

A Scalable Implementation for Human-Friendly URIs

Gerco Ballintijn
Patrick Verkaik
Egaon Amade
Maarten van Steen
Andrew S. Tanenbaum

Internal report IR-466
November 1999

Abstract.

In the Web, Uniform Resource Identifiers (URIs) are used to name resources. The most common form of URI, the Uniform Resource Locator (URL) has, unfortunately, some scalability problems. In this paper, we propose the use of Human-Friendly Names (HFNs) to solve these scalability problems. HFNs are high-level names that allow (human) users to easily deal with names. We also describe a scalable HFN-to-URL resolution mechanism. This mechanism is based on the existing Domain Name System (DNS) and the Globe Location Service. To gain experience and validate our ideas, we have implemented our HFN resolution scheme.

Keywords: Naming, Resource Location, Scalability, Wide-Area Systems

This work was sponsored by a grant from the NLnet Foundation.



vrije Universiteit

Department of Mathematics and Computer Science

1 Introduction

Resource in the WWW are named using a Uniform Resource Identifiers (URIs) [4]. In the current Web, the most common and well-known form of URI is Uniform Resource Locator (URL) [5]. A URL is used in the WWW for two distinct purposes: (1) to identify and (2) to access these resources. Combining these two uses, unfortunately, leads to scalability problems, since resource identification has different requirements than resource access. For instance, if we want to replicate a popular Web page to improve its availability, we need support in our naming system to provide transparent access. However, the URL naming mechanism cannot provide us with this support, since a URL identifies only one location. Replicated Web pages in the current Web therefore use ad hoc solutions. They have multiple URLs naming the same (replicated) Web resource, and require the client to make an explicit choice on which URL to use.

Uniform Resource Names (URNs) [13] provide a solution to the scalability problems associated with URLs. A URN is, like a URL, also a form of URI. However, a URN differs from a URL in that it only *identifies* a Web resource. A URN does not indicate the location of a resource, nor does it contain other information that might change in the future. It is solely used to identify a resource. For example, ISBN numbers are potential URNs. To access the resource identified by a URN, the URN needs to be resolved into access information, for instance a URL. Using URNs to identify resources and URLs to access resources allows one URN to (indirectly) refer to many locations (URLs). The separation thus provides us with the ability to transparently support replicated Web resources. Since a URN is a stable reference to a resource and not its location, we can also move the resource around without changing its URN. A URN can thus support mobile resources by (indirectly) referring to a dynamic set of URLs.

Since URNs are intended to be primarily used by machines, there is no requirement to make them easy to use by humans (other than being easily transcribable). However, humans do need a way to name Web resources in a location-independent way. To fill this gap, a new kind of URI needs to be introduced, as suggested in [12], the Human-Friendly Name (HFN). HFNs are high-level names that allow (human) users to easily deal with names. Unlike URNs, HFNs therefore explicitly allow the use of descriptive names. However, HFNs do eventually need to be resolved to URLs, when the user needs to access the named resource. One way to do this, is to bind HFNs to URNs, effectively making HFN resolution a two-step process: first resolve the HFN to a URN, and then resolve the URN to a URL. Unfortunately, the problem of resolving HFNs to URLs in a scalable manner has not been addressed.

In this paper, we describe the implementation of a HFN-to-URL resolution mechanism. We have taken special care to make the resolution mechanism scalable in two ways. First, we can support a large number of resources. Second, we can support resources distributed over a large geographical area. The HFN-to-URL resolution mechanism supports a hierarchical name space, similar to the UNIX file system. Our solution is based on the existing Domain Name System (DNS) [1, 10] and the Globe Location Service [15]. To our knowledge, our design provides the first real solution to the HFN-to-URL resolution problem.

The rest of this paper is structured as follows. Section 2 describes the resource and name space model we use. Section 3 describes the architecture of our name resolution mechanism. In Section 4, we discuss the scalability aspects of our system. In Section 5, we describe our current implementation. Section 6 contains a discussion of related work, and in Section 7 we draw our conclusion and describe some future work.

2 Model

For our naming system, we restrict ourselves to highly popular, and thus replicated Web resources such as popular Web pages and software distributions. Other resource types, such as personal Web pages or mobile resources, are not yet supported due to scalability constraints. It is our goal to support 10^7 named resources. We restrict the number of supported resources to this large but limited number, because our architecture uses the existing DNS infrastructure. We also assume that changes to a particular name always originate from the same geographical area. The reasons for choosing this specific resource model are further explained in Section 4.

For our name space model we stay close to the familiar hierarchical name space, as found in the UNIX file system. An example of a human friendly URI supported by our system is `hfn://nl/vu/cs/globe`. The name space has one root directory named `//`, and every HFN is a sequence of simple names, indicating a path in the name space, from the root down to some leaf directory. Our name space uses `/` as separator and allows only letters, digits, and hyphen (`-`) in a simple name. Names are case insensitive. The reason for using a limited character set is also related to the underlying DNS infrastructure.

Security in our model consist only of preventing unauthorized changes to the HFN-to-URL mapping. We do not intend to make the HFN-to-URL mapping confidential, since we assume that HFNs will be used in much the same way as URLs are used today. Our resolution scheme can therefore not be used as an access control scheme.

3 Architecture

The HFN-to-URL mapping is an n-to-m relation, that might change frequently. To efficiently store, retrieve, and update this mapping, we split it into two separate mappings and introduce the **object handle**. The first mapping is the HFN-to-object handle mapping. The second mapping is the object handle-to-URL mapping. By splitting the HFN-to-URL mapping in two separate mappings, we have an n-to-1 relation and a 1-to-m, which are far easier to maintain.

The main purpose of HFN-to-object handle mapping is to *identify* a resource by providing its object handle. The object handle identifies a Web resource and is globally unique. The object handle is, in fact, a URN, as described in the introduction. The HFN-to-object handle mapping is maintained by a **name service**. The object handle-to-URL mapping is maintained by a **location service**, and its main purpose is to *locate* a resource. HFN resolution thus consists of two steps. In the first step, the HFN is resolved to an object handle by the name service, and in the second step, the object handle is resolved to a URL by a location service. The server indicated in the URL should be near the user requesting the HFN resolution.

To use our naming system, we add three new elements to the normal setup of Web browsers and HTTP servers: a HFN-to-URL proxy, a name service, and a location service. It is the task of the HFN-to-URL proxy to recognize HFNs and resolve them by querying the name and location service. With the URL obtained from the location service, the proxy accesses the named resource. In the future, a HFN-aware browser will perform the task of the proxy itself.

Figure 1 shows the setup we propose to retrieve Web resources named by HFNs. When a user enters a HFN in the Web browser the browser contacts the HFN-to-URL proxy to obtain the Web resource named by the HFN (step 1). The proxy recognizes the HFN and contacts the name service in step 2. The name service resolves the name to an object handle, and returns it to the proxy (step 3). The proxy then contacts the location service in step 4. The location service resolves the object handle to a URL, and returns it to the proxy (step 5). The proxy can now contact, for instance, an HTTP

server in step 6, which returns an HTML page in step 7. The proxy then returns this HTML page to the Web browser (step 8).

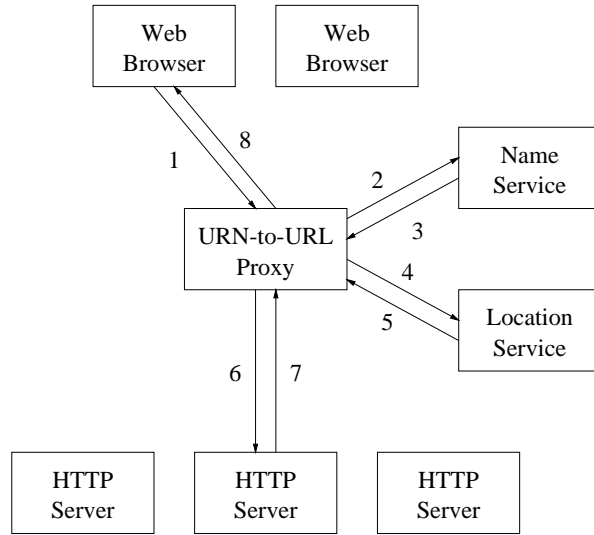


Figure 1: The setup to retrieve Web resources named by HFNs.

3.1 Name Service

We use the Domain Name System (DNS) to store the mapping from a HFN to an object handle. DNS is at the moment primarily used to name Internet hosts and Email destinations. We can, however, reuse the existing DNS infrastructure with only minimal changes.

3.1.1 The Domain Name System

DNS provides an extensible hierarchical name space, in which more general naming authorities delegate responsibility for parts of their name space (subdomains) to more specific naming authorities. For example, the naming authority responsible for the `com` domain, delegates the responsibility for the `intel.com` domain to the Intel Corporation. A naming authority is responsible for providing the resources needed to store and query a DNS name, and can decide for itself which names to store in its subdomain. The Intel Corporation can thus create whatever host name or email destination it wants in its subdomain.

Resolving a host name in DNS consists, conceptually, of contacting a sequence of name servers. The domains stored by the sequence of name servers are increasingly specific, allowing the resolution of an increasing part of the host name. For example, to resolve the host name `www.intel.com`, the resolution process visits, in turn, the name servers responsible for the root (i.e., "."), `com`, and `intel.com` domain, respectively. The last name server will be able to resolve the complete host name.

To enhance its performance, DNS makes extensive use of caching. When a name server is asked to resolve a DNS name recursively, it will contact the sequence of name servers itself to resolve the name. The name server can then cache the intermediate and end results of the resolution process, to

avoid having to contact the sequence of name servers a second time. To allow effective caching, DNS assumes that the name-to-address mapping does not change frequently.

DNS uses **resource records** to store the different kinds of name mappings at name servers. A DNS name can have zero or more resource records. There are two kinds of resource records. The first kind stores user data, like the resource records for naming Internet hosts and Email destinations. This kind of record associates an Internet address or a mail server with a DNS name. The second kind is the name server resource record, which is used internally to implement the name space delegation. This resource record associates another DNS server with a DNS name, indicating another name server at which to continue name resolution.

3.1.2 Using DNS to Store HFNs

To store a HFN in DNS, we first need to define a translation between our name space and the DNS name space. A name in our name space is a sequence of simple names, where the simple names at the start of the name refer to directories closer to the root. In DNS, simple names at the end of a name refer to directories (domains) closer to the root. We thus need to reverse the order of simple names in a HFN to store it in DNS. A second difference is that we prefer to use the slash (i.e., “/”) as a separator, while DNS uses a dot (i.e., “.”). We therefore also need to change slashes into dots. So, for example, we will use the DNS name `c.b.a.` to store the HFN `/a/b/c`.

To resolve a HFN, the HFN-to-URL proxy starts by removing the urn tag and name space identifier of the HFN. The resulting name then needs to be translated to a proper DNS name. During the translation the order of the simple names is reversed and slashes are changed into dots. The resulting DNS name can be used to query the DNS name service. The query will specify the DNS name and the request for a resource record holding an object handle.

In our approach, we need to associate an object handle with a DNS name, instead of associating an IP addresses or Email servers. We therefore need to introduce a new resource record to store object handle mappings at DNS servers. This resource record will contain an object handle encoded using an ASCII encoding.

When a new HFN is introduced, a new resource record with an object handle needs to be inserted at the proper name server. The proper name server is the server responsible for the parent domain of the HFN. For instance, to insert `/a/b/c`, we need to contact the server responsible for the `b.a.` domain. The actual insertion at the server can be done dynamically using the new RPC-like DNS *update* operations [17].

3.2 Location Service

To resolve an object handle into a URL we use the Globe location service [15]. The Globe location service allows us to associate a set of URLs with an object handle. The service offers, besides the object handle look-up operation, two update operations: insert and delete. Insert and delete operations are used to add a URL to or remove a URL from the set of URLs associated with an object handle. Since the location service is distributed across a wide-area network, it should deal gracefully with server failures, network partitions, and (long) communication delays.

3.2.1 Architecture

To efficiently update and look-up URLs, we organize the underlying wide-area network (i.e. the Internet) as a hierarchy of **domains**. For example, a lowest level domain may represent a campus-wide network of a university, whereas the next higher level domain represents the city where that

campus is located. Each domain is represented in the location service by a **directory node**. Together the directory nodes form a worldwide search tree. Note that even though the concept of a domain in the location service is somewhat similar to the concept of a domain in DNS, the location service hierarchy is independent of the organization of DNS.

A directory node has a **contact record** for every (registered) resource in its domain. The contact record is divided into a number of **contact fields**, one for each child node. A directory node stores either a **forwarding pointer** or the actual URLs in the contact field. A forwarding pointer indicates that URLs can be found at the child node. Contact records at leaf nodes are different, they contain only one contact field storing the URLs from the leaf domain.

Every URL has a path of forwarding pointers from the root down, pointing to it. We can always locate a URL by following this path. In the normal case, URLs are stored in leaf nodes. However, storing URLs at intermediate nodes may, in the case of highly mobile resources, lead to considerably more efficient look-ups, as we explain in [15]. To simplify the discussion, we assume, for this paper, that URLs are always stored in leaf nodes.

Figure 2 shows as an example the contact records for one object handle. In this example, root node N0 has one forwarding pointer for the object handle, indicating that URLs can be found in its left subtree, rooted at node N1. Node N1, in turn, has two forwarding pointers, pointing to nodes N2 and N3. Both of these nodes have a forwarding pointer to a leaf node where an actual URL is stored.

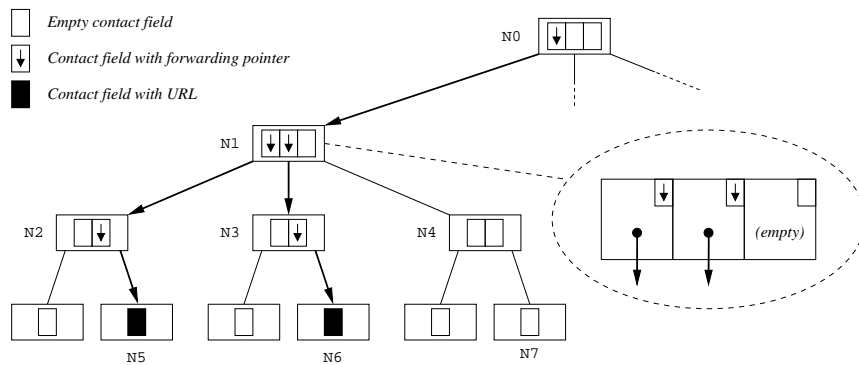


Figure 2: The organization of contact records in the tree for a specific resource.

Our search tree described so far obviously does not yet scale. In particular, higher-level directory nodes have to store a large number of contact records and handle a large numbers of requests, since there is a path of forwarding pointers from the root node down for every object handle. Our solution is to partition a directory node into one or more directory subnodes, such that each subnode is responsible for only a subset of the records originally stored at the directory node. We use a hashing technique on the object handles to identify subnodes at parents and children (see [3] for further details).

3.2.2 Operations

When a client wants to know the URL of a resource, it initiates a look-up operation at the leaf node of the domain in which it resides. The client provides the resource's object handle as parameter. The look-up operation starts by checking if the leaf node has a contact record for the object handle. If the leaf node has a contact record, the operation returns the URL found in the contact record. Otherwise,

it recursively checks nodes on the path from the leaf node up to the root. If the look-up operation finds a contact record at any of these nodes, the path of forwarding pointers starting at this node is followed downwards to a leaf node where a URL is found. If no contact record is found at any of the nodes on the path from the leaf node to the root, the object handle is unknown.

For example, consider in Figure 2 a client located near leaf node N7. When the leaf node is contacted by the client with a request for a URL, it will forward the request to its parent, node N4, since it does not contain a contact record. Node N4 also does not know about the resource, and will, in turn, also forward the request to its parent, node N1. Node N1 does know about the resource, and forwards the request to one of its children indicated by a forwarding pointer. The look-up operation can now follow the path of forwarding pointer to one of the leaf nodes, for instance, leaf node N6. By going higher in the search tree, the look-up operation effectively broadens the area that is searched for a URL.

The goal of the insert operation is to store a URL at a leaf node and create a path of forwarding pointers to the leaf node. When a resource has a new URL in a leaf domain, the object inserts this new URL at the node of the leaf domain. The insert operation starts by inserting the URL in the contact record of the leaf node. The insert operation then recursively requests the parent nodes to install a forwarding pointer. The recursion stops when a node is found that already contains a forwarding pointer, or otherwise at the root. The delete operation removes the URL and path of forwarding pointers analogous to the insert operation. Further technical details can be found in [14].

4 Discussion

An important aspect of our HFN-to-URL resolution scheme is its scalability. As explained in the introduction, we can distinguish two types of scalability: (1) the support of a large number of resources, and (2) the support for resources that are distributed over a large geographical area. For our resolution scheme to be scalable, both kinds of scalability need to be addressed in the name service and in the location service.

4.1 Name Service

The first form of scalability requires our name service to deal effectively with a large number of resources, i.e. deal with a large number of HFN-to-object handle mappings. We tackle this problem by limiting the number of supported HFNs. By placing an upper bound on the number of supported resources, we can avoid overloading the current DNS infrastructure with a too large number of new names. Given that the current DNS infrastructure supports in the order of 10^8 host names, we feel we can safely add 10^7 new names.

The second form of scalability requires our name service to deal effectively with names distributed over a large geographical area. We tackle this second problem by ensuring the use of locality in look-up and update operations. The locality of look-up operations in DNS is provided by caches. If a resource named by a HFN is popular, its name-to-object handle mapping will be stored in the caches of name servers, providing clients located near the cache with local access to the name-to-object handle mapping. A DNS query to obtain the object handle can thus be answered directly, without the need to contacting a name server located far away. By assuming the use of popular Web resources and a stable HFN to object handle mapping, we ensure that caching remains effective. Update operations in the name service use locality as well. We can always place the name server that stores the resource record for a certain name near the area where the changes originate, since we assume that those

changes always originate from the same geographical area.

Under the assumptions stated in Section 2, DNS is certainly an attractive name service given its existing infrastructure. Unfortunately, if we want to drop those assumptions in favor of a less restrictive resource model, scalability problems arise in DNS that prevent our HFN resolution mechanism from scaling further. If we want to support more than 10^7 resources, or resources that are unpopular or mobile, we need to replace DNS with a more scalable name service. We describe the design of such a naming service in [2].

A different problem with using DNS as the name service in our scheme, is that it places restrictions on the syntax of the HFN name space. Since we need to translate our HFN names to DNS names, we can support only the small subset of the ASCII character set that is allowed by DNS. Given the increasing number of Web users speaking languages not supported by ASCII, this limit will become a problem in the future.

4.2 Location Service

The Globe location service deals with a large number of object handle-to-URL mappings by partitioning the nodes within its search tree into subnodes. If a tree node becomes too big for one machine to handle, the node can be split into two or more subnodes that can be run on separate machines.

The location service deals with URLs distributed over a large geographical area by using locality through its distributed search tree and related look-up algorithm. By starting the look-up operation at the leaf node to search the nearby area, and continuing at higher nodes in the tree to search larger areas, the location service avoids using remote resources, when a URL can be found using local resources only. The use of locality in the Globe location service is further discussed in [15].

5 Implementation

We have implemented our HFN resolution scheme to support naming in the Globe framework [16]. We are using the software created by the BIND project to implement the name service part, and we have implemented a prototype of the location service and the HFN-to-URL proxy ourselves. The software is currently used in an experimental setup to validate the Globe framework. To store object handles, we choose to use the existing TXT resource record, instead of implementing a new resource record. Using the TXT resource record, allowed us to use the BIND software in an unmodified form.

For the moment, we use the standard authorization scheme to prevent unauthorized update operations on the name service. In this scheme, a DNS name server performs update operations only when they are initiated from hosts trusted by the server. In the future, we could use the new security features that have been added to DNS [8, 7]. Security in the Globe location service is part of ongoing research.

6 Related Work

The URI research community has so far shown little interest area of human friendly names. Most worked currently done falls in either the URL or the URN category. The URL work deals mainly with assignment and specification of new access schemes.

The IETF URN working group has created several RFC documents describing the concepts behind URNs. Their focus thus far has been on the definition of a URN name space [13], and a general architecture [12]. In this architecture, the URN name space actually consists of several independent URN name spaces, and every URN name space has (potentially) its own specific URN resolver. Therefore

to resolve a URN, the first thing to do is to select the appropriate name resolver. This selection of name resolver is done by a Resolver Discovery Service (RDS).

There are currently two proposals for an RDS. Daniel and Mealling [6] propose to build an RDS using DNS. In their proposal, DNS contains resource records specifying rewrite rules. When a URN needs to be resolved, these rewrite rules are applied to the URN, resulting in resolver that can resolve the complete URN, or possibly even the resource itself. Slottow (and Sollins) [11] propose to build an RDS as a distributed database, using a distributed B-tree. Within the nodes of the B-tree, replication is used to maintain high availability.

As mentioned in Section 3, an object handle performs the role of URN. In this sense, the location service performs the role of a specific URN resolver. The location service can potential also be used to resolve other kinds of URNs. Since we were looking for this research at one specific URN space, i.e. the object handle space, our work has not included a resolver discovery system.

Lampson [9] has designed a global name service with a focus on scalability, high availability, and continuing evolution. The name service uses a tree-shaped name space like DNS, but distinguishes at the implementation level between local and global directories. The global directories guide name resolution from the root directory down to local directories. The global directories are replicated and maintain consistency through the use of a *sweep* operation that propagates state changes between replicas. The local directories allow further name resolution to retrieve the values we are interested in. Like DNS, caching provides only form of locality.

7 Conclusions and Future Work

As part of our research into wide-area distributed systems, we have developed a location service that, together with DNS, can be used to resolve HFNs to URLs in a scalable fashion. Scalability is achieved by using to distinct mappings, one for naming resources and one of locating them. Using this separation, we can apply techniques specific to the service to obtain locality and thus scalability. An important part of our design is the reuse of the existing DNS infrastructure. This provides us with large benefits, in the form of the existing experience and infrastructure, and some problems like only support for a limited (but large) number of specific kind of resource.

Future work will consist of using our naming scheme in larger experimental setups to gain more experience. Security is currently designed, by still needs to be implemented. A larger project will be the implementation of our own name service, as described in [2], to replace DNS.

References

- [1] P. Albitz and C. Liu. *DNS and BIND*. O'Reilly & Associates, Sebastopol, CA., 1992.
- [2] G. Ballintijn and M. van Steen. Scalable Naming in Global Middleware. Technical Report IR-464, Vrije Universiteit, Amsterdam, The Netherlands, Oct. 1999.
- [3] G. Ballintijn, M. van Steen, and A. S. Tanenbaum. Exploiting Location Awareness for Scalable Location-Independent Object IDs. In *Proc. Fifth Annual ASCI Conference*, pages 321–328, Heijen, The Netherlands, June 1999. ASCI.
- [4] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, Aug. 1998.

- [5] T. Berners-Lee, L. Masinter, and M. McCahill. RFC 1738: Uniform Resource Locators (URL), Dec. 1994.
- [6] R. Daniel and M. Mealling. RFC 2168: Resolution of Uniform Resource Identifiers using the Domain Name System, June 1997.
- [7] D. Eastlake Third. RFC 2137: Secure Domain Name System Dynamic Update, Apr. 1997.
- [8] D. Eastlake Third and C. Kaufman. RFC 2065: Domain Name System Security Extensions, Jan. 1997.
- [9] B. W. Lampson. Designing a Global Name Service. In *Proc. 4th ACM Symposium on Principles Of Distributed Computing*, pages 1–10. ACM, 1985.
- [10] P. Mockapetris. RFC 1034: Domain Names - Concepts and Facilities, Nov. 1987.
- [11] E. C. Slottow. Engineering a Global Resolution Service. Master's thesis, MIT, Cambridge, Ma., USA, June 1997.
- [12] K. Sollins. RFC 2276: Architectural Principles of Uniform Resource Name Resolution, Jan. 1998.
- [13] K. Sollins and L. Masinter. RFC 1737: Functional Requirements for Uniform Resource Names, Dec. 1994.
- [14] M. van Steen, F. J. Hauck, G. Ballintijn, and A. S. Tanenbaum. Algorithmic Design of the Globe Wide-Area Location Service. *The Computer Journal*, 41(5):297–310, 1998.
- [15] M. van Steen, F. J. Hauck, P. Homburg, and A. S. Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, pages 104–109, Jan. 1998.
- [16] M. van Steen, P. Homburg, and A. S. Tanenbaum. Globe: A Wide-Area Distributed System. *IEEE Concurrency*, pages 70–78, Jan. 1999.
- [17] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE), Apr. 1997.