Resource Cataloging and Distribution System*

Keith Moore, Shirley Browne, Jason Cox, and Jonathan Gettler

University of Tennessee Technical Report

January 1997

Abstract

We describe an architecture for cataloging the characteristics of Internet-accessible resources, for replicating such resources to improve their accessibility, and for registering the current locations of the resources so replicated. Message digests and public-key authentication are used to ensure the integrity of the files provided to users. The service is designed to provide increased functionality with only minimal changes to either a client or a server. Resources can be named either by URNs or by existing URLs, and either type of resource name can be resolved to a description and ultimately to a set of locations from which the resource can be retrieved.

*This material is based upon work supported by the U.S. Army Research Office under grant number DAAH04-95-1-0595

1 Introduction

Almost any user of the World Wide Web will be familiar with the following problems:

- Frequently, the file server mentioned in a particular URL is down, unreachable, or busy.
- While additional copies of the file named by that URL may exist (on so-called mirror sites), there is no mechanism for finding them, no way to know whether such copies are current, and no means of ensuring that the mirrored copy has not been altered.
- URLs become stale, that is, a URL which once pointed to a particular file no longer points to any version of that file.
- Search services are often out-of-date due to the sheer size of the net and the necessity to periodically poll each server to see whether its files have changed.
- Search services that return URLs often return duplicate hits because the same file is accessible by multiple URLs. Search services do attempt to eliminate duplicates, but often with the result of eliminating the more desirable URLs from the point of view of proximity.
- There is a need to be able to label resources according to certain criteria, and for the user to be able to examine such labels before attempting to access the resource.
- Given the ease by which many file servers (whether primary servers or mirror sites) may be compromised, there is a need for a service that allows the integrity and authenticity of a file to be checked.

The Internet Architecture Board (IAB) held a workshop on Internet Information Infrastructure in October of 1994 which led to the following recommendations [7]:

- increased focus on a general caching and replication architecture,
- rapid deployment of name resolution services,
- articulation of a common security architecture for information applications.

The workshop report pointed out performance, reliability, and scaling problems with current Internet information services similar to those we have listed above. In general, the report recommended inclusion of a "wholesale", or "middleware" layer in the information architecture that would sit between the lower "raw materials" layer and upper "retail" layer, with standard interfaces between the layers.

We propose an architecture for a system, called the Resource Cataloging and Distribution System (RCDS) which addresses the problems described above. We hope that our work will constitute a substantial contribution toward realization of the middleware layer described in the IAB report and toward solving the scaling problems described in [3].

1.1 Design Goals

The goals of RCDS include:

- It must be easy to deploy in the current Internet.
- It must be highly reliable and fault-tolerant.
- It must use the network efficiently.
- It must provide adequate security, both to ensure that its authentication/integrity assurance services are trustworthy, and to thwart denial-of-service attacks.

- It must be flexible and general so that it can incorporate existing network protocols as well as evolve to meet future needs.
- It must be scalable to several orders of magnitude beyond the current resource base size without fundamental changes in the structure of resource names, or in the means by which a resource name is resolved to the network location of a server that provides the resource.

These goals have certain implications for our design:

- The flexibility goal dictates that the system should not assume present-day notions of roles such as "author", "publisher", or "editor" in determining who can supply information about a resource. It also compels us to accommodate multiple data models for use by catalog records, as well as a variety of cryptographic authentication and integrity checking algorithms. Likewise, the service should accomodate several different protocols for accessing and retrieving resources, including those that will be defined in the future as well as those in use today.
- The scalability, reliability, and network efficiency goals dictate that the system maintain replicated copies of the information which it provides and keep those copies in reasonable synchronization.
- The goal of ease of deployment implies that the service should augment, rather than replace, the current World Wide Web infrastructure. Furthermore, it should be easy for authors or publishers to set up and maintain their RCDS servers for the resources that they own.

1.2 Issues

The following issues must be considered:

- *Transition issues.* In general, it is difficult to build new infrastructure in the Internet, because the infrastructure must be in place before its costs can be justified by its benefits.
- *Security and Survivability.* It is difficult to provide network services that are immune to hostile attack or accidental damage. Doing so requires careful attention to the operation of the server machines, the ability to detect possible problems, and sufficient logging to analyze security breaches and recover from file loss or corruption.
- *Consistency.* Consistency models for replicated data range from strict one-copy serializability [1] to a best-effort, eventual convergence model. Because of the high overhead of maintaining strict consistency, most distributed databases and file replication schemes deployed on the Internet, such as the Domain Name System, use looser consistency models. However, some applications, for example an automatic program builder that retrieves software components from distributed sites and constructs an executable suitable for the user's platform, may require stricter consistency.
- *DNS.* There are both advantages and disadvantages to using the Domain Name System as a component of a resource cataloging system. On the positive side, DNS is widely deployed and implementations are already available for most platforms. On the negative side, DNS is known to be insecure against attack, to have problems with stale data, to have difficulty tolerating domains with a large fan-out (such as the .COM domain), and to be easy to misconfigure. All but the last of these problems are being addressed by IETF working groups, and similar issues would be encountered in any other widely distributed database.

The assumed significance of transition issues on the success of the project influenced our design in the following ways: (a) we allow ordinary URLs as one kind of resource name, (b) we use existing file servers and file access protocols, and (c) we employ DNS as a component of the system rather than building a new distributed database from the ground up.

The need for reliable authentication and integrity assurances, coupled with the difficulty of providing secure servers, prompted us to use end-to-end (between information provider and user) authentication, consisting of public-key signatures and cryptographically signed certificates, rather than depending on the security of resource catalog servers or file servers (though reasonable security for these is still required to thwart denial-of-service attacks).

We have implemented a flexible consistency model which combines a convergence based peer update protocol for resource locations with a stricter token-based protocol for catalog information.

Finally, some of the inherent limitations of DNS and the desire to separate administration of "naming authority" names from administration of resource names for a particular naming authority, led us to use DNS only as a means to identify one or more resource catalog servers for a particular resource naming authority, rather than to provide resource location or catalog information directly through DNS.

2 Description of RCDS

The Resource Cataloging and Distribution System (RCDS) consists of the following components:

- *Clients*, which are the consumers of the resources provided by the system. One kind of RCDS client is an ordinary WWW browser with slight modifications to make use of the resolution system. Unmodified WWW browsers can also access RCDS through the use of a RCDS-aware proxy server.
- *File servers*, which provide access to the files themselves. These can be ordinary general-purpose file servers that use http, ftp, nntp, imap, NFS, or some other file access protocol, or servers that use a protocol designed for some special purpose (such as those designed for the real-time transmission of audio or video), or even multicast transmission sources.
- *Resource catalog servers*, which maintain information about the characteristics of network-accessible resources and accept queries about the characteristics of such resources from clients.
- *Location servers*, which maintain information about the locations of network-accessible resources, and accept queries for location data from clients.
- *Collection managers*. The collection of files on a file server is maintained by a collections manager, which learns about newly published files and determines when a file server should acquire replicas of new files and reap old ones, according to site-specified criteria. The collections manager is also responsible for actually acquiring and deleting the chosen files. Finally, when a new file is added to the collection or an old one removed, the collections manager informs the location servers about changes in file availability.
- *Publication tools*, which accept new files and descriptions from content providers (e.g. authors), and inject them into the system.
- *SONAR*, which allows a choice of locations to be made by an RCDS client. Given several alternative locations from which to access a resource, an application program can query a nearby SONAR server to determine which locations are closer than others. The SONAR server will return, for several of those locations, a metric which provides a rough indication of the proximity of that location to the SONAR server. Using those metrics, the application can then choose a single location from those returned from which to access the resource.

2.1 Resource names

RCDS uses three kinds of resource names: URLs, URNs, and LIFNs. Web users will already be familiar with the syntax of URLs and how they are used. URNs or Uniform Resource Names are similar in appearance to to URLs but are designed to be stable, persistent, and independent of any access protocol or location. RCDS can serve as a *resolution system* for both URNs and URLs.

2.1.1 URNs and LIFNs

URNs (Uniform Resource Names) are used to provide stable names for resources whose characteristics may vary over time. For instance, a URN may be used as a stable reference to a web page. The web page can then move, be replicated, or change its contents and still remain accessible through the same URN. In contrast to URLs which have wired-in location information, the location information and other characteristics of a URN are provided by external *resolution servers*.

A LIFN (Location-Independent File Name) is similar to a URN in that it is a stable name and that it can be resolved to find locations of a resource that it names. However, unlike a URN, a LIFN is constrained to name a specific fixed instance of a resource. All copies of a file named by a LIFN are byte-for-byte identical. The meaning of a LIFN also does not change over time. Once a LIFN is used to refer to a particular file, it must always refer to that same sequence of octets. [1] A URN is associated with a *description* of the resource it names, while a LIFN is associated with with one or more *locations* of identical copies of that resource.

The description associated with a URN normally contains one or more LIFNs, which name particular instances of that resource, and describes the characteristics of each instance. For example, if the resource named by a particular URN exists in several different data formats (e.g. plain text, PostScript, PDF, HTML), the description for that URN will list each of these, along with a LIFN for each specific instance. Similarly, if the resource associated with a URN has changed over time, and multiple versions of the resource are accessible, the description of that resource would contain a list of the current and previous versions along with the LIFNs for each. Because the LIFN can then be used to find the current locations of a resource, it serves as a "link" or "file handle" from the description of a resource instance to the list of its current locations.

LIFNs have several purposes in RCDS:

1. A LIFN serves as a link between a catalog record that describes a resource and the locations of a particular instance of that resource.

2. LIFNs are used by replication daemons (mirroring tools) which create new replicas by copying files across a network. The replication daemons use LIFNs to refer to the files being replicated, so that there is no ambiguity about which version of a file is being copied. This also allows the locations of all replicas created by such daemons to be associated with the same identifier.

3. LIFNs are intended to be used as cache validators. If a client determines that a user request can be satisfied by the file named by a particular LIFN, and the client finds a cached copy of the file named by that LIFN, (and the client trusts the integrity of the cache), the client can use the cached copy. In this use LIFNs are similar to the "strong entity tags" of the HTTP/1.1 protocol, and can be used in HTTP's `ETag:` entity-header field.

The distinction between URNs and LIFNs was crafted for several reasons:

- Location data and descriptions are maintained by different parties. The description of a resource will normally be maintained by its author, publisher, editor, or reviewers, while the location data will be maintained by the managers of specific file servers.

- If a resource suddenly becomes of interest to a large population, the set of replicated copies of that resource may need to change quickly according to demand. Under such conditions, the location data for that resource may need to change much more quickly than the description of the resource itself.

- The location directory (accessed by LIFN) and the description server (accessed by URN) have different needs for consistency across replicas. There is little need to maintain a consistent list of locations across the set of replicated location servers. On the other hand, it can be very important to have an up-to-date description of a resource and for the replicated copies of that description to be consistent with one another.

[1] Although LIFNs were intended to refer to files, it should also be possible to use LIFNs to refer to replicated services that provide the same results at different locations, but that change over time. Such an extension of LIFNs is a subject for future work.

- The portions of RCDS responsible for replicating files and keeping track of their locations need an unambiguous name for a particular instance of a resource, to avoid confusing it with other instances of the same resource.
- If all instances of a file associated with a LIFN are identical, the client's choice of which instance of a resource to access (data type, version, etc) may be cleanly separated from its choice of which location to use when accessing the resource. The former choice can then be made on the basis of browser capabilities, user requirements, etc., while the latter choice can be based on (say) proximity estimates.

2.1.2 Format of URNs and LIFNs

NOTE: The format of RCDS URNs and LIFNs described here is based on recent discussions in the Internet Enginering Task Force and elsewhere, and is subject to change until the standards are finalized. RCDS can thus serve as an early URN testbed, and will migrate to whatever URN format and resolution schemes are adopted as Internet standards.

A URN consists of three parts:

1. The fixed prefix string `URN:`.
2. A *namespace identifier* (NSI), which identifies the format of the remaining portion of the URN.
3. A *namespace specific suffix* (NSS), which is an identifier assigned according to the rules for that particular name space.

The NSS will usually be subdivided into a *naming authority*, and a string which is assigned by that naming authority, which may itself be subdivided if further delegation is needed. So `URN:inet:foo.bar:mumblefrotz` would be a URN that was assigned by the naming authority `foo.bar`.

The location of the naming authority within the NSS is not fixed; rather, it is a characteristic of the name space. This allows URNs to serve as an "umbrella" for other naming schemes (e.g. ISBNs, SGML Formal Public Identifiers, Usenet Message-IDs) that have a variety of structures.

For RCDS URNs, the name space identifier will normally be `inet` (though any URN can be used) and the naming authority will be an Internet domain name. A LIFN is a URN from the name space `lifn`; the naming authority portion of a LIFN will also be an Internet domain name. While any Internet domain name could potentially be used, domain names used for URNs and LIFNs should be chosen to allow them to be persistent for the useful life of the resource.

2.1.3 Resolution of URNs and LIFNs

The structure of URNs provides just enough of a toe-hold to facilitate scalable, distributed resolution. Resolution is a process by which a client may access the resource named by a URN. It works as follows:

1. The client queries a well-known registry for information about the namespace identifier. The information returned will either indicate services and locations for all URNs with that identifier, or it will contain instructions, specific to that name space, which indicate the location of the naming authority within the NSS, and where to find the registries for that naming authority.
2. In the latter case, the naming authority is extracted from the URN and one of the registries for that naming authority is queried.
3. Each registry queried returns either (a) referral instructions for further queries (for a narrower portion of the name space) or (b) a list of services, locations of those services, and protocols which may be used when communicating with those services.
4. When the latter is found, the resolution process stops, and the client chooses one of the available services and locations. Among the services provided by RCDS are: URN-to-catalog record mapping and LIFN-to-location mapping.

The resolution process allows for mutiple servers at each registry, so it is both scalable (allowing the load to be split across multiple servers) and fault-tolerant (if a query fails at one server, the same query may be submitted to a different server).

RCDS clients may also be configured to consult "proxy" resolution servers (which perform queries on behalf of clients and cache results) as well as "fallback" resolution servers (which can be consulted when there are no "official" servers for a domain or when the "official" servers do not respond.)

2.2 Publication and Distribution

Figure 1 illustrates how files are published in RCDS.

1. An author submits a file to RCDS using a publication tool. If this is a new file, a new description (containing catalog information) of that file is created and a new URN is assigned; otherwise, the description of the old URN is updated to reflect the new version of the file. A LIFN is assigned to the new file, and this LIFN is included in the description of that file. The part of the description containing the LIFN and file fingerprint (and perhaps other parts of it) is cryptographically signed by the author using the publication tool.
2. The publication tool deposits a copy of the file on a file server, and a copy of the description on any resource catalog server associated with the URN's domain. [2] The publication tool also sends a copy of the description of the new file to interested parties, which might include file servers and search services.
3. The resource catalog server updates its peers with the new description.
4. The file server informs any location server associated with the LIFN's domain, that it has a copy of the file with that particular LIFN.
5. As other file servers find out about the existence of the new file, their collections managers decide whether to acquire it. When a file server acquires the new file and makes it accessible, it informs a location server about it.
6. The location servers propagate new file location information to one another.

2.3 Access and Retrieval

Figure 2 illustrates how files are accessed or retrieved in RCDS.

1. A user acquires a URN of a resource that seems to suit his needs from a search service, hypertext link, or other means. This URN is resolved using DNS (see below) to find the network addresses of one or more resource catalog servers. One of those servers is selected by the client, perhaps based on network proximity estimates.
2. The resource catalog server is queried for the description of the resource named by the URN. This description may itself contain sub-descriptions for each of several versions of a resource (which might vary according to time created, medium, language, etc.). Each of these sub-descriptions will contain a LIFN for that particular version. The client selects a particular LIFN from those available.
3. The client resolves the LIFN using DNS to find the network addresses of one or more location servers. One of those location servers is then queried for locations of the file named by that LIFN.
4. The location server returns one or more URLs at which the file can be obtained.

[2] If this is a modification to the existing URN, the publication tool also ensures that that particular catalog server has a current copy of the description. This discipline is followed any time a catalog server is updated, to ensure single-copy serialization of resource descriptions.

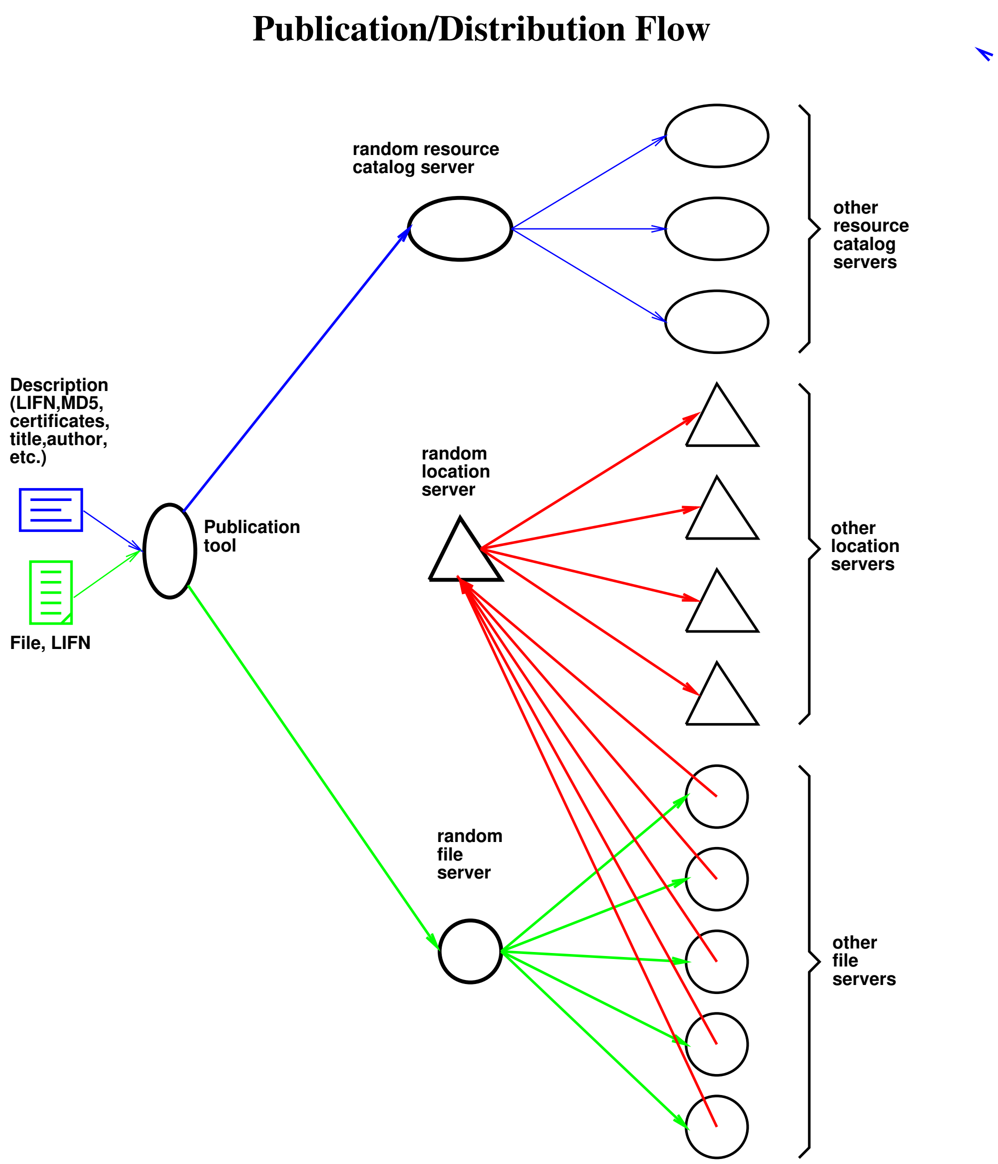

Figure 1: Publication of files in RCDS.

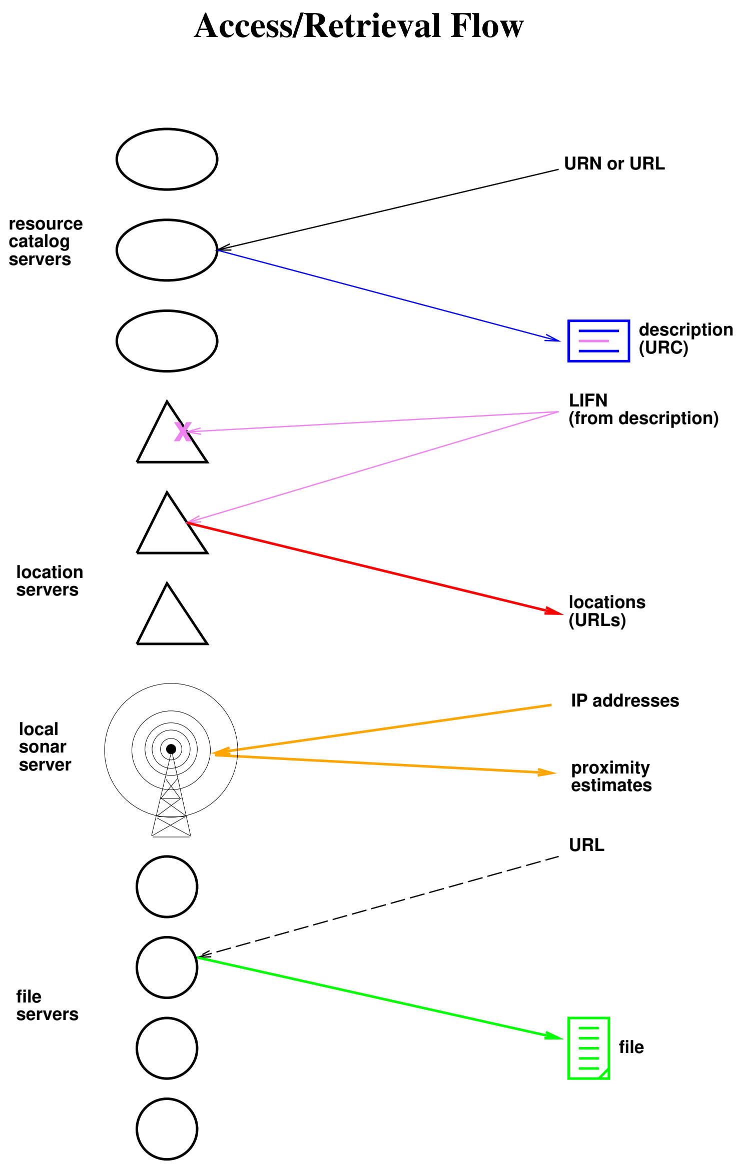

Figure 2: Accessing a file via RCDS.

5. The client chooses one of those file servers (again, perhaps based on network proximity estimates) and fetches the file from that server.

The interaction with RCDS may be accomplished either directly by a client, or via a proxy server which communicates with the client via HTTP. The proxy arrangement is shown in Figure 3.

3 Protocols

Because an understanding of some of the protocol details is important to understand how well RCDS achieves it goals, this section outlines important aspects of the protocols used by the current prototype.

RCDS currently uses a lightweight query-response protocol based on Sun's Open Network Computing Remote Procedure Call technology. Either UDP datagrams or TCP streams may be used. Unlike normal RPC applications which use a separate binding protocol to associate an RPC function to a TCP or UDP port, RCDS requests are sent to a "well-known port" on the server machine, cutting the network overhead in half.

There are currently five function calls:

1. `update_name` adds zero or more *assertions* and/or *certificates* to the catalog record for a URN or URL.
2. `update_lifn` adds a resource location (URL) to the list of locations associated with a LIFN.
3. `query_name` allows a client to obtain the catalog record (or portions thereof) associated with a URN or URL.
4. `query_lifn` allows a client to obtain the current list of resource locations for a particular LIFN.
5. `create_uri` allows a client to request that the server allocate a unique URN or LIFN in a particular domain. This is used by publication tools.

Although the catalog records and the locations are maintained separately, the server optimizes for the common case where a catalog record contains a LIFN whose locations are kept on the same server. In this case, the URLs associated with that LIFN are returned in the same response that contains the catalog record, space permitting.

For the update calls, authentication is accomplished by the use of a secret shared by client and server. The request, along with the shared secret and a timestamp, is used to calculate a 128-bit digest using a variant of the "keyed MD5" algorithm. This digest is transmitted by the client along with the request and the timestamp, but omitting the shared secret. The server computes the same digest using its copy of the shared secret, and compares the result it obtained with the digest included in the request. Only if the result matches is the client considered to be authenticated. The client must still have appropriate permissions to perform the request.

Replay attacks are thwarted (while allowing for duplicated UDP datagrams) as follows: the server keeps a copy of the last request id and the last result of any update call from a particular client. If the last request was repeated, the server repeats the response obtained from the previous call (which consists of a single integer indicating success or failure), without modifying the database. The client must increase the request id on each call; if the request id from a particular client is less than the previous request id, the request is considered to be a delayed duplicate and ignored.

Catalog records used by the `update_name` and `query_name` functions are composed of *assertions* and *certificates*. An assertion is essentially of the form "A states that, as of time T_a, the attribute named N of the resource named U had value V, and that this value is expected to be valid until time T_e." A certificate is essentially of them form "C warrants that, as of time T_c, assertions $A_1, A_2, ..., A_n$" are valid. A certificate also contains C's cryptographic signature, computed over T_c and the contents of assertions $A_1, A_2, ..., A_n$.

Location records associated with a LIFN (and used by the `update_lifn` and `query_lifn` functions) consist of a URL, a cache retention time-to-live T_r, and an expiration date T_x. The cache retention time-to-live is a contract by the file server that supplied the binding, that it will notify the server at least T_r seconds

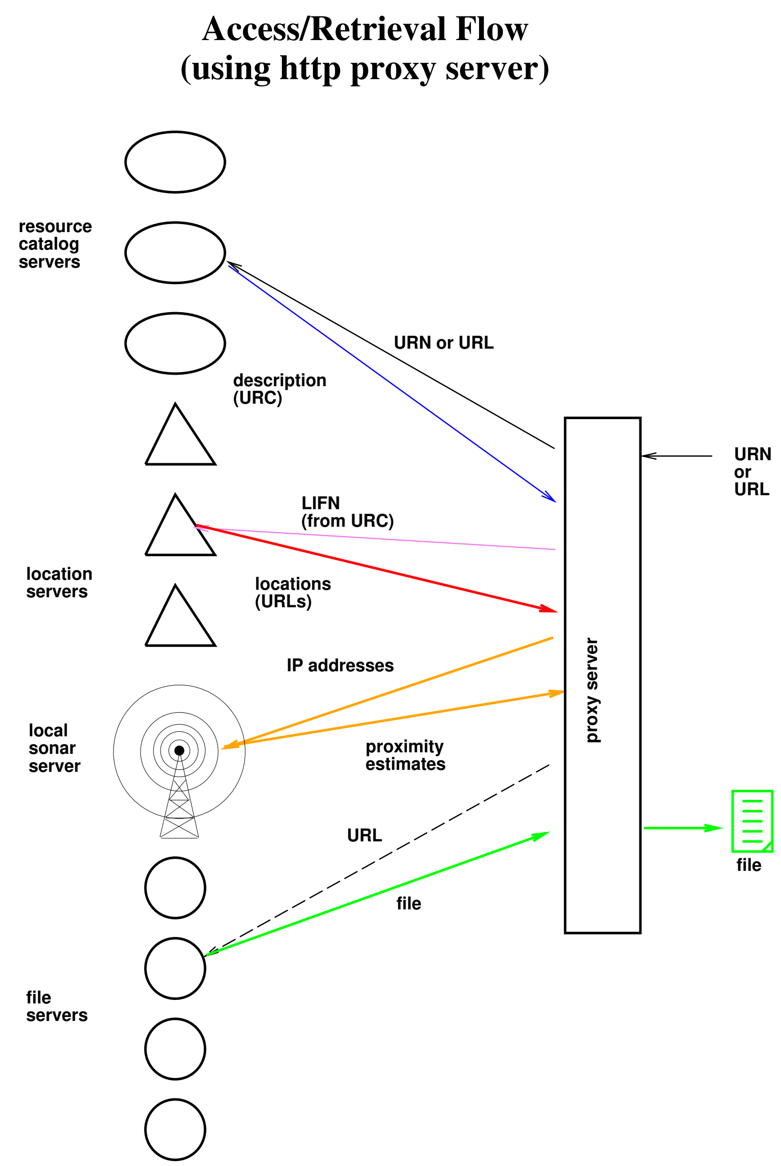

Figure 3: Accessing a file via RCDS, using a proxy server.

before making that file inaccessible to clients. Clients and caches should not use that LIFN to URL binding after T_r has expired. [3] The expiration date is an indication that

4 How RCDS Achieves its Goals

4.1 Ease of Deployment

Deployment of RCDS requires no new infrastructure other than that which can be provided directly by existing publishers. Clients, information providers, and mirror servers can each begin supporting RCDS independently of one another, except that "official" mirror servers need to obtain credentials in order to add location information to the resource owner's RCDS server.

Clients. A user must have an RCDS aware client in order to benefit directly from RCDS. Such a client may be either a web browser which has RCDS support built-in, or a web proxy which serves as an intermediary between the client and the rest of the Web, using RCDS whenever available. *Information providers.* In order to support RCDS, an information provider must install (preferably multiple) RCDS servers, register the locations of those servers in the Domain Name System (DNS), and update those servers when resources are changed or new locations are added. The RCDS and DNS servers are freely available. The publisher must also arrange to update the RCDS servers when a resource is changed. *Mirror servers.* Many resources today are replicated, or mirrored, from their original server. RCDS clients can find such replicas automatically if the original server supports RCDS, and the mirror server updates the appropriate RCDS server whenever it creates or deletes a replica. We have modified some commonly-used mirroring tools to perform this function, and we are developing a new high-performance replication tool. Mirror servers do need permission (and authentication credentials) to add their locations to an RCDS server.

4.2 Reliability and Fault-tolerance

Reliability is achieved by robust construction of the components and by allowing redundancy at every step. Multiple DNS servers may exist for a particular domain, multiple RCDS servers for any portion of URI-space may be registered in DNS, and RCDS servers may list multiple locations for a particular resource. Fault-tolerance is achieved by having clients attempt to reach multiple servers before declaring failure.

4.3 Efficient use of the network

RCDS promotes efficient use of the network by providing a lightweight protocol for queries and updates. In most cases, the resolution process is expected to cost one extra long-distance round-trip (to an RCDS server), and one extra local round-trip (to a SONAR server), as compared to the name-to-address lookup for a URL. The benefit is that the client can then choose to access the resource from a nearby server (rather than the one explicitly listed in a URL); the client can also avoid fetching the resource at all if it can tell by the catalog record that it is not needed, or (using the LIFN as a cache validator) that a locally-cached copy exists. Finally, because RCDS allows listing of multiple URLs for a resource, it allows the client to choose not only a nearby server, but also the best access protocol that it supports. RCDS thus provides a means to transition from ftp and http to more efficient protocols; for instance, streaming protocols for real-time audio and video, or multicast-based protocols for information which is transmitted simultaneously to many users.

4.4 Security

RCDS servers are associated (via the URI resolution process) with a particular subset of URI-space; it is assumed that the official RCDS servers for a resource are maintained or authorized by the owner of that resource. The owner or an authorized party must therefore provide permission and authentication

[3] Note that T_r for a LIFN to URL binding is no indication of the expected lifetime of the resource itself, and is only a lower bound of the expected time during which the resource is expected to be available at that location.

credentials to a party before it can add assertions, certificates, or locations to an RCDS database. This provides a mechanism to ensure that only authorized catalog information or replicas are listed in the official servers. Nothing prevents an unauthorized third party from establishing its own RCDS servers, but (barring attack of the resolution system) those servers must be explicitly configured by the user – they will not be found by a normal URI resolution process.

The authentication required to update an RCDS server does **not** ensure the authenticity or integrity of a resource listed by that server; it is intended only to provide some protection against denial-of-service attacks. RCDS provides end-to-end authenticity and integrity assurance for ordinary files through the use of message digests such as SHA or MD5, and public-key signatures for assertions that a particular message digest represents a particular version of a file.

Public keys are of limited utility without a means of key certification. It may be possible to use RCDS to as a repository for key certificates – signed assertions from a well-known party that a particular public key is associated with a particular RCDS asserter or certifier. The difficulty is not with RCDS itself, but in finding a trusted certificate authority (or chain of intermediaries) that can verify the asserter's public key. While there appears to be no general solution to this problem (i.e. no single certificate hierarchy) that can be trusted for every authentication need, it is often possible to establish a hierarchy or "web of trust" for specific purposes.

4.5 Flexibility

RCDS provides flexibility by having few wired-in assumptions about the structure of catalog information. No particular data model is assumed, and any party can (subject to permissions) potentially add any kind of assertion. Catalog information stored in an RCDS server is currently assumed to be "flat" lists of (attribute, value) pairs. Clients can retrieve portions of the catalog information for a resource by the attribute's full name or by the prefix of a name.

Likewise, RCDS catalog servers know nothing about particular message digest algorithms or signature algorithms. This allows RCDS to work with multiple existing public-key authentication methods, multiple signature formats, and a variety of key certificate formats. Experience with several other protocols (X.400, X.500, MIME) indicates that it is difficult to retro-fit security into a protocol after it is deployed, especially if there are multiple ways in which a protocol element may be represented. RCDS therefore supports cryptographic authentication without specifying any particular algorithm for signing certificates. In particular, it **does not specify** the representation of a set of assertions before signing. Such representation is bound to the signature algorithm identifier. In general, RCDS catalog servers do not interpret assertions or certificates. They merely serve as repositories at which they can be stored and retrieved.

4.6 Scalability

Scalability is provided by allowing multiple instances of any particular server (to distribute the load), and by minimizing the overhead necessary to propagate updates between RCDS servers. In particular, there is no requirement that all RCDS servers for a particular portion of LIFN-space contain the same set of locations, so long as a client can query multiple RCDS servers until it finds an accessible location for the desired resource. Location updates can thus be submitted to any location server associated with the LIFN's domain, and propagated to the others without needing a commit protocol. Similarly, updates to catalog records can be submitted to any catalog server for the URN's domain, though the asserter must first update that catalog server if it has stale data. In effect, updates to catalog records use a token-based algorithm, with the asserter obtaining the token and any accompanying updates before applying the new update.

Scalability of RCDS is also enhanced by limiting its scope (for instance, it does not do searching) and allowing it to be optimized for a small number of simple functions.

5 Current Implementation

We have implemented the catalog and location server in a single piece of software called **rc_server**. The server replicates its database using a simple replica updating scheme based on journal files. A more sophisticated replication scheme is needed. The catalog servers currently use shared secrets for symmetric authentication of udpates; public key authentication would be better.

We have modified the popular **ftp-mirror** replication tool to update a RCDS location server when a file is mirrored. We are developing a high-performance bulk file transfer tool called **blitzfer** which uses multiplexing and compression and supports checkpoint and restart.

We have written a publication tool that extracts a resource description from the results of filling out an HTML form, signs the description with RSA, and uploads the resource files to a designated file server.

We have implemented an HTTP proxy server that acts as a RCDS client. We have modified an older version of Mosaic to understand RCDS. We will modify the new redesigned Mosaic to be an RCDS client as soon as it becomes available. We are also looking at the possibility of an Active X plug-in for use with Microsoft Explorer.

Our initial implementation of SONAR sends ICMP echo requests to each of the addresses for which a proximity measure is desired, and uses the time between when the request is sent, and when an ICMP echo reply is received, as its metric for evaluating relative proximity. Round-trip-time estimates obtained using ICMP are cached for a certain amount of time (currently several hours) so that subsequent SONAR queries within that time do not result in generation of additional ICMP requests. Any ICMP "unreachable" replies are also noted, though these are cached for a smaller amount of time. We are investigating alternative algorithms for measuring proximity that place less load on the network.

A testbed for RCDS has been established at five geographically dispersed US sites (University of Tennessee, Syracuse, Rice, CalTech, and Argonne). This testbed serves as a platform to stress-test RCDS implementations and to provide a demonstration of RCDS technology. The RCDS testbed servers are being used to keep track of widely-mirrored network resources, including the Linux operating system and related software, the Internet RFC and Internet-Drafts repositories, the netlib software repository, and others. This provides a means to test the use of RCDS with resources that have large numbers of replicas.

6 Related Work

The Harvest caching architecture, in which individual caches can be interconnected hierarchically to mirror an internetwork's topology, is described in [5]. There is an effort underway, headed by Hans-Werner Braun and Kim Claffy at the San Diego Supercomputer Center, to put Harvest caches into place at high-level parts of the U.S. research and educational networking infrastructure, for the purpose of reducing backbone network traffic [8].

Server-based replication of objects is argued for in [2] and [6]. Both papers argue that network traffic and server load could be reduced considerably by having servers disseminate their most popular documents on servers closer to clients. Both also point out the need for integration of producer-based dissemination and consumer-based caching of documents, with caching being the most effective way to reduce latency. Furthermore, both papers point out the need for naming conventions and name resolution protocols that provide location and replication transparency.

The use of a wide-area file system for storing and retrieving WWW documents is proposed in [10]. Wide-area file system features of location transparency, access control lists, authentication, client caching, data replication, and file migration are suggested for improving performance, decreasing server and network load, and increasing security. Based on their analyses of WWW client and server traces, the authors recommend the use of *predictive* document repositories rather than *reactive* document caches.

The Internet Engineering Task Force (IETF) has formed a working group to standardize formats for Uniform Resource Names (URNs) [9] and protocols for URN resolution.

7 Future Work

A subject for future work is extending the use LIFNs to name services in addition to files. An interesting example of a network service is the NetSolve system which accepts linear algebra problem descriptions from users, computes a solution, and returns the results to the user [4]. Not all NetSolve computational servers solve all problems in all precisions. The computational servers also have different machine architectures and performance characteristics. Ideally one would like to be able to determine, for a given problem and input characterization, a equivalence relation on the set of available computational servers, where the servers in a given equivalence class provide identical results for the given problem. The idea would be to name each equivalence class with a LIFN and to resolve each LIFN to the set of locations for the servers. Computational servers could then dynamically add and remove themselves to and from a given equivalence class by contacting an RCDS location server.

Other planned work involves use of RCDS to distribute Usenet discussion groups. Such a RCDS-netnews hybrid would allow formation of "private" newsgroups, would not require each newsgroup to be replicated onto each server, would have more robust propagation than Usenet's flooding algorithm, and would allow seamless integration of local stores of recent articles, with "archives" of older ones, even on different servers, with the user viewing them as a single collection.

A "program builder" client is planned which will allow automatic building of software packages on various platforms. A description of the software package, consisting of a bill of materials and assembly instructions, will be stored in an RCDS-accessible file. The components of the package will be named with LIFNs and their descriptions (along with that of the package itself) will be cryptographically signed. The "program builder" will thus be able to fetch all components of a particular package, verify their authenticity and integrity, compile them and assemble them according to the instructions. This is intended to facilitate greater sharing of software packages by making it easier to construct large programs out of components from many different sources.

We plan to investigate an RCDS/Harvest Cache hybrid by modifying the Harvest Cache system to use RCDS servers. The hybrid system would combine the performance benefits of server-side ("push") replication with those obtained from client-side ("pull") replication.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[2] A. Bestavros. Demand-based document dissemination for the World-Wide Web. Technical Report TR-95-003, Computer Science Department, Boston University, Feb. 1995. Available as ftp://cs-ftp.bu.edu/techreports/95-003-web-server-dissemination.ps.Z.

[3] C. M. Bowman, P. B. Danzig, U. Manber, and M. F. Schwartz. Scalable Internet resource discovery: Research problems and approaches. *Commun. ACM*, 37(8):98–107, Aug. 1994.

[4] H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science problems. In *Supercomputing '96*, Pittsburgh, PA, Nov. 1996.

[5] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. Technical Report 95-611, Computer Science Department, University of Southern California, Mar. 1995.

[6] J. Gertzman and M. Seltzer. The case for geographical push-caching. In *HotOS Conference*, 1994. Available as ftp://das-ftp.harvard.edu/techreports/tr-34-94.ps.gz.

[7] M. McCahill, J. Romkey, M. Schwartz, K. Sollins, T. Verschuren, and C. Wieder. Report of the IAB workshop on Internet Information Infrastructure. *Internet Request For Comments*, 1862, Nov. 1995.

[8] M. Schwartz. Harvest project status and directions. Available at http://harvest.cs.colorado.edu/harvest/projstatus.html, Dec. 1995.

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

[10] M. Spasojevic, C. M. Bowman, and A. Spector. Using wide-area file systems within the world-wide web. In *Second International WWW Conference*, Chicago, Illinois, Oct. 1994. Available as http://www.transarc.com/afs/transarc.com/public/trg/papers/www94/index.html.