Network Working Group
Internet-Draft
Expires: December 28, 2004

L. Daigle
A. Newton
VeriSign, Inc.

June 29, 2004

# Domain-based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)
**draft-daigle-snaptr-01.txt**

Abstract

This memo defines a generalized mechanism for application service naming that allows service location without relying on rigid domain

naming conventions (so-called name hacks). The proposal defines a Dynamic Delegation Discovery System (DDDS) Application to map domain name, application service name, and application protocol to target server and port, dynamically.

[Note to be removed for RFC publication: this work was originally referred to as "napstr", and draft-daigle-napstr-04 is the immediate precursor of draft-daigle-snaptr-00.]

Table of Contents

## 1. Introduction

This memo defines a generalized mechanism for application service naming that allows service location without relying on rigid domain naming conventions (so-called name hacks). The proposal defines a Dynamic Delegation Discovery System (DDDS -- see [4]) Application to map domain name, application service name, and application protocol to target server and port, dynamically.

As discussed in Section 5, existing approaches to using DNS records to dynamically determining the current host for a given application service are limited in terms of the use cases supported. To address some of the limitations, this document defines a DDDS Application to map service+protocol+domain to specific server addresses using both NAPTR [5] and SRV ([3]) DNS resource records. This can be viewed as a more general version of the use of SRV and/or a very restricted application of the use of NAPTR resource records.

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 RFC2119 ([1]).

## 2. Straightforward-NAPTR (S-NAPTR) Specification

The precise details of the specification of this DDDS application are given in Section 6. This section defines the usage of the DDDS application.

### 2.1 Key Terms

An "application service" is a generic term for some type of application, indpendent of the protocol that may be used to offer it. Each application service will be associated with an IANA-registered tag. For example, retrieving mail is a type of application service, which can be implemented by different application-layer protocols (e.g., POP3, IMAP4). A tag, such as "RetMail", could be registered for it. (N.B.: this has not been done, and there are no plans to do so at the time of this writing).

An "application protocol" is used to implement the application service.  These are also associated with IANA-registered tags.  Using the mail example above, "POP3" and "IMAP4" could be registered as application protocol tags.  In the case where multiple transports are available for the application, separate tags should be defined for each transport.

The intention is that the combination of application service and protocol tags should be specific enough that finding a known pair (e.g., "RetMail:POP3" is sufficient for a client to identify a server with which it can communicate.

Some protocols support multiple application services. For example, LDAP is an application protocol, and can be found supporting various services (e.g., "whitepages", "directory enabled networking", etc).

## 2.2 S-NAPTR DDDS Application Usage

As defined in Section 6, NAPTR records are used to store application service+protocol information for a given domain. Following the DDDS standard, these records are looked up, and the rewrite rules (contained in the NAPTR records) are used to determine the successive DNS lookups, until a desirable target is found.

For the rest of this section, refer to the set of NAPTR resource records for example.com shown in the figure below, where "WP" is the imagined application service tag for "white pages", and "EM" is the application service tag for an imagined "Extensible Messaging" application service.

```
example.com.
;;       order pref flags
IN NAPTR 100   10   ""     "WP:whois++"      ( ; service
                           ""                  ; regexp
                           bunyip.example.     ; replacement
                                             )
IN NAPTR 100   20   "s"    "WP:ldap"         ( ; service
                           ""                  ; regexp
                          _ldap._tcp.myldap.example.com. ; replacement
                                             )
IN NAPTR 200   10   ""     "EM:protA"        ( ; service
                           ""                  ; regexp
                           someisp.example.    ; replacement
                                             )
IN NAPTR 200   30   "a"    "EM:protB"          ; service
                           ""                  ; regexp
                           myprotB.example.com.; replacement
                                             )
```

### 2.2.1 Ordering and Preference

A client retrieves all of the NAPTR records associated with the target domain name (example.com, above). These are to be sorted in terms of increasing ORDER, and increasing PREF within each ORDER.

### 2.2.2 Matching and non-Matching NAPTR Records

Starting with the first sorted NAPTR record, the client examines the SERVICE field to find a match. In the case of the S-NAPTR DDDS application, that means a SERVICE field that includes the tags for the desired application service and a supported application protocol.

If more than one NAPTR record matches, they are processed in increasing sort order.

### 2.2.3 Terminal and Non-Terminal NAPTR Records

A NAPTR record with an empty FLAG field is "non-terminal". That is, more NAPTR RR lookups are to be performed. Thus, to process a NAPTR record with an empty FLAG field in S-NAPTR, the REPLACEMENT field is used as the target of the next DNS lookup -- for NAPTR RRs.

In S-NAPTR, the only terminal flags are "S" and "A". These are called "terminal" NAPTR lookups because they denote the end of the DDDS/NAPTR processing rules. In the case of an "S" flag, the REPLACEMENT field is used as the target of a DNS query for SRV RRs, and normal SRV processing is applied. In the case of an "A" flag, an address record is sought for the REPLACEMENT field target (and the default protocol port is assumed).

### 2.2.4 S-NAPTR and Successive Resolution

As shown in the example NAPTR RR set above, it is possible to have multiple possible targets for a single application service+protocol pair. These are to be pursued in order until a server is successfully contacted or all possible matching NAPTR records have been successively pursued to terminal lookups and servers contacted. That is, a client must backtrack and attempt other resolution paths in the case of failure.

"Failure" is declared, and backtracking must be used when

- o the designated remote server (host and port) fail to provide appropriate security credentials for the *originating* domain
- o connection to the designated remote server otherwise fails -- the

specifics terms of which are defined when an application protocol
is registered

- the S-NAPTR-designated DNS lookup fails to yield expected results -- e.g., no A RR for an "A" target, no SRV record for an "S" target, or no NAPTR record with appropriate application service and protocol for a NAPTR lookup.  Except in the case of the very first NAPTR lookup, this last is a configuration error:  the fact that example.com has a NAPTR record pointing to "bunyip.example" for the "WP:Whois++" service and protocol means the administrator of example.com believes that service exists. If bunyip.example has no "WP:Whois++" NAPTR record, the application client MUST backtrack and try the next available "WP:Whois++" option from example.com. As there is none, the whole resolution fails.

An application client first queries for the NAPTR RRs for the domain of a named application service. The application client MUST select one protocol to choose The PREF field of the NAPTR RRs may be used by the domain administrator to The first DNS query is for the NAPTR RRs in the original target domain (example.com, above).

### 2.2.5 Clients Supporting Multiple Protocols

In the case of an application client that supports more than one protocol for a given application service, it MUST pursue S-NAPTR resolution completely for one protocol, exploring all potential terminal lookups in PREF and ORDER ranking, until the application connects successfully or there are no more possibilities for that protocol.

That is, what the client MUST NOT do is start looking for one protocol, observe that a successive NAPTR RR set supports another of its preferred protocols, and continue the S-NAPTR resolution based on that protocol. For example, even if someisp.example offers the "EM" service with protocol "ProtB", there is no reason to believe it does so on behalf of example.com (since there is no such pointer in example.com's NAPTR RR set).

It MAY choose which protocol to try first based on its own preference, or from the PREF ranking in the first set of NAPTR records (i.e., those for the target named domain). However, the chosen protocol MUST be listed in that first NAPTR RR set.

It MAY choose to run simultaneous DDDS resolutions for more than one protocol, in which case the requirements above apply for each protocol independently. That is, do not switch protocols mid-resolution.

## 3. Guidelines

## 3.1 Guidelines for Application Protocol Developers

The purpose of S-NAPTR is to provide application standards developers with a more powerful framework (than SRV RRs alone) for naming service targets, without requiring each application protocol (or service) standard to define a separate DDDS application.

Note that this approach is intended specifically for use when it makes sense to associate services with particular domain names (e.g., e-mail addresses, SIP addresses, etc). A non-goal is having all manner of label mapped into domain names in order to use this.

Specifically not addressed in this document is how to select the domain for which the service+protocol is being sought. It is up to other conventions to define how that might be used (e.g., new messaging standards can define what domain to use from their URIs, how to step down from foobar.example.com to example.com, and so on, if that is applicable).

Although this document proposes a DDDS application that does not use all the features of NAPTR resource records, it does not mean to imply that DNS resolvers should fail to implement all aspects of the NAPTR RR standard. A DDDS application is a client use convention.

The rest of this section outlines the specific elements that protocol developers must determine and document in order to make use of S-NAPTR.

### 3.1.1 Registration of application service and protocol tags

Application protocol developers that wish to make use of S-NAPTR must make provision to register any relevant application service and application protocol tags, as described in Section 7.

### 3.1.2 Definition of conditions for retry/failure

One other important aspect that must be defined is the expected behaviour for interacting with the servers that are reached via S-NAPTR. Specifically, under what circumstances should the client retry a target that was found via S-NAPTR? What should it consider a failure that causes it to return to the S-NAPTR process to determine the next serviceable target (a less preferred target)?

For example, if the client gets a "connection refused" from a server, should it retry for some (protocol-dependent) period of time? Or, should it try the next-preferred target in the S-NAPTR chain of resolution? Should it only try the next-preferred target if it receives a protocol-specific permanent error message?

The most important thing is to select one expected behaviour and document it as part of the use of S-NAPTR.

As noted earlier, failure to provide appropriate credentials to identify the server as being authoritative for the original taret domain is always considered a failure condition.

### 3.1.3 Server identification and handshake

As noted in Section 8, use of the DNS for server location increases the importance of using protocol-specific handshakes to determine and confirm the identity of the server that is eventually reached.

Therefore, application protocol developers using S-NAPTR should identify the mechanics of the expected identification handshake when the client connects to a server found through S-NAPTR.

## 3.2 Guidelines for Domain Administrators

Although S-NAPTR aims to provide a "straightforward" application of DDDS and use of NAPTR records, it is still possible to create very complex chains and dependencies with the NAPTR and SRV records.

Therefore, domain administrators are called upon to use S-NAPTR with as much restraint as possible, while still achieving their service design goals.

The complete set of NAPTR, SRV and A RRs that are "reachable" through the S-NAPTR process for a particular application service can be thought of as a "tree". Each NAPTR RR retrieved points to more NAPTR or SRV records; each SRV record points to several A record lookups. Even though a particular client can "prune" the tree to use only those records referring to application protocols supported by the client, the tree could be quite deep, and retracing the tree to retry other targets can become expensive if the tree has many branches.

Therefore,

- o Fewer branches is better: for both NAPTR and SRV records, provide different targets with varying preferences where appropriate (e.g., to provide backup services, etc), but don't look for reasons to provide more.
- o Shallower is better: avoid using NAPTR records to "rename" services within a zone. Use NAPTR records to identify services hosted elsewhere (i.e., where you cannot reasonably provide the SRV records in your own zone).

## 3.3 Guidelines for Client Software Writers

To properly understand DDDS/NAPTR, an implementor must read [4]. However, the most important aspect to keep in mind is that, if one target fails to work for the application, it is expected that the application will continue through the S-NAPTR tree to try the (less preferred) alternatives.

## 4. Illustrations

### 4.1 Use Cases

The basic intended use cases for which S-NAPTR has been developed are:

- o Service discovery within a domain. For example, this can be used to find the "authoritative" server for some type of service within a domain (see the specific example in Section 4.2).
- o Multiple protocols. This is increasingly common as new application services are defined. This includes the case of extensible messaging (a hypothetical service) which can be offered with multiple protocols (see Section 4.3).
- o Remote hosting. Each of the above use cases applies within the administration of a single domain. However, one domain operator may elect to engage another organization to provide an application service. See Section 4.4 for an example that cannot be served by SRV records alone.

### 4.2 Service Discovery within a Domain

There are occasions when it is useful to be able to determine the "authoritative" server for a given application service within a domain. This is "discovery", because there is no a priori knowledge as to whether or where the service is offered; it is therefore important to determine the location and characteristics of the offered service.

For example, there is growing discussion of having a generic mechanism for locating the keys or certificates associated with particular application (servers) operated in (or for) a particular domain. Here's a hypothetical case for storing application key or certificate data for a given domain. The premise is that some credentials registry (CredReg) service has been defined to be a leaf node service holding the keys/certs for the servers operated by (or for) the domain. Furthermore, it is assumed that more than one protocol is available to provide the service for a particular domain. This DDDS-based approach is used to find the CredReg server that holds the information.

Thus, the set of NAPTR records for thinkingcat.example might look like this:

```
   thinkingcat.example.
   ;;       order pref flags
   IN NAPTR 100   10   ""    "CREDREG:ldap:iris.beep"   ( ; service
                             ""                           ; regexp
                             theserver.thinkingcat.example. ; replacement
```

```
                                                              )
```

Note that another domain, offering the same application service, might offer it using a different set of application protocols:

```
anotherdomain.example.
;;       order pref flags
IN NAPTR 100   10   ""    "CREDREG:iris.lwz:iris.beep"  ( ; service
                          ""                              ; regexp
                          foo.anotherdomain.example.      ; replacement
                                                        )
```

## 4.3 Multiple Protocols

A hypothetical application service, extensible messaging, will be used for the purpose of illustration. (For an example of a real application service with multiple protocols, see [9] and [10]). Assuming that "EM" was registered as an application service, this DDDS application could be used to determine the available services for delivering to a target.

Two particular features of this hypothetical extensible messaging should be noted:

1. gatewaying is expected to bridge communications across protocols
2. extensible messaging servers are likely to be operated out of a different domain than the extensible messaging address, and servers of different protocols may be offered by independent organizations

For example, "thinkingcat.example" may support its own servers for the "ProtA" extensible messaging protocol, but rely on outsourcing from "example.com" for "ProtC" and "ProtB" servers.

Using this DDDS-based approach, thinkingcat.example can indicate a preference ranking for the different types of servers for the extensible messaging service, and yet the out-sourcer can independently rank the preference and ordering of servers. This independence is not achievable through the use of SRV records alone.

Thus, to find the EM services for thinkingcat.example, the NAPTR records for thinkingcat.example are retrieved:

```
   thinkingcat.example.
   ;;   order pref flags
   IN NAPTR 100  10   "s"   "EM:ProtA"                 (   ; service
                            ""                             ; regexp
                           _ProtA._tcp.thinkingcat.example. ; replacement
                                                       )
   IN NAPTR 100  20   "s"   "EM:ProtB"                 (   ; service
                            ""                             ; regexp
                            _ProtB._tcp.example.com.       ; replacement
                                                       )
   IN NAPTR 100  30   "s"   "EM:ProtC"                 (   ; service
                            ""                             ; regexp
                            _ProtC._tcp.example.com.       ; replacement
                                                       )
```

and then the administrators at example.com can manage the preference rankings of the servers they use to support the ProtB service:

```
   _ProtB._tcp.example.com.
    ;;    Pref Weight Port  Target
   IN SRV 10    0     10001 bigiron.example.com.
   IN SRV 20    0     10001 backup.em.example.com.
   IN SRV 30    0     10001 nuclearfallout.australia-isp.example.
```

## 4.4 Remote Hosting

In the Instant Message hosting example in Section 4.3, the service owner (thinkingcat.example) had to host pointers to the hosting service's SRV records in the thinkingcat.example domain.

A better way to approach this is to have one NAPTR RR in the thinkingcat.example domain pointing to all the hosted services, and the hosting domain has NAPTR records for each service to map them to whatever local hosts it chooses (and may change from time to time).

```
   thinkingcat.example.
   ;;      order pref flags
   IN NAPTR 100  10   "s"   "EM:ProtA"               ( ; service
                            ""                         ; regexp
                           _ProtA._tcp.thinkingcat.example. ; replacement
                                                     )
```

```
      IN NAPTR 100  20   ""    "EM:ProtB:ProtC"          ( ; service
                               ""                          ; regexp
                               thinkingcat.example.com.    ; replacement
                                                         )
```

and then the administrators at example.com can break out the individual application protocols and manage the preference rankings of the servers they use to support the ProtB service (as before):

```
thinkingcat.example.com.
;;      order pref flags
IN NAPTR 100  10   "s"    "EM:ProtC"                ( ; service
                          ""                          ; regexp
                          _ProtC._tcp.example.com.    ; replacement
                                                    )
IN NAPTR 100  20   "s"    "EM:ProtB"                ( ; service
                          ""                          ; regexp
                          _ProtB._tcp.example.com.    ; replacement
                                                    )
```

```
_ProtC._tcp.example.com.
 ;;    Pref Weight Port  Target
IN SRV 10    0     10001 bigiron.example.com.
IN SRV 20    0     10001 backup.em.example.com.
IN SRV 30    0     10001 nuclearfallout.australia-isp.example.
```

## 4.5 Sets of NAPTR RRs

Note that the above sections assumed that there was one service available (via S-NAPTR) per domain. Often, that will not be the case. Assuming thinkingcat.example had the CredReg service set up as described in Section 4.2 and the extensible messaging service set up as described in Section 4.4, then a client querying for the NAPTR RR set from thinkingcat.com would get the following answer:

```
thinkingcat.example.
;;       order pref flags
IN NAPTR 100   10   "s"    "EM:ProtA"               ( ; service
                           ""                         ; regexp
                           _ProtA._tcp.thinkingcat.example. ; replacement
                                                    )
IN NAPTR 100   20   ""     "EM:ProtB:ProtC"         ( ; service
                           ""                         ; regexp
                           thinkingcat.example.com.   ; replacement
                                                    )
```

```
   IN NAPTR 200   10   ""    "CREDREG:ldap:iris-beep" ( ; service
                             ""                         ; regexp
                             bouncer.thinkingcat.example. ; replacement
                                                         )
```

Sorting them by increasing "ORDER", the client would look through the SERVICE strings to determine if there was a NAPTR RR that matched the application service it was looking for, with an application protocol it could use.  The first (lowest PREF) record that so matched is the one the client would use to continue.

## 4.6  Sample sequence diagram

Consider the example in Section 4.3.  Visually, the sequence of steps required for the client to reach the final server for a "ProtB" service for EM for the thinkingcat.example domain is as follows:

```
   Client   NS for                NS for
            thinkingcat.example   example.com    backup.em.example.com
                |                     |                  |
     1 -------->|                     |                  |
     2 <--------|                     |                  |
     3 ------------------------------>|                  |
     4 <------------------------------|                  |
     5 ------------------------------>|                  |
     6 <------------------------------|                  |
     7 ------------------------------>|                  |
     8 <------------------------------|                  |
     9 ------------------------------------------------->|
    10 <-------------------------------------------------|
    11 ------------------------------------------------->|
    12 <-------------------------------------------------|
   (...)
```

1.  the name server (NS) for thinkingcat.example is reached with a request for all NAPTR records
2.  the server responds with the NAPTR records shown in Section 4.3.
3.  the second NAPTR record matches the desired criteria; that has an "s" flag and a replacement fields of "_ProtB._tcp.example.com". So, the client looks up SRV records for that target, ultimately making the request of the NS for example.com.
4.  the response includes the SRV records listed in Section 4.3.
5.  the client attempts to reach the server with the lowest PREF in the SRV list -- looking up the A record for the SRV record's target (bigiron.example.com).
6.  the example.com NS responds with an error message -- no such

machine!

7. the client attempts to reach the second server in the SRV list, and looks up the A record for backup.em.example.com

8. the client gets the A record with the IP address for backup.em.example.com from example.com's NS.
9. the client connects to that IP address, on port 10001 (from the SRV record), using ProtB over tcp.
10. the server responds with an "OK" message.
11. the client uses ProtB to challenge that this server has credentials to operate the service for the original domain (thinkingcat.example)
12. the server responds, and the rest is EM.

## 5. Motivation and Discussion

Increasingly, application protocol standards are using domain names to identify server targets, and stipulating that clients should look up SRV resource records to determine the host and port providing the server. This enables a distinction between naming an application service target and actually hosting the server. It also increases flexibility in hosting the target service:
- o the server may be operated by a completely different organization without having to list the details of that organization's DNS setup (SRVs)
- o multiple instances can be set up (e.g., for load balancing or secondaries)
- o it can be moved from time to time without disrupting clients' access, etc.

This is quite useful, but Section 5.1 outlines some of the limitations inherent in the approach.

That is, while SRV records can be used to map from a specific service name and protocol for a specific domain to a specific server, SRV records are limited to one layer of indirection, and are focused on server administration rather than on application naming. And, while the DDDS specification and use of NAPTR allows multiple levels of redirection before locating the target server machine with an SRV record, this proposal requires only a subset of NAPTR strictly bound to domain names, without making use of the REGEXP field of NAPTR. These restrictions make the client's resolution process much more predictable and efficient than with some potential uses of NAPTR records. This is dubbed "S-NAPTR" -- a "S"traightforward use of NAPTR records.

### 5.1 So, why not just SRV records?

An expected question at this point is: this is so similar in structure to SRV records, why are we doing this with DDDS/NAPTR?

Limitations of SRV include:

- o SRV provides a single layer of indirection -- the outcome of an SRV lookup is a new domain name for which the A RR is to be found.
- o the purpose of SRV is focused on individual server administration, not application naming: as stated in [3] "The SRV RR allows administrators to use several servers for a single domain, to move services from host to host with little fuss, and to designate some hosts as primary servers for a service and others as backups."
- o target servers by "service" (e.g., "ldap") and "protocol" (e.g., "tcp") in a given domain. The definition of these terms implies specific things (e.g., that protocol should be one of UDP or TCP) without being precise. Restriction to UDP and TCP is insufficient for the uses described here.

The basic answer is that SRV records provide mappings from protocol names to host and port. The use cases described herein require an additional layer -- from some service label to servers that may in fact be hosted within different administrative domains. We could tweak SRV to say that the next lookup could be something other than an address record, but that is more complex than is necessary for most applications of SRV.

## 5.2 So, why not just NAPTR records?

That's a trick question. NAPTR records cannot appear in the wild -- see [4]. They must be part of a DDDS application.

The purpose here is to define a single, common mechanism (the DDDS application) to use NAPTR when all that is desired is simple DNS-based location of services. This should be easy for applications to use -- some simple IANA registrations and it's done.

Also, NAPTR has very powerful tools for expressing "rewrite" rules. That power (==complexity) makes some protocol designers and service administrators nervous. The concern is that it can translate into unintelligible, noodle-like rule sets that are difficult to test and administer.

This proposed DDDS application specifically uses a subset of NAPTR's abilities. Only "replacement" expressions are allowed, not "regular expressions".

## 6. Formal Definition of <Application Service Location> Application of DDDS

This section formally defines the DDDS application, as described in [4].

## 6.1 Application Unique String

The Application Unique String is domain label for which an authoritative server for a particular service is sought.

## 6.2 First Well Known Rule

The "First Well Known Rule" is identity -- that is, the output of the rule is the Application Unique String, the domain label for which the authoritative server for a particular service is sought.

## 6.3 Expected Output

The expected output of this Application is the information necessary to connect to authoritative server(s) (host, port, protocol) for an application service within a given a given domain.

## 6.4 Flags

This DDDS Application uses only 2 of the Flags defined for the URI/URN Resolution Application ([6]): "S" and "A". No other Flags are valid.

Both are for terminal lookups. 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 label for which one or more SRV [3] records exist. "A" means that the output of the Rule is a domain name and should be used to lookup address records for that domain.

Consistent with the DDDS algorithm, if the Flag string is empty the next lookup is for another NAPTR record (for the replacement target).

## 6.5 Service Parameters

Service Parameters for this Application take the form of a string of characters that follow this ABNF ([2]):

```
      service-parms = [ [app-service] *(":" app-protocol)]
      app-service   = experimental-service  / iana-registered-service
      app-protocol  = experimental-protocol / iana-registered-protocol
      experimental-service      = "x-" 1*30ALPHANUMSYM
      experimental-protocol     = "x-" 1*30ALPHANUMSYM
      iana-registered-service   = ALPHA *31ALPHANUMSYM
      iana-registered-protocol  = ALPHA *31ALPHANUM
      ALPHA         =  %x41-5A / %x61-7A   ; A-Z / a-z
      DIGIT         =  %x30-39 ; 0-9
      SYM           =  %x2B / %x2D / %x2E  ; "+" / "-" / "."
      ALPHANUMSYM   =  ALPHA / DIGIT / SYM
      ; The app-service and app-protocol tags are limited to 32
      ; characters and must start with an alphabetic character.
      ; The service-parms are considered case-insensitive.
```

Thus, the Service Parameters may consist of an empty string, just an app-service, or an app-service with one or more app-protocol specifications separated by the ":" symbol.

Note that this is similar to, but not the same as the syntax used in the URI DDDS application ([6]). The DDDS DNS database requires each DDDS application to define the syntax of allowable service strings. The syntax here is expanded to allow the characters that are valid in any URI scheme name (see [8]). Since "+" (the separator used in the RFC3404 service parameter string) is an allowed character for URI scheme names, ":" is chosen as the separator here.

#### 6.5.1 Application Services

The "app-service" must be an IANA-registered service; see Section 7 for instructions on registering new application service tags.

#### 6.5.2 Application Protocols

The protocol identifiers that are valid for the "app-protocol" production are standard, registered protocols; see Section 7 for instructions on registering new application protocol tags.

### 6.6 Valid Rules

Only substitution Rules are permitted for this application. That is, no regular expressions are allowed.

### 6.7 Valid Databases

At present only one DDDS Database is specified for this Application. [5] 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 First Well Known Rule produces a domain name, and this is the Key that is used for the first lookup -- the NAPTR records for that domain are requested.

DNS servers MAY interpret Flag values and use that information to include appropriate NAPTR, SRV or 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 [5] for more information on NAPTR records and the Additional Information section of a DNS response packet.

## 7. IANA Considerations

This document calls for 2 IANA registries: one for application service tags, and one for application protocol tags.

### 7.1 Application Service Tag IANA Registry

IANA is to establish and maintain a registry for S-NAPTR Application Service Tags, listing at least the following information for each such tag:

- o Application Service Tag: a string conformant with the iana-registered-service defined in Section 6.5.
- o Defining publication: the RFC used to define the Application Service Tag, as defined in the registration process, below.

An initial Application Service Tag registration is contained in [9].

### 7.2 Application Protocol Tag IANA Registry

IANA is to establish and maintain a registry for S-NAPTR Application Protocol Tags, listing at least the following information for each such tag:

- o Application Protocol Tag: a string conformant with the iana-registered-protocol defined in Section 6.5.
- o Defining publication: the RFC used to define the Application Protocol Tag, as defined in the registration process, below.

An initial Application Protocol Tag registration is defined in [10].

## 7.3 Registration Process

All application service and protocol tags that start with "x-" are considered experimental, and no provision is made to prevent duplicate use of the same string.  Use them at your own risk.

All other application service and protocol tags are registered based on the "specification required" option defined in [7], with the further stipulation that the "specification" is an RFC (of any category).

There are no further restrictions placed on the tags other than that they must conform with the syntax defined below (Section 6.5).

The defining RFC must clearly identify and describe, for each tag being registered:

- o Application protocol or service tag
- o Intended usage
- o Interoperability considerations
- o Security considerations (see Section 8 of this document for further discussion of the types of considerations that are applicable)
- o Any relevant related publications

## 8. Security Considerations

The security of this approach to application service location is only as good as the security of the DNS servers along the way. If any of them is compromised, bogus NAPTR and SRV records could be inserted to redirect clients to unintended destinations. This problem is hardly unique to S-NAPTR (or NAPTR in general). A full discussion of the security threats pertaining to DNS can be found in [11].

To protect against DNS-vectored attacks, secured DNS (DNSSEC) [12] can be used to ensure the validity of the DNS records received.

Whether or not DNSSEC is used, applications should define some form of end-to-end authentication to ensure that the correct destination has been reached. Many application protocols such as HTTPS, BEEP, IMAP, etc... define the necessary handshake mechansims to accomplish this task. Newly-defined application protocols should take this into consideration and incorporate appropriate mechanisms.

The basic mechanism works in the following way:

1. During some portion of the protocol handshake, the client sends to the server the original name of the desired destination (i.e. no transformations that may have resulted from NAPTR

replacements, SRV targets, or CNAME changes). In certain cases where the application protocol does not have such a feature but TLS may be used, it is possible to use the "server_name" TLS extension.

2. The server sends back to the client a credential with the appropriate name. For X.509 certificates, the name would either be in the subjectDN or subjectAltName fields. For Kerberos, the

name would be a service principle name.

3. Using the matching semantics defined by the application protocol, the client compares the name in the credential with the name sent to the server.
4. If the names match and the credentials have integrity, there is reasonable assurance that the correct end point has been reached.
5. The client and server establish an integrity-protected channel.

It is important to note that this document does not define either the handshake mechanism, the specific credential naming fields, nor the name matching semantics. Definitions of S-NAPTR for particular application protocols MUST define these.

## 9. Acknowledgements

Many thanks to Dave Blacka, Patrik Faltstrom, Sally Floyd, and Ted Hardie for discussion and input that has (hopefully!) provoked clarifying revisions of this document.

## 10. References

### 10.1 Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

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

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

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

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

[7]  Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

### 10.2 Informative References

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

[9] Newton, A. and M. Sanz, "IRIS Domain Registry Schema", draft-ietf-crisp-iris-dreg-06 (work in progress), April 2004.

[10] Newton, A. and M. Sanz, "Using the Internet Registry Information Service (IRIS) over the Blocks  Extensible Exchange Protocol (BEEP)", draft-ietf-crisp-iris-beep-04 (work in progress), April 2004.

[11] Atkins, D. and R. Austein, "Threat Analysis Of The Domain Name System", draft-ietf-dnsext-dns-threats-07 (work in progress), April 2004.

[12] Arends, R., Larson, M., Austein, R. and D. Massey, "Protocol Modifications for the DNS Security Extensions", draft-ietf-dnsext-dnssec-protocol-06 (work in progress), May 2004.

Authors' Addresses

Leslie Daigle
VeriSign, Inc.
21355 Ridgetop Circle
Dulles, VA  20166
US

EMail: leslie@verisignlabs.com; leslie@thinkingcat.com

Andrew Newton
VeriSign, Inc.
21355 Ridgetop Circle
Dulles, VA  20166
US

EMail: anewton@verisignlabs.com

## Appendix A. Pseudo pseudocode for S-NAPTR

### A.1 Finding the first (best) target

Assuming the client supports 1 protocol for a particular application service, the following pseudocode outlines the expected process to find the first (best) target for the client, using S-NAPTR.

```
 target = [initial domain]
 naptr-done = false

 while (not naptr-done)
  {
   NAPTR-RRset = [DNSlookup of NAPTR RRs for target]
   [sort NAPTR-RRset by ORDER, and PREF within each ORDER]
   rr-done = false
   cur-rr = [first NAPTR RR]

   while (not rr-done)
      if ([SERVICE field of cur-rr contains desired application
           service and application protocol])
         rr-done = true
         target= [REPLACEMENT target of NAPTR RR]
      else
         cur-rr = [next rr in list]

      if (not empty [FLAG in cur-rr])
         naptr-done = true
  }

 port = -1

 if ([FLAG in cur-rr is "S"])
  {
   SRV-RRset = [DNSlookup of SRV RRs for target]
   [sort SRV-RRset based on PREF]
   target = [target of first RR of SRV-RRset]
   port = [port in first RR of SRV-RRset]
  }

 ; now, whether it was an "S" or an "A" in the NAPTR, we
 ; have the target for an A record lookup

 host = [DNSlookup of target]
```

```
    return (host, port)
```

## A.2 Finding subsequent targets

The pseudocode in Appendix A is crafted to find the first, most preferred, host-port pair for a particular application service an protocol. If, for any reason, that host-port pair did not work (connection refused, application-level error), the client is expected to try the next host-port in the S-NAPTR tree.

The pseudocode above does not permit retries -- once complete, it sheds all context of where in the S-NAPTR tree it finished. Therefore, client software writers could

- o entwine the application-specific protocol with the DNS lookup and RRset processing described in the pseudocode and continue the S-NAPTR processing if the application code fails to connect to a located host-port pair;
- o use callbacks for the S-NAPTR processing;
- o use an S-NAPTR resolution routine that finds *all* valid servers for the required application service and protocol from the originating domain, and provides them in sorted order for the application to try in order.

## Appendix B. Availability of Sample Code

Sample Python code for S-NAPTR resolution is available from http://www.verisignlabs.com/pysnaptr-0.1.tgz .

## Intellectual Property Statement

## Full Copyright Statement

## Acknowledgment

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