Network Working Group
Request for Comments: 3404
Obsoletes: 2915, 2168
Category: Standards Track

M. Mealling
VeriSign

October 2002

Dynamic Delegation Discovery System (DDDS) Part Four: The Uniform Resource Identifiers (URI) Resolution Application

Status of this Memo

Copyright Notice

Abstract

This document describes a specification for taking Uniform Resource Identifiers (URI) and locating an authoritative server for information about that URI. The method used to locate that authoritative server is the Dynamic Delegation Discovery System.

This document is part of a series that is specified in "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS" (RFC 3401). It is very important to note that it is impossible to read and understand any document in this series without reading the others.

Table of Contents

1. Introduction

The Dynamic Delegation Discovery System (DDDS) is used to implement lazy binding of strings to data, in order to support dynamically configured delegation systems. The DDDS functions by mapping some unique string to data stored within a DDDS Database by iteratively applying string transformation rules until a terminal condition is reached.

This document describes a DDDS Application for resolving Uniform Resource Identifiers (URI). It does not define the DDDS Algorithm or a Database. The entire series of documents that do so are specified in "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS" (RFC 3401) [1]. It is very important to note that it is impossible to read and understand any document in that series without reading the related documents.

Uniform Resource Identifiers (URI) have been a significant advance in retrieving Internet-accessible resources. However, their brittle nature over time has been recognized for several years. The Uniform Resource Identifier working group proposed the development of Uniform Resource Names (URN) [8] to serve as persistent, location-independent identifiers for Internet resources in order to overcome most of the problems with URIs. RFC 1737 [6] sets forth requirements on URNs.

During the lifetime of the URI-WG, a number of URN proposals were generated. The developers of several of those proposals met in a series of meetings, resulting in a compromise known as the Knoxville framework. The major principle behind the Knoxville framework is that the resolution system must be separate from the way names are assigned. This is in marked contrast to most URIs, which identify the host to contact and the protocol to use. Readers are referred to [7] for background on the Knoxville framework and for additional information on the context and purpose of this proposal.

Separating the way names are resolved from the way they are constructed provides several benefits. It allows multiple naming approaches and resolution approaches to compete, as it allows different protocols and resolvers to be used. There is just one problem with such a separation - how do we resolve a name when it can't give us directions to its resolver?

For the short term, the Domain Name System (DNS) is the obvious candidate for the resolution framework, since it is widely deployed and understood. However, it is not appropriate to use DNS to maintain information on a per-resource basis. First of all, DNS was never intended to handle that many records. Second, the limited record size is inappropriate for catalog information. Third, domain names are not appropriate as URNs.

Therefore our approach is to use the DDDS to locate "resolvers" that can provide information on individual resources, potentially including the resource itself. To accomplish this, we "rewrite" the URI into a Key following the rules found in the DDDS. This document describes URI Resolution as an application of the DDDS and specifies the use of at least one Database based on DNS.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

All capitalized terms are taken from the vocabulary found in the DDDS algorithm specification found in RFC 3403 [3].

3. The Distinction Between URNs and URIs

From the point of view of this system, there is no theoretical difference between resolving URIs in the general case and URNs in the specific case. Operationally however, there is a difference that stems from URI resolution possibly not becoming of widespread use. If URN resolution is collapsed into generic URI resolution, URNs may suffer by the lack of adoption of URI resolution.

The solution is to allow for shortcutting for URN resolution. In the following specification generic URI resolution starts by inserting rules for known URI schemes into the 'uri.arpa.' registry. For the 'URN:' URI scheme, one of the rules found in 'uri.arpa.' would be for the 'urn' URI scheme. This rule would simply delegate to the 'urn.arpa.' zone for additional NAPTRs based on the URN namespace. Essentially, the URI Resolution Rewrite Rule for 'URN:' is the URN Resolution Application's First Well Known Rule.

Therefore, this document specifies two DDDS Applications. One is for URI Resolution and the other is for URN Resolution. Both are technically identical but by separating the two URN Resolution can still proceed without the dependency.

4. The URI and URN Resolution Application Specifications

This template defines the URI and URN Resolution DDDS Application according to the rules and requirements found in [3]. The DDDS database used by this Application is found in [4] which is the document that defines the Naming Authority Pointer (NAPTR) DNS Resource Record (RR) type.

4.1 Application Unique String

The Application Unique String is the URI or URN for which an authoritative server is being located. This URI or URN MUST be canonicalized and hex encoded according to the "absolute-uri" production found in the Collected ABNF from RFC 2396 [15].

4.2 First Well Known Rule

In the URI case, the first known key is created by taking the URI scheme. In the URN case, the first known key is the Namespace Identifier. For example, the URI 'http://www.example.com/' would have a 'http' as its Key. The URN 'urn:foo:foospace' would have 'foo' as its first Key.

4.3 Flags

At this time only four flags, "S", "A", "U", and "P", are defined. The "S", "A" and "U" flags are for a terminal lookup. This means that the Rule is the last one and that the flag determines what the next stage should be. The "S" flag means that the output of this Rule is a domain-name for which one or more SRV [9] records exist. See Section 5 for additional information on how URI and URN Resolution use the SRV record type. "A" means that the output of the Rule is a domain-name and should be used to lookup either A, AAAA, or A6 records for that domain. The "U" flag means that the output of the Rule is a URI [15].

The "P" flag says that the remainder of the DDDS Algorithm is ignored and that the rest of the process is application specific and outside the scope of this document. An application can use the Protocol part found in the Services field to identify which Application specific set of rules that should be followed next. The record that contains the 'P' flag is the last record that is interpreted by the rules in this document. One might think that this would also make the "P"

flag an indicator of a terminal lookup but this would be incorrect since a "terminal" Rule is a DDDS concept and this flag indicates that anything after this rule does not adhere to DDDS concepts at all.

The remaining alphabetic flags are reserved for future versions of this specification. The numeric flags may be used for local experimentation. The S, A, U and P flags are all mutually exclusive, and resolution libraries MAY signal an error if more than one is given. (Experimental code and code for assisting in the creation of Rewrite Rules would be more likely to signal such an error than a client such as a browser.) It is anticipated that multiple flags will be allowed in the future, so implementers MUST NOT assume that the flags field can only contain 0 or 1 characters. Finally, if a client encounters a record with an unknown flag, it MUST ignore it and move to the next Rule. This test takes precedence over any ordering since flags can control the interpretation placed on fields. A novel flag might change the interpretation of the regexp and/or replacement fields such that it is impossible to determine if a record matched a given target.

The "S", "A", and "U" flags are called 'terminal' flags since they halt the looping DDDS algorithm. If those flags are not present, clients may assume that another Rule exists at the Key produced by the current Rewrite Rule.

4.4 Services Parameters

Service Parameters for this Application take the form of a string of characters that follow this ABNF:

```
service_field = [ [protocol] *("+" rs)]
protocol      = ALPHA *31ALPHANUM
rs            = ALPHA *31ALPHANUM
; The protocol and rs fields are limited to 32
; characters and must start with an alphabetic.
```

In other words, an optional protocol specification followed by 0 or more resolution services. Each resolution service is indicated by an initial '+' character.

The empty string is also valid. This will typically be seen at the beginning of a series of Rules, when it is impossible to know what services and protocols will be offered at the end of a particular delegation path.

4.4.1 Services

The service identifiers that make up the 'rs' production are generic for both URI and URN resolution since the input value types itself based on the URI scheme. The list of valid services are defined in [11].

Examples of some of these services are:

I2L: given a URI return one URI that identifies a location where the original URI can be found.

I2Ls: given a URI return one or more URIs that identify multiple locations where the original URI can be found.

I2R: given a URI return one instance of the resource identified by that URI.

I2Rs: given a URI return one or more instances of the resources identified by that URI.

I2C: given a URI return one instance of a description of that resource.

I2N: given a URI return one URN that names the resource (Caution: equality with respect to URNs is non-trivial. See [6] for examples of why.)

4.4.2 Protocols

The protocol identifiers that are valid for the 'protocol' production MUST be defined by documents that are specific to URI resolution. At present the THTTP [10] protocol is the only such specification.

It is extremely important to realize that simply specifying any protocol in the services field is insufficient since there are additional semantics surrounding URI resolution that are not defined within the protocols. For example, if Z39.50 were to be specified as a valid protocol it would have to additionally define how it would encode requests for specific services, how the URI is encoded, and what information is returned.

4.4.3 Applicability of Services

Since it is possible for there to be a complex set of possible protocols and services a client application may often need to apply a more complex decision making process to a set of records than simply matching on an ordered list of protocols. For example, if there are 4 rules that are applicable the last one may have a more desirable Service field than the first. But since the client may be satisfied by the first it will never know about the 4th one which may be 'better'.

To mitigate this the client may want to slightly modify the DDDS algorithm (for this application only!) in order to determine if more applicable protocols/services exist. This can safely be done for this application by using a more complex interaction between steps 3 and 4 of the DDDS algorithm in order to find the optimal path to follow. For example, once a client has found a rule who's Substitution Expression produces a result and who's Service description is acceptable, it may make note of this but continue to look at further rules that apply (all the while adhering to the Order!) in order to find a better one. If none are found it can use the one it made note of.

Keep in mind that in order for this to remain safe, the input to step 3 and the output of step 4 MUST be identical to the basic algorithm. The client software MUST NOT attempt to do this optimization outside a specific set of Rewrite Rules (i.e., across delegation paths).

4.5 Valid Databases

At present only one DDDS Database is specified for this Application. "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database" (RFC 3403) [4] specifies a DDDS Database that uses the NAPTR DNS resource record to contain the rewrite rules. The Keys for this database are encoded as domain-names.

The output of the First Well Known Rule for the URI Resolution Application is the URI's scheme. In order to convert this to a unique key in this Database the string '.uri.arpa.' is appended to the end. This domain-name is used to request NAPTR records which produces new keys in the form of domain-names.

The output of the First Well Known Rule of the URN Resolution Application is the URN's namespace id. In order to convert this to a unique key in this Database the string '.urn.arpa.' is appended to the end. This domain-name is used to request NAPTR records which produces new keys in the form of domain-names.

DNS servers MAY interpret Flag values and use that information to include appropriate SRV and A records in the Additional Information portion of the DNS packet. Clients are encouraged to check for additional information but are not required to do so. See the Additional Information Processing section of RFC 3404 for more information on NAPTR records and the Additional Information section of a DNS response packet.

The character set used to encode the substitution expression is UTF-8. The allowed input characters are all those characters that are allowed anywhere in a URI. The characters allowed to be in a Key are those that are currently defined for DNS domain-names. The "i" flag to the substitution expression is used to denote that, where appropriate for the code points in question, any matches should be done in a case-insensitive way.

5. Examples

5.1 An Example Using a URN

Consider a URN that uses the hypothetical FOO namespace. FOO numbers are identifiers for approximately 30 million registered businesses around the world, assigned and maintained by Fred, Otto and Orvil, Inc. The URN might look like:

```
urn:foo:002372413:annual-report-1997
```

The first step in the resolution process is to find out about the FOO namespace. The namespace identifier [8], "foo", is extracted from the URN and prepended to '.urn.arpa.', producing 'foo.urn.arpa.'. The DNS is queried for NAPTR records for this domain which produces the following results:

```
foo.urn.arpa.
;;      order pref flags service          regexp        replacement
IN NAPTR 100  10  "s" "foolink+I2L+I2C"  ""   foolink.udp.example.com.
IN NAPTR 100  20  "s" "rcds+I2C"          ""  rcds.udp.example.com.
IN NAPTR 100  30  "s" "thttp+I2L+I2C+I2R" ""  thttp.tcp.example.com.
```

The order field contains equal values, indicating that no order has to be followed. The preference field indicates that the provider would like clients to use the special 'foolink' protocol, followed by the RCDS protocol, and that THTTP is offered as a last resort. All the records specify the "s" flag which means that the record is terminal and that the next step is to retrieve an SRV record from DNS for the given domain-name.

The service fields say that if we speak of foolink, we will be able to issue either the I2L, I2C or I2R requests to obtain a URI or ask some complicated questions about the resource. The Resource Cataloging and Distribution Service (RCDS) [12] could be used to get some metadata for the resource, while THTTP could be used to get a URI for the current location of the resource.

Assuming our client does not know the foolink protocol but does know the RCDS protocol, our next action is to lookup SRV RRs for rcds.udp.example.com, which will tell us hosts that can provide the necessary resolution service. That lookup might return:

```
;;                              Pref Weight Port Target
rcds.udp.example.com  IN SRV 0    0    1000 deffoo.example.com.
                      IN SRV 0    0    1000 dbexample.com.au.
                      IN SRV 0    0    1000 ukexample.com.uk.
```

telling us three hosts that could actually do the resolution, and giving us the port we should use to talk to their RCDS server. (The reader is referred to the SRV specification [9] for the interpretation of the fields above.)

There is opportunity for significant optimization here. RFC 3404 defines that Additional Information section may be available. In this case the the SRV records may be returned as additional information for terminal NAPTRs lookups (as well as the A records for those SRVs). This is a significant optimization. In conjunction with a long TTL for *.urn.arpa. records, the average number of probes to DNS for resolving most URIs would approach one.

Note that the example NAPTR records above are intended to represent the result of a NAPTR lookup using some client software like nslookup; zone administrators should consult the documentation accompanying their domain name servers to verify the precise syntax they should use for zone files.

Also note that there could have been an additional first step where the URN was resolved as a generic URI by looking up urn.uri.arpa. The resulting rule would have specified that the NID be extracted from the URN and '.urn.arpa.' appended to it resulting in the new key 'foo.urn.arpa.' which is the first step from above.

5.2 CID URI Scheme Example

Consider a URI scheme based on MIME Content-Ids. The URI might look like this:

cid:199606121851.1@bar.example.com

(Note that this example is chosen for pedagogical purposes, and does not conform to the CID URI scheme.)

The first step in the resolution process is to find out about the CID scheme. The scheme is extracted from the URI, prepended to '.uri.arpa.', and the NAPTR for 'cid.uri.arpa.' looked up in the DNS. It might return records of the form:

```
cid.uri.arpa.
;;       order pref flags service        regexp           replacement
IN NAPTR 100   10   ""    ""  "!^cid:.+@([^\.]+\.)(.*)$!\2!i"    .
```

Since there is only one record, ordering the responses is not a problem. The replacement field is empty, so the pattern provided in the regexp field is used. We apply that regexp to the entire URI to see if it matches, which it does. The \2 part of the substitution expression returns the string "example.com". Since the flags field is empty, the lookup is not terminal and our next probe to DNS is for more NAPTR records where the new domain is 'example.com'.

Note that the rule does not extract the full domain name from the CID, instead it assumes the CID comes from a host and extracts its domain. While all hosts, such as 'bar', could have their very own NAPTR, maintaining those records for all the machines at a site could be an intolerable burden. Wildcards are not appropriate here since they only return results when there is no exactly matching names already in the system.

The record returned from the query on "example.com" might look like:

```
example.com.
;;      order pref flags service          regexp  replacement
IN NAPTR 100 50 "s" "z3950+I2L+I2C"     ""    z3950.tcp.example.com.
IN NAPTR 100 50 "s" "rescap+I2C"        ""    rescap.udp.example.com.
IN NAPTR 100 50 "s" "thttp+I2L+I2C+I2R" ""    thttp.tcp.example.com.
```

Continuing with the example, note that the values of the order fields are equal for all records, so the client is free to pick any record. The Application defines the flag 's' to mean a terminal lookup and that the output of the rewrite will be a domain-name for which an SRV record should be queried. Once the client has done that, it has the following information: the host, port, the protocol, and the services available via that protocol. Given these bits of information the client has enough to be able to contact that server and ask it questions about the cid URI.

Recall that the regular expression used \2 to extract a domain name from the CID, and \. for matching the literal '.' characters separating the domain name components. Since '\' is the escape character, literal occurrences of a backslash must be escaped by another backslash. For the case of the cid.uri.arpa record above, the regular expression entered into the master file should be "!^cid:.+@([^\\.]+\\.)(.*)$!\\2!i". When the client code actually receives the record, the pattern will have been converted to "!^cid:.+@([^\.]+\.)(.*)$!\2!i".

5.3 Resolving an HTTP URI Scheme

Even if URN systems were in place now, there would still be a tremendous number of host based URIs. It should be possible to develop a URI resolution system that can also provide location independence for those URIs.

Assume we have the URI for a very popular piece of software that the publisher wishes to mirror at multiple sites around the world:

http://www.example.com/software/latest-beta.exe

We extract the prefix, "http", and lookup NAPTR records for 'http.uri.arpa.'. This might return a record of the form:

```
http.uri.arpa. IN NAPTR
;;  order   pref flags service      regexp             replacement
     100     90   ""      ""   "!^http://([^/:]+)!1!i"       .
```

This expression returns everything after the first double slash and before the next slash or colon. (We use the '!' character to delimit the parts of the substitution expression. Otherwise we would have to use backslashes to escape the forward slashes, and would have a regexp in the zone file that looked like this: "/^http:\\/\\/([^\\/:]+)/\\1/i").

Applying this pattern to the URI extracts "www.example.com". Looking up NAPTR records for that might return:

```
www.example.com.
;;       order pref flags   service  regexp     replacement
 IN NAPTR 100  100  "s"   "thttp+L2R"   ""    thttp.example.com.
 IN NAPTR 100  100  "s"   "ftp+L2R"    ""     ftp.example.com.
```

Looking up SRV records for thttp.example.com would return information on the hosts that example.com has designated to be its mirror sites. The client can then pick one for the user.

6. Notes

- Registration procedures for the 'urn.arpa.' and 'uri.arpa.' DNS zones are specified in "Dynamic Delegation Discovery System (DDDS) Part Five: URI.ARPA Assignment Procedures" (RFC 3405 [5].

- If a record at a particular order matches the URI, but the client doesn't know the specified protocol and service, the client SHOULD continue to examine records that have the same order. The client MUST NOT consider records with a higher value of order. This is necessary to make delegation of portions of the namespace work. The order field is what lets site administrators say "all requests for URIs matching pattern x go to server 1, all others go to server 2".

- Note that SRV RRs impose additional requirements on clients.

7. IANA Considerations

The use of the "urn.arpa." and "uri.arpa." zones requires registration policies and procedures to be followed and for the operation of those DNS zones to be maintained. These policies and procedures are spelled out in a "Dynamic Delegation Discovery System (DDDS) Part Five: URI.ARPA Assignment Procedures (RFC 3405)" [5]. The operation of those zones imposes operational and administrative responsibilities on the IANA.

The registration method used for values in the Services and Flags fields is for a specification to be approved by the IESG and published as either an Informational or standards track RFC.

The registration policies for URIs is found in RFC 2717 [17]. URN NID registration policies are found in RFC 2611 [16].

8. Security Considerations

The use of "urn.arpa." and "uri.arpa." as the registry for namespaces is subject to denial of service attacks, as well as other DNS spoofing attacks. The interactions with DNSSEC are currently being studied. It is expected that NAPTR records will be signed with SIG records once the DNSSEC work is deployed.

The rewrite rules make identifiers from other namespaces subject to the same attacks as normal domain names. Since they have not been easily resolvable before, this may or may not be considered a problem.

Regular expressions should be checked for sanity, not blindly passed to something like PERL.

This document has discussed a way of locating a resolver, but has not discussed any detail of how the communication with the resolver takes place. There are significant security considerations attached to the communication with a resolver. Those considerations are outside the scope of this document, and must be addressed by the specifications for particular resolver communication protocols.

9. Acknowledgments

The editors would like to thank Keith Moore for all his consultations during the development of this document. We would also like to thank Paul Vixie for his assistance in debugging our implementation, and his answers on our questions. Finally, we would like to acknowledge our enormous intellectual debt to the participants in the Knoxville series of meetings, as well as to the participants in the URI and URN working groups.

Specific recognition is given to Ron Daniel who was co-author on the original versions of these documents. His early implementations and clarity of thinking was invaluable in clearing up many of the potential boundary cases.

References

[1] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS", RFC 3401, October 2002.

[2] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm", RFC 3402, October 2002.

[3] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, October 2002.

[4] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Four: The Uniform Resource Identifiers (URI) Resolution Application", RFC 3404, October 2002.

[5] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Five: URI.ARPA Assignment Procedures", RFC 3405y, October 2002.

[6] Sollins, K. and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, December 1994.

[7] Arms, B., "The URN Implementors, Uniform Resource Names: A Progress Report", D-Lib Magazine, February 1996.

[8] Moats, R., "URN Syntax", RFC 2141, May 1997.

[9] Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[10] Daniel, R., "A Trivial Convention for using HTTP in URN Resolution", RFC 2169, June 1997.

[11] Mealling, M., "URI Resolution Services Necessary for URN Resolution", RFC 2483, January 1999.

[12] Moore, K., Browne, S., Cox, J. and J. Gettler, "Resource Cataloging and Distribution System", Technical Report CS-97-346, December 1996.

[13] Sollins, K., "Architectural Principles of Uniform Resource Name Resolution", RFC 2276, January 1998.

[14] Daniel, R. and M. Mealling, "Resolution of Uniform Resource Identifiers using the Domain Name System", RFC 2168, June 1997.

[15] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[16] Daigle, L., van Gulik, D., Iannella, R. and P. Falstrom, "URN Namespace Definition Mechanisms", RFC 2611, BCP 33, June 1999.

[17] Petke, R. and I. King, "Registration Procedures for URL Scheme Names", RFC 2717, BCP 35, November 1999.

[18] Mealling, M. and R. Daniel, "The Naming Authority Pointer (NAPTR) DNS Resource Record", RFC 2915, August 2000.

Appendix A. Pseudo Code

For the edification of implementers, pseudocode for a client routine using NAPTRs is given below. This code is provided merely as a convenience, it does not have any weight as a standard way to process NAPTR records. Also, as is the case with pseudocode, it has never been executed and may contain logical errors. You have been warned.

```
//
// findResolver(URN)
// Given a URN, find a host that can resolve it.
//
findResolver(string URN) {
  // prepend prefix to ".urn.arpa."
  sprintf(key, "%s.urn.arpa.", extractNS(URN));
  do {
    rewrite_flag = false;
    terminal = false;
    if (key has been seen) {
      quit with a loop detected error
    }
    add key to list of "seens"
    records = lookup(type=NAPTR, key); // get all NAPTR RRs for 'key'

    discard any records with an unknown value in the "flags" field.
    sort NAPTR records by "order" field and "preference" field
        (with "order" being more significant than "preference").
    n_naptrs = number of NAPTR records in response.
    curr_order = records[0].order;
    max_order = records[n_naptrs-1].order;

    // Process current batch of NAPTRs according to "order" field.
    for (j=0; j < n_naptrs && records[j].order <= max_order; j++) {
      if (unknown_flag) // skip this record and go to next one
         continue;
      newkey = rewrite(URN, naptr[j].replacement, naptr[j].regexp);
      if (!newkey) // Skip to next record if the rewrite didn't
         match continue;
      // We did do a rewrite, shrink max_order to current value
      // so that delegation works properly
      max_order = naptr[j].order;
      // Will we know what to do with the protocol and services
      // specified in the NAPTR? If not, try next record.
      if(!isKnownProto(naptr[j].services)) {
        continue;
      }
      if(!isKnownService(naptr[j].services)) {
        continue;
```

```
        }

        // At this point we have a successful rewrite and we will
        // know how to speak the protocol and request a known
        // resolution service. Before we do the next lookup, check
        // the flags to see if we're done.
        // Note: it is possible to rewrite this so that this valid
        // record could be noted as such but continue on in order
              // to find a 'better' record. But that code would be to
        // voluminous and application specific to be illustrative.
        if (strcasecmp(flags, "S")
         || strcasecmp(flags, "P"))
         || strcasecmp(flags, "A")) {
           terminal = true;
           services = naptr[j].services;
           addnl = any SRV and/or A records returned as additional
                   info for naptr[j].
        }
        key = newkey;
        rewriteflag = true;
        break;
      }
    } while (rewriteflag && !terminal);

    // Did we not find our way to a resolver?
    if (!rewrite_flag) {
       report an error
       return NULL;
    }

    // Leave rest to another protocol?
    if (strcasecmp(flags, "P")) {
       return key as host to talk to;
    }

    // If not, keep plugging
    if (!addnl) { // No SRVs came in as additional info, look them up
      srvs = lookup(type=SRV, key);
    }

    sort SRV records by preference, weight, ...
    for each (SRV record) { // in order of preference
      try contacting srv[j].target using the protocol and one of the
          resolution service requests from the "services" field of the
          last NAPTR record.
      if (successful)
        return (target, protocol, service);
        // Actually we would probably return a result, but this
```

```
        // code was supposed to just tell us a good host to talk to.
     }
     die with an "unable to find a host" error;
   }
```

Author's Address

Michael Mealling
VeriSign
21345 Ridgetop Circle
Sterling, VA 20166
US

EMail: michael@neonym.net
URI: http://www.verisignlabs.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.