using this kind of URL, the Web browser uses the SFR client to fetch the meta-data and construct an HTTP request. The path in the HTTP request is the concatenation of a path from the oinfo field with the optional_path from the original URL. This design preserves HTTP's semantics. The SFR infrastructure is invisible to Web servers, which continue to receive HTTP GET requests with server-specific paths. The optional_path permits flexibility, as we describe below, and it also permits dynamic content because embedded links can have paths with application-specific semantics. Without the optional_path, content providers, Web clients, and Web servers would need to involve the SFR infrastructure to construct unique URLs. That these SFR-based URLs contain semantics illustrates that SFR allows applications to define their own semantics while still using a semantic-free referencing infrastructure.[9]

4.1 Benefits of Web-over-SFR

Resilient linking. The SFR approach permits a general migration solution: if a piece of content, currently referenced by an SFRTag, moves to another Web server at a different path, the content provider need only change the location and oinfo fields in the o-record in order to permit the correct reference resolution to occur for Web clients. Web pages linking to the object continue to maintain the same references.

This approach is flexible about how much the reference functions as an abstraction. An SFRTag can refer to a machine (so the optional_path is the same as it is with today's URLs), to a file (so the SFRTag abstracts the entire URL and the optional_path is empty), or to a directory structure (so the SFRTag abstracts the entire URL up until the root of the directory, and the optional_path is everything underneath the directory). For example, a researcher might have a large collection of publications in one directory and wish to abstract only the collection's location. In this case, the SFRTag would abstract the IP address or domain name of the Web server as well as the path on the server up until the document collection. The publications would be differentiated by their file names. So the SFR URLs could be:

sfr://fbcd123/pub1.ps
sfr://fbcd123/pub2.ps

If the researcher's affiliation then changes, he or she alters the o-record corresponding to fbcd123 and inserts the new Web server and new path on the server. A referring Web page embedding sfr://fbcd123/pub1.ps can safely be ignorant of the move.

If a particular object separates from a directory that had been abstracted by an SFRTag, then, under the design as so far explained, existing references would break. Our solution for this case adds a level of indirection: the content owner would update the o-record to point to a new location that would maintain a map of old pathnames to new location/pathname values. Although this solution can implement HTTP redirection (if the new location were the same as the old server location), our solution does not mandate this approach. In Section 7.1, we discuss a more elegant, but more demanding, solution that does not require this level of indirection.

The main reason the solution we have described above is more powerful than today's DNS-based RRS for achieving resilient linking is that, unlike DNS, SFR is able to resolve both the tag and the pathname before any HTTP messages are sent to the Web server. Achieving similar behavior today would either require a prescient content provider to have a domain name for each potentially movable piece of content beforehand, or rely on HTTP redirection; the former is impractical and a management challenge, whereas the latter is hard to ensure when one moves between organizations.

Flexible object replication. SFR provides a natural solution for replicating Web objects: in response to a request for an SFRTag, the infrastructure can return a num-

[9] Note that DNS could certainly be enhanced with a record type that abstracted individual Web objects, instead of hosts, but as we explain in Section 7.2, such a system would either inherit the problems we have identified with today's use of DNS or else look very much like SFR.

ber of different logical locations and paths. This property might seem inconsequential, but consider how hard it would be under the current Web to replicate a given object in two places *without creating mirror machines containing* exactly *the same content*. To do so would require (a) creating separate DNS names for each *object* being replicated, (b) using virtual hosting so that the two Web servers were configured to recognize each per-object DNS name and (c) configuring DNS entries to refer to both Web servers. Using a domain like `www.personalname.org` would not work since that forces all objects in the domain to be resolved by the same administrative entity forever, making it impossible, *e.g.*, for an individual object like `www.personalname.org/photos` to migrate later without breaking existing, referring hyperlinks. The Web-over-SFR solution is much simpler and would allow several collaborators to replicate each other's content quite easily, yielding a *grass-roots* replication service.

In the case of *massive replication*, namely when it would be absurd or inappropriate to return to the client all of the locations of all replicas, we expect that the SFR infrastructure would direct clients to external services, such as a replication server that would direct requests to the appropriate replica using information from the requester's IP address and other hints. We discuss alternatives to this decision in Section 7.1.

Reliable pointer services. Because SFR permits anyone to insert `o-records` into the infrastructure, third parties can become known as good indirectors—they can create and expose `SFRTags` that always resolve to particular sites or objects, and it would be their responsibility to track the object's movements. Referring Web pages could embed the `SFRTags` established by the indirectors, and then these providers of reliable pointers might have an incentive to make the location service work. For example, a service might provide a pointer to "This year's tax forms"; no matter what year it is, you can access the necessary tax forms by following the `SFRTag`.

4.2 Human-Usability Challenges

The challenges that arise under the SFR framework but which are not explicitly solved by the infrastructure are related to *user-level names*, our term for the ways that humans identify content, such as search queries, typed-in URLs, AOL keywords, hyperlinks in documents, saved bookmarks, and URLs sent in e-mail. DNS-based Web URLs conflate the *reference*, a low-level tag resolved by the RRS, and the *user-level name*, which allows people to find what they are looking for. SFR, in contrast, separates the two functions, focusing on reference resolution and exposing an interface that permits many user-level naming solutions to co-exist.

Canonical names. A natural question is how humans will retrieve content if references are human-unfriendly. The answer is, first, that humans mostly do not depend on typed-in URLs today. All other current user-level naming methods work perfectly under SFR: search queries, for example, return candidate SFR URLs instead of DNS-based URLs. Individuals could also e-mail SFR URLs to each other. (Users are already used to dealing with human-unfriendly URLs this way: links sent in e-mail from `amazon.com`, for example, are essentially a domain name plus a semantic-free string.)

Second, when users do type URLs, they use DNS as a *canonicalization service*: a well-known mapping from human-readable names to Web objects. To permit equivalent functionality under SFR, DNS need not be the canonicalization source. Moreover, it might be desirable if several mapping services existed. We believe that if SFR becomes popular then Web service providers with appropriate expertise would compete to provide such services. Two obvious models already exist—AOL keywords and the paper yellow pages—and we can imagine a wide spectrum of services that map user-level names to a particular `SFRTag` or set of `SFRTags`.

We observe that to the extent DNS provides canonical handles today, it does so mainly for Web *sites* and seldom for individual Web *objects*. Since references in SFR can be as coarse as per-site or as granular as per-object, any canonicalization service for SFR would naturally be able to name entire sites and individual objects, ultimately yielding a more complete canonicalization function than DNS.

Of course, under SFR, the problem of bootstrapping exists, namely how users get pointers to directory services. A number of possibilities exist, including pointers shipped with browsers, links sent in e-mail from friends, applications on the local host that populate a local database of useful sites, and network administrators or ISPs dynamically providing pointers to useful canonicalization services with DHCP. DNS names themselves could provide one canonicalization service, *e.g.*, `www.foo.com` would map to an `SFRTag` for the home page for Foo, Inc. This is similar to the scheme presented by Ballintijn *et al.* [2]. Using DNS in this way is not a contradiction: we are not using DNS for *referencing* but rather as a user-level naming service that competes with any number of other such services.

Confidence. Although human users of the Web usually have confidence in Web content because of the associated DNS name, we note that this level of confidence is actually quite weak. It depends entirely on whether the "correct" company owns a given domain name, and it is easy to create spoof sites that give users misplaced confidence in content. Nevertheless, domain names convey meaning and help users validate URLs before visiting them (*e.g.*, when selecting among search results). Hence, we anticipate that search engines (and others) would hide SFR URLs and give humans confidence in new ways.

Rather than give lengthy detail on ways humans can have well-placed confidence in content under SFR, we outline just one (though imagining others is not hard): hyperlinks on Web pages optionally embed a `taginfo` object alongside the `SFRTag`. This object contains cryptographic statements of the form "Entity E says that this tag is CNN", where E is a Web service provider that users trust. Users' browsers would inform them about who is certifying the link. We note that this scheme is implemented entirely above the SFR layer and that it is an application function, leaving SFR performing only reference resolution. With such a scheme, content authentication could be granular—it could occur at the level of individual objects, rather than at the level of administrative domains (as certificates issued by certificate authorities do today, for example).

Rather than "hard-coding" one approach to canonicalization and confidence, we believe the infrastructure itself should permit multiple schemes to co-exist. Separating the functions of user-level naming and reference resolution will certainly not solve the intractable problem of humans fighting over names. But it will move these tussles [3] to an arena in which multiple services can compete but in which none of these competing services is part of the core reference resolution infrastructure.

4.3 Pragmatics

So far we have focused on the fundamental problems faced by SFR. However, there are more pragmatic concerns that would have to be addressed before the Web-over-SFR could be viable. We don't believe they represent insurmountable difficulties, but they will require new tools. We now mention a few of these issues.

Local references. Currently, with DNS-based URLs, if a hyperlink in a Web page points to another page in the same administrative domain, there are two possibilities: either the hyperlink will be local (*e.g.*, `<A HREF=/imgs/dog.gif>`) or the hyperlink will reference the current domain (*e.g.*, `<A HREF=http://mysite.org/imgs/dog.gif>`). The first case does not involve any lookups, so it would continue to work in an SFR-based Web (although, because references can abstract any portion of the path component, only absolute links will work, not relative ones). In the second case, under DNS, the client need not do another lookup because it would have the address information for `mysite.org` cached.

Under SFR, however, if the content provider used another `SFRTag` to refer somewhere on the same site (*e.g.*, `<A HREF=sfr://ab12126/dog.gif>`), the client

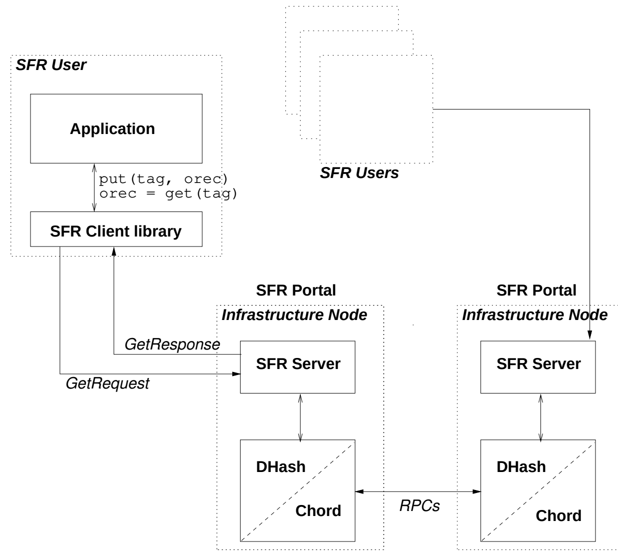


Figure 4: SFR implementation.

would have to do a separate SFR lookup, incurring additional latency. Our (currently unimplemented) solution is to allow the content provider to insert hints next to *local* URLs. These hints would indicate that the reference is local and would also contain a pathname.

Optimizations. As described earlier, `o-records`' `location` fields may contain DNS names, possibly introducing extra latency (since clients would have to do two sets of lookups, one for SFR and one for DNS). If reducing latency were paramount for the `o-record` owner, however, the owner might avoid this layer of indirection and instead rely on an external system that directly updates the IP addresses in batches of `o-records`.

Tools. We believe that any realistic deployment of SFR would necessarily be accompanied by new editing tools for content providers that either hide the actual references or else make them easier to work with. Although questions about how to build these tools are worthwhile, they are outside the scope of this paper.

5 Implementation

5.1 SFR Implementation

SFR portal nodes run a slightly modified version of MIT's DHash/Chord [8, 31] along with a separate SFR server module that uses DHash's API to store and retrieve `o-records` in response to client requests. Client applications interact with SFR by linking to the SFR client library, which communicates with a nearby SFR portal via a simple request/response protocol, as pictured in Figure 4. We have not yet implemented the SFR relay. The SFR client library exposes `put()` and `get()` methods to applications for storing and retrieving `o-records`. Both the SFR server and SFR client cache `o-records` according to the TTL field.

The SFR server has several purposes: it abstracts the underlying DHT for applications that use SFR; it exposes a narrow interface (so that SFR clients need not conform to the wider interfaces that DHTs sometimes require); and it serves as a marshal for client requests, allowing SFR to control its clients' interaction with the DHT and allowing administrators to extend the SFR server to implement other security and access control functions.

To achieve reference integrity through randomness, we modified DHash to enforce the relationship described in Section 3.3 between the reference, a salt and a public key. Like DHash, the SFR server and SFR client are written in C++, use the SFS toolkit [17] for asynchronous programming and cryptographic operations, and run on FreeBSD and Linux. Because of the simple network protocol between the SFR client and server, we anticipate that writing SFR clients in other languages and on other platforms will not be difficult.

The protocol has four messages: `GetRequest`, `GetResponse`, `PutRequest`, and `PutResponse`. The messages' contents (including items like the salt, the `o-record`, and the public key) are sent using type-length-value (TLV) encoding; both the client and server sign their messages. Applications must supply the public key of the SFR portal to the SFR client library (the converse is not necessary because the SFR portal does not need to identify its clients, except by the public key, which the client supplies). The implementation currently assumes a stream-oriented connection (which is certainly not optimal for performance), but it would not require much effort to move to an unreliable service like UDP.

5.2 Web-over-SFR Implementation

SFR clients are not yet embedded in Web browsers and so to prototype the Web-over-SFR, we use a Web proxy that simulates how a Web browser would interact with SFR if SFR were ubiquitous. The proxy is written in C++ and uses the SFS toolkit and SFR client library. The proxy's basic operation is translating URLs submitted by clients into SFR URLs. The proxy serves several functions: (1) it allows end-users to experience the latency associated with SFR as compared to DNS; (2) it allows us to dynamically populate SFR with the `o-record`s that would exist if the whole Web used SFR; and (3) it allows us to test the usability of semantic-free URLs.

In its usual mode, the proxy addresses (1) and (2). When a client browser requests a traditional URL, the proxy translates it into an SFR lookup by first hashing the URL and using that hash as the salt, and then hashing this salt together with the proxy's public key, thereby creating an `SFRTag` (as described in Section 3.3). The proxy then uses the SFR client to retrieve meta-data for this `SFRTag`. If the lookup is successful, the proxy uses the IP, port, and pathname information in the returned `o-record` to contact the actual Web server and then begins returning content to the client, thereby incurring the latency associated with an SFR lookup. If the lookup is unsuccessful, the proxy, besides returning content to the client, populates the SFR infrastructure on demand by constructing an appropriate `o-record` (based on a DNS lookup) and inserting it into the infrastructure.

This `o-record` contains a list of (IP address, port) pairs as well as a corresponding list of paths, a timestamp, and the TTL from DNS (different from the `o-record`'s TTL field, discussed in Section 3.1). The proxy stores these latter two items to obey DNS's semantics: if the proxy does an SFR lookup and the TTL has expired, the proxy executes another DNS request and inserts the updated `o-record` into the infrastructure.

The SFR Web proxy also directly accepts URLs of the form `http://0123aa.../optional_path` and treats the `0123aa` portion as an `SFRTag` (as we described in Section 4). Given Web pages with this type of SFR URL, we can test SFR's usability. In the future, we plan to have the proxy also rewrite traditional URLs in the Web pages that it returns to clients to make these URLs semantic-free, thereby permitting convenient usability tests.

6 Evaluation

We analyze SFR's performance using a combination of real-world data and simulation.

6.1 Latency Data

We deployed SFR nodes running DHash/Chord and the SFR portal software on the PlanetLab testbed [26]. The Chord ring uses approximately 130 physical hosts and 390 virtual nodes [31]. We also deployed our Web proxy at three different PlanetLab locations, and seven people (including the authors) used this proxy for days at a time over a one month period. When the proxy receives a URL, it creates two `SFRTags`—one corresponding to the hostname portion of the URL and the other corresponding to the entire URL—and then submits both to the embedded SFR client (which in turn contacts the SFR portal running on the local host). In order to permit a fair experiential comparison with DNS, the proxy returns content to the user as soon as the SFR client returns the `o-record` corresponding to the hostname digest. The SFR client cache (which obeys DNS TTLs in our implementation) is then equivalent to a DNS cache. If SFR were actually deployed, the number of `SFRTags` would be in between the number of hostnames and the number of distinct URLs on the Web: many `SFRTags` will certainly refer to directories under Web sites but not to individual Web pages.

Figure 5 compares the CDF of SFR's latency (as measured by the SFR portals) to a dataset for DNS that depicts a CDF of latency, as measured by a resolver at MIT. Only the SFR lookups that resulted in a Chord lookup are rep-

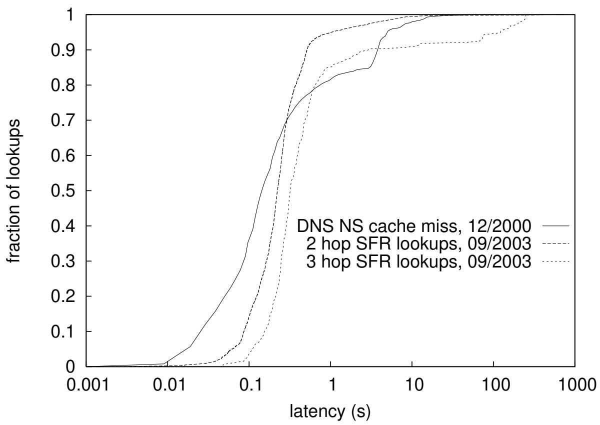


Figure 5: CDFs of SFR latency and DNS dataset

resented; the rest are satisfied via the SFR client's or the SFR portal's `o-record` cache. We use lookups from an eight-day period in September 2003; the depicted period occurred during the one month period mentioned above and after a bug fix that slightly improved latency. Because of the limited size of the PlanetLab Chord ring, aggressive caching of other virtual nodes' locations, and sharing of these caches among virtual nodes, 98% of the almost 15,000 lookups resolved in two Chord hops (the usual minimum); the rest required three hops. We show in simulation below that even in a large Chord ring, aggressive location caching results in two or three hops per lookup.

The DNS data comes from the work by Jung *et al.* [16] and depicts the end-to-end latency experienced by a resolver at MIT when NS record cache misses occurred. We do not incorporate A-record cache misses because doing so would unfairly count the many small requests for low-TTL A-records that are directed to CDNs, which move the name servers for popular content close to the client in many cases. We anticipate that if SFR were widely deployed, CDNs running over it would be able to implement similar optimizations. The DNS data is three years old and was collected at a single institution; hence, this comparison is meant to be suggestive, only, and not conclusive.

The feedback from our users is that perceived latency was generally indistinguishable from DNS, and Figure 5 supports this claim, suggesting that SFR's latency in the common case (two hops) is reasonably close to DNS's.

6.2 Simulation

We have just seen that on a testbed shared by hundreds of researchers, two and even three hop Chord lookups yield reasonable latencies. We now wish to confirm with simulations that—despite the $O(\log n)$ theoretical bound for number of lookups—two and three hop lookups will, in fact, be the norm when the hosts implementing the DHT do aggressive location caching.

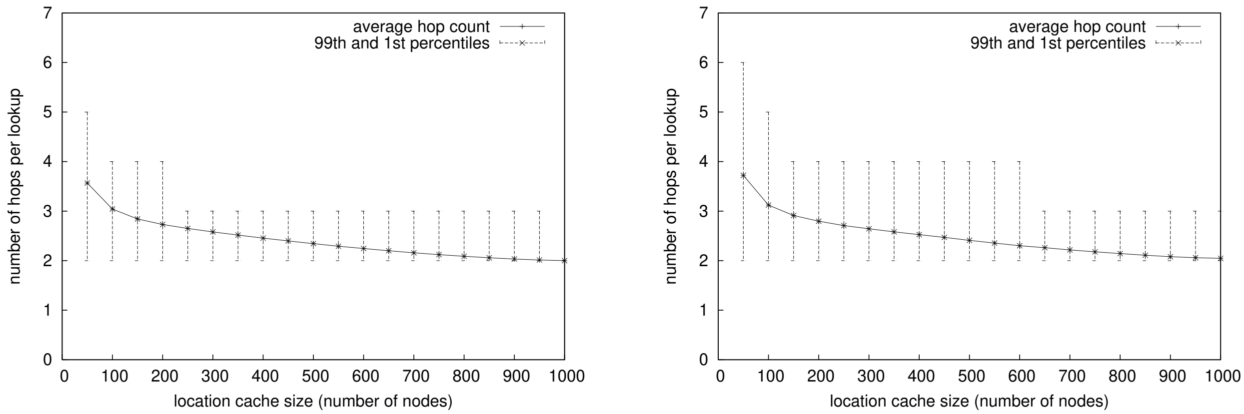
We used a modified version of the Chord simulator described by Stoica *et al.* [31] to gather trace-driven results. In the simulator, nodes add any node with which they communicate to the location cache. Eviction proceeds LRU, though a node's fingers will never be ejected. Because of Chord's routing, aggressive location caching causes nodes to accumulate relatively more information about nodes nearby in ID space.

To drive our simulations, we used two days of NLANR cache trace data [14], aggregating the separate caches' logs. Each URL in the aggregated trace causes a simulated SFR lookup of the URL's hash. (Hashing hostnames produced slightly better results, so we conservatively present the former.) To "warm up" the location cache, we ran the simulator on a day's worth of NLANR requests and then tabulated hop counts for the next 10^6 requests.

Figure 6(a) presents the results of this experiment for various location cache sizes and a 1,000 node Chord ring. Location caching reduces the number of hops to two or three because "being close counts": if the originating node, O, has not cached the target of a lookup, T, O is nonetheless likely to know about a node P near T, and P is likely to know about T. We believe that caching constant fractions of the Chord ring is reasonable because the number of nodes in a deployed SFR infrastructure would be bounded.

We must be careful, however. If the DHT's membership often changes, larger location caches could have more stale state and thus be detrimental. We expect, though, that as long as the membership changes relatively infrequently compared to the rate of requests, then failures will not much alter hop counts or even latencies. To see why, note that if a DHT node O attempts to communicate with a failed node, F, O will wait for a set length of time (500 ms in the simulation) before concluding that F is inaccessible. After this interval, O evicts F's entry from its cache and then tries to contact a different node. This permits lookups to make progress, even if the lookup path reaches a failed node [31]. If requests arrive frequently, then stale state will be corrected frequently and the number of timeouts will be relatively small; on *average*, therefore, the churn does not increase latency much. (This "cleaning-on-demand" is in addition to Chord's stabilization procedure, which also helps clear old state in the location cache.)

To examine failures experimentally, we first used the NLANR trace to warm up the cache with a million requests. (Although this warm-up period differs from the previous one, the effect is negligible in practice.) We then used interleaved Poisson processes: node deaths and births each occur on average once every 10 seconds, and, concurrently, 10,000 document lookups occur at an average rate of 20 per second, the approximate lookup rate in



(a) (b)

Figure 6: Simulated effect of location caching on hop count (a) without failures and (b) with node failures. 1000 nodes.

the NLANR trace. We believe that a failure in the infrastructure every 10 seconds is a significant over-estimate. Figure 6(b) shows the results: failures do not affect average hop counts and only slightly affect the 99th and 1st percentiles. Timeouts vary slightly with the location cache size but are never more frequent than an average of .04 timeouts per lookup, with a 99th percentile of two.

We conclude this section by noting that DNS's failure resilience depends on proper manual configuration of name servers and on zone transfers between primary and secondary servers. We do not envision the same degree of manual involvement in SFR's operation.

7 Alternatives and Related Work

In this section, we consider alternative designs and related work, first focusing on design decisions we made in the context of SFR and then discussing other proposals with similar goals.

7.1 SFR with More or Less

SFR−−. We considered a more minimal design, called SFR−−, that associates every `SFRTag` to an authoritative domain that hosts the actual `o-record`. In this model, looking up an `SFRTag` in the global SFR infrastructure returns only a pointer to an organization's resolver, and then this organization-specific resolver maps the `SFRTag` to the object's actual meta-data. This approach unfortunately requires each organization to host its own SFR service and each client to do an extra lookup (if caching fails), namely the one inside the organization.

However, there are several advantages to this approach, and they are instructive. First, SFR−− is analogous to the way DNS's top-level domain servers point to NS records, so SFR−− inherits the usual benefits of hierarchy (*e.g.*, fate sharing), but it does so without any structure built into the references themselves. Second, SFR−−'s global records point only to individual organizations and so would rarely change, and third, because SFR−− offloads many of the reference resolution problems to the individual organizations, it explicitly allows each organization to implement its own solutions to problems like object migration and replication.

SFR++. SFR can't directly handle massive replication because sending all locations to the client is unwieldy, and SFR itself doesn't have any application-independent way of selecting which locations are best for the client. A modified design, SFR++, would allow SFR to disambiguate between multiple locations based on *selector fields*. That is, content providers could associate several logical `o-records` to the same `SFRTag`; when a client does a lookup on a given `SFRTag`, the infrastructure could use a client attribute, such as IP address, as a selection mechanism for choosing from a set of `o-records` the one that corresponds to a location near the client.

The ability to disambiguate based on selector fields would also allow SFR to deal more gracefully with object *transformation*, when the object referred to by a given `SFRTag` splits into several component objects. Currently SFR uses redirection (see Section 4.1), which is not ideal; with SFR-embedded disambiguation, however, clients could submit, on a lookup, the `optional_path` component of an SFR URL in addition to the `SFRTag`. The responsible SFR node could then do longest prefix matches to track transformed objects, according to a table set by the content provider and stored alongside the `o-record`.

7.2 RRS: Other Approaches

The Globe literature [2, 32, 33] articulates the case for a single, general-purpose infrastructure for mapping persistent object identifiers to current locations. While they do not state that references should be inherently human-unfriendly, they do observe (1) that persistence implies that references cannot encode information about how they are resolved and (2) that human-level names should be strictly separated from identifiers. Globe's choice of a resolving substrate, however, differs from ours; in particular, the Globe location service relies on distributed trees overlaid on a static hierarchy of nodes (though, as in SFR, the identifiers themselves are not hierarchical nor is there any *a priori* difference among the hosts comprising the tree). SFR's approach to reference integrity, fate sharing, and latency differ from Globe's as well.

The URN community [2, 5, 9, 19, 28, 29] makes a case nearly identical to Globe's for persistent identifiers that identify individual objects; they propose a framework in which each application would have its own resolving infrastructure and its own namespace. In addition, the URN standards specify that references are to be human-unfriendly [29], but they neither specifically advocate that the infrastructure enforce randomness in the references nor do they propose a way to resolve these references.

O'Donnell articulates the need for a human-unfriendly namespace with persistent identifiers, and his vision is similar to ours [23, 24]. However, his numeric Open Network Handles would each exist in their own DNS domain underneath particular altruistic providers (*e.g.*, `h1282132.nicesponsor.org`); this approach contrasts with our claim that full location independence means not encoding any identifying information—not even about the provider responsible for the meta-data—into the reference itself.

If Open Network Handles were enhanced so the altruistic provider were removed from the URL, then all of these handles would exist in the same domain, and the challenge of routing in an unpartitioned namespace would arise, along with the other challenges we mention. This scheme would thus be functionally equivalent to SFR. (This assumes DNS were augmented with a record type that abstracted individual objects, otherwise these Handles could not provide location-independent, per-object references.)

As a final alternative, the Secure File System (SFS) [18] is an example of an existing system that relies on human-unfriendly identifiers with cryptographic guarantees. Although SFS references consist of a hostname, a hash of a public key, and a pathname, and are thereby tied to administrative domains, one could extend SFS to provide machine independent references by, for example, removing the hostname component and mapping the hash of the public key to a machine via a level of indirection like our `o-record`. At that point, SFS would either face similar challenges to the ones we have identified, or it would make use of SFR (though it would additionally have all of SFS's security benefits). However, this scheme could not provide any kind of object migration without redirects (implemented via symbolic links in SFS space), and the SFS literature has never articulated the need for persistent, location-independent identifiers.

7.3 Other Related Work

Frankston notes DNS's conflation of user-level names and references and also proposes a set of semantic-free references for the Web, though he does not detail a design [10]. The PURL project provides a layer of indirection via HTTP redirects to give location-independent, persistent URLs that may or may not contain semantics [27]. Phelps and Wilensky suggest a scheme for robust hyperlinks in which every document would have a unique signature, consisting of several words, and every referring hyperlink would embed the signature so that if a link were broken, a search engine could then find the document [25]. These schemes make use of, and are therefore constrained by, the existing Web infrastructure.

Digital Object Identifiers (DOIs) [13] are a URN implementation with persistent object identifiers in a managed but human-unfriendly namespace. DOIs are in use (including by ACM) and rely on the Handle System [6], an RRS that maps persistent identifiers to object meta-data using two levels of hierarchy.

The *i3* infrastructure envisions a widely deployed substrate for a general form of indirection [30]. This service indirects *routing* whereas SFR is an application-level layer of indirection for *naming*. Cox *et al.* describe an implementation of DNS in which they use Chord as a lookup mechanism for DNS A-records, thereby eliminating many administrative problems that result from the hierarchy in DNS [7]. They hash domain names into a flat namespace and use the original names both as identifiers and as a way of creating a public key hierarchy to authenticate a given A-record. They do not assume widespread caching of either the data being delivered (`o-records` in our case, RRsets in theirs) or of the other nodes in the Chord ring. Based on pessimistic assumptions about the infrastructure (ones we do not share because we think our system will be a managed service in which locations are cached), they conclude that the performance of DNS over Chord is unacceptable. They make no arguments in favor of an application-independent, semantic-free, general purpose referencing infrastructure and envision using current DNS names as references.

8 Conclusion

The goal of SFR is not to provide equivalent functionality to DNS, which ought to continue with its original purpose of hostname translation, but rather to provide a more attractive alternative for the subclass of applications, like referencing Web objects, that require an RRS.

In this paper, we have knowingly adopted an extreme view, namely that references should encode neither human readable semantics nor any other information about the referenced object. It is entirely possible, however, that the referencing system of the future will be somewhere in between DNS and SFR, either because human readability turns out to be critical or because a hierarchical resolving scheme that ties references to particular providers turns out to be the right economic model. For now, we simply observe that from a usability perspective, today's DNS and SFR each offer something the other does not. DNS makes composing and publicizing content easy while SFR attempts to achieve the full potential of the Web as a medium in which anyone can publish (even without controlling a domain), in which objects can freely migrate, and for which the infrastructure is simple, robust, and accessible.

Acknowledgments

We gratefully acknowledge: the anonymous reviewers, Brad Karp, David Andersen, Frans Kaashoek, Jaeyeon Jung, Nick Feamster, Maxwell Krohn, Bret Hull, Michael Afergan, Bob Briscoe, Karthik Lakshminarayanan, Sriram Ramabhadran, and our shepherd, Bill Weihl, who all read drafts and made many helpful suggestions; David Mazières, Bryan Ford, Karen Sollins, Jeffrey Considine, and Eddie Kohler, for useful conversations and ideas; Emil Sit, Benjie Chen, and Frank Dabek, for help with Chord; and Jaeyeon Jung for help with the DNS data. This research was conducted as part of the IRIS project (http://project-iris.net/), supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660.

References

[1] H. Balakrishnan, S. Shenker, and M. Walfish. Semantic-free referencing in linked distributed systems. In *2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, Feb. 2003.

[2] G. Ballintijn, M. van Steen, and A. S. Tanenbaum. Scalable user-friendly resource names. *IEEE Internet Computing*, 5(5):20–27, 2001.

[3] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's Internet. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[4] Computerworld. `http://www.computerworld.com/securitytopics/security/cybercrime/story/0,10801,82888,00.html`.

[5] D. Connolly. Naming and addressing: URIs, URLs, ... W3C Architecture Document.

[6] Corporation for National Research Initiatives. `http://www.handle.net/`.

[7] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a peer-to-peer lookup service. In *1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002.

[8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. 18th ACM Symposium on Operating Systems Principles*, Banff, Canada, Oct. 2001.

[9] L. Daigle, D. van Gulik, R. Iannella, and P. Faltstrom. URN namespace definition mechanisms, June 1999. RFC 2611.

[10] B. Frankston. DNS: A safe haven. `http://www.frankston.com/public/ESSAYS/DNSSafeHaven.asp`.

[11] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, 2004.

[12] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, March 2003.

[13] International DOI Foundation. `http://www.doi.org/`.

[14] IRCache. `http://www.ircache.net/`. NSF (grants NCR-9616602 and NCR-9521745), National Laboratory for Applied Network Research.

[15] Infrastructure for resilient Internet systems. `http://www.project-iris.net/`, 2002.

[16] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS peformance and the effectiveness of caching. *IEEE/ACM Trans. on Networking*, 10(5), Oct. 2002.

[17] D. Mazières. A toolkit for user-level file systems. In *Proceedings of the 2001 USENIX Technical Conference*, pages 261–274, Boston, MA, June 2001.

[18] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proc. 17th ACM Symposium on Operating Systems Principles*, pages 124–139, Kiawah Island, SC, Dec. 1999.

[19] R. Moats. URN syntax, May 1997. RFC 2141.

[20] P. Mockapetris. Domain Names – Concepts and Facilities, Nov 1987. RFC 1034.

[21] P. V. Mockapetris and K. J. Dunlap. Development of the Domain Name System. In *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1998.

[22] M. Mueller. *Ruling the Root: Internet Governance and the Taming of Cyberspace*. MIT Press, Cambridge, MA, May 2002.

[23] M. O'Donnell. Open network handles implemented in DNS, Sep. 2002. Internet Draft, draft-odonnell-onhs-imp-dns-00.txt.

[24] M. O'Donnell. A proposal to separate Internet handles from names. `http://people.cs.uchicago.edu/~odonnell/Citizen/Network_Identifiers/`, Feb 2003. submitted for publication.

[25] T. A. Phelps and R. Wilensky. Robust hyperlinks: Cheap, everywhere, now. In *Proceedings of Digital Documents and Electronic Publishing (DDEP00)*, Munich, Germany, Sept 2000.

[26] PlanetLab. `http://www.planet-lab.org`.

[27] K. Shafer, S. Weibel, E. Jul, and J. Fausey. Introduction to persistent uniform resource locators. `http://purl.oclc.org/docs/inet96.html`. OCLC Online Computer Library Center, Inc., 6565 Frantz Road, Dublin, Ohio 43017-3395.

[28] K. Sollins. Architectural principles of uniform resource name resolution, Jan 1998. RFC 2276.

[29] K. Sollins and L. Masinter. Functional requirements for uniform resource names, Dec 1994. RFC 1737.

[30] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[31] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[32] M. van Steen and G. Ballintijn. Achieving scalability in hierarchical location services. In *Proc. 26th International Computer Software and Applications Conference*, Oxford, UK, Aug. 2002.

[33] M. van Steen, F. J. Hauck, P. Homburg, and A. S. Tanenbaum. Locating objects in wide-area systems. *IEEE Communications Magazine*, 36(1):104–109, Jan. 1998.