Network Working Group
Internet-Draft
Expires: December 29, 2002

M. Mealling
L. Daigle
VeriSign, Inc.
June 30, 2002

Service Lookup System (SLS)
draft-mealling-sls-02.txt

Status of this Memo

Copyright Notice

Abstract

<1>
Developing technology to allow for truly internationalized Internet identifiers is proving a hard nut to crack within the framework of the existing DNS. At the same time, the DNS continues to do an excellent job at serving its original mandate for providing efficient mappings between machine-readable labels and network resources. What is not clear is whether the existing DNS can be transformed into a service that can handle the more human oriented identification services it is now being asked to provide. This document embraces, extends and complements a proposal by John Klensin to address the requirements for a directory layer above the existing DNS that can

better solve these problems. The discussion concludes by proposing a strawman called the Service Lookup System (SLS).

Table of Contents

1. Introduction

<2>

In "A Search-based access model for the DNS" [I-D.klensin-dns-search], the author discusses approaching the problems of international domain-names and enhanced DNS with a layered approach that leaves the current DNS' form and function unmodified. The three layers are:

<3>

Layer 1 -- The DNS, with the existing lookup mechanisms

<4>

Layer 2 -- A restricted lookup system where the identifiers are qualified by additional attributes called facets. Facets include concepts such as locale, category and language.

<5>

Layer 3 -- Localized and topic-specific search environments.

This memo describes the problem statement, reviews intended usage scenarios, provides a straw proposal for implementing a Layer 2 service, and discusses the rational that ties these elements together.

2. The Problem Statement

<6>

Roughly stated, the goal of Layer 1 is to provide unique, machine friendly identifiers for network level resources that can be used as protocol elements. Layer 3 is for search services such as search engines (Google) and localized/topic specific directory services (LDAP); e.g. very human and/or task specific services where the queries and results are not universally standardizable. Layer 2 attempts to be a bridge between Layer 1 and Layer 3. The problem is: what is the functional and deployable middle ground? This includes even the fundamental question of exactly what is the problem Layer 2 will attempt to solve?

<7>

Much of the discussion to date has dealt with the internationalization of Internet identifiers (specifically domain-names). For Western cultures the need for anything beyond simple matches on characters is not immediately apparent. Since the Internet, and DNS specifically, were designed using Western characters, it is much easier for Western speakers to learn to live with the limitations and thus those limitations aren't as glaringly apparent. But when confronted with other character sets from Asian languages, the simple "match on characters" semantic quickly becomes unworkable and in many cases fundamentally cannot address the identification requirements of the user. Requirements such as 'match based on the locale of the querier' and 'order of the name components to match user expectation' have been common enough to illustrate that the problems that are attempting to be solved are beyond DNS'

capabilities. It is exactly the work being done in the IDN Working Group that is bringing these problems to light.

<8>

It is also interesting to look at what might be the root cause of all of these problems. In the authors' opinion, many of these problems stem from the disconnect between what the DNS was meant to identify and what it is actually being used for. In many cases the DNS is being forced into service as as a way to identify complex services that have no concrete network level representation.

<9>

For example, when a user types 'cnn.com' into a web browser they are not explicitly asking for the index.html file at the root level context of the HTTP server running on the default port of the host whose A record is returned from the DNS query for 'cnn.com'. The user's view of the process is that he/she is requesting the current news from CNN via the Internet. The disconnect between these two different interpretations of the same action is where the problem lies.

<10>

The problem is that the DNS and by extension IDN and similar efforts are attempting to use a simple name to number mapping system for network identifiers as a tool for mapping real world entities (companies, individuals, services) into network services (not identifiers). Since networks are designed and evolve to meet technical and network administration needs, their evolution is often at odds with that of the services that real world entities (individuals, organizations) wish to communicate about. This stress is particularly noticeable in the identifier strings themselves (domain and host names) -- companies, individuals and services must be named using labeling conventions that were devised for network machines. This simply doesn't fit.

2.1 Requirements

<11>

The problems and features discussed [I-D.klensin-dns-search] suggest a system with certain behaviors. This document continues that discussion by proposing a specific implementation for that system. In order to do so, some of the unanswered questions in [I-D.klensin-dns-search] need to discussed:

<12>

o Character sets -- Full Unicode support at a minimum. There is some desire to enable other character sets but most comments have said that mapping into Unicode is acceptable as long as there can be some method for communicating what locale was used for doing that mapping and which normalization steps were taken. It has become evident that, in order to know what normalization steps occurred, the client may need to express what original character set was used as input into that normalization step. (It has even

been suggested that the exact software vendor and version implementing that normalization may be needed for some languages.)

<13>

- Localization -- In many cases there are semantic differences in what an appropriate match should be that are based on location, jurisdiction, or region specific dialect.

<14>

- Geographic scoping -- In other cases, it is appropriate to distinguish between identifiers based on the region or geographical scope of applicability. For example, trademarks have traditionally been scoped by geographical boundaries.

<15>

- Category based scoping -- To fully handle most trademark law and the human habit of using the same word to mean two different things, names also need to be scoped by the category they fit into. The problem here is to figure out which categories to use since there is no single taxonomy in which all things can be categorized.

<16>

- Syntactic sugar -- If at all possible, the system should not place synthetic syntactic restrictions or requirements on identifiers. One main reason is that there are no common syntactic elements among all languages. This includes both computational, structured syntax (e.g. dot separators) and no requirements or constraints on the interpretation of the identifier (e.g. any Unicode character is valid).

<17>

- DNS Characteristics worth preserving -- Since the DNS provides some of the motivation for a Layer 2 service, it is worth looking at in terms of characteristics that are worth emulating: a) limited match semantics (lookup only); b) deterministic relationship between the name and the answer set; c) all public names are globally available; and d) in the case of an A record, the result is service independent.

<18>

- Uniqueness is an important characteristic of DNS that should be emulated by some aspects of the system, though which aspects and how are uncertain. It is at least a requirement that a given name/facet set/service tuple be unique.

<19>

- There are no requirements that the names are structured

<20>

- There are requirements that facets be structured, highly standardized, limited in number and with values that come from controlled vocabularies.

<21>

- It should be possible for a result to identify a service independent network node so that the client may contact that node

for multiple services without having to re-query the Layer 2 service again and again for each different service.

<22>

- o While locale in its various standardized forms does communicate some aspects of 'location', additional information is needed in order to support various human assumptions such as trademark law and locality of reference (geographic and category scoping).

<23>

- o Entries must be globally unique, but 2 entries may be distinguishable by as little information as the service through which they are made available. In other words, names and their facets, as a whole, are unique within a service and are scoped to that service.

<24>

- o A result must return its entire context. This includes not only the name and the identification component but ALL of the facets that made up the match.

<25>

- o There are no requirements or restrictions on the entities that can be identified. A name can apply to a human, a corporation, etc. Some services may not make sense for a given entity but that it simply reflected in that name simply not begin registered with a provider for that service type.

<26>

- o It is expected that Layer 2 services will be provided on a competitive basis. This means multiple service providers that may cover the same areas and who compete directly with each other.

2.2 Service Providers

<27>

The concept of Layer 2 'service providers' has been mentioned several times so far and needs to be discussed itself. In order to avoid requiring a single, structured global delegation of registration and lookup servers, we start from the assumption that there will be multiple independent collections of name/facets. Name/facet tuples must be globally unique across all publicly accessible collections. This is accomplished by including the service provider as one of the facets; essentially making name/facet tuples unique to their provider. Beyond this there is no other defined relationship between service providers. Whether providers coordinate or compete with each other is beyond the scope of this document. The only material effect is that we need to determine whether "discovery" is a required component of the Layer 2 query protocol. There may be a requirement that a tuple have a service provider independent and globally unique identifier to allow for a tuple to 'migrate' from provider to provider but this is more of a policy requirement than a technical one.

2.3 Outstanding Questions

<28>

Questions still to be answered are:

<29>

- o Is Unicode sufficient? If not by itself then is a mapping from the local character set onto Unicode provided the mapping used is communicated to the service via the locale facet sufficient? If not, then is the requirement that _all_ character sets be supported?

<30>

- o In many cases 'locale' is a combination of pieces of information. The value associated with any Posix locale setting is a combination of the ISO 3166-1 two letter country code and a two letter language code. Is this concept of locale sufficient for the boundary cases found in some languages? Does the definition need to be augmented by ISO 3166-2 subregion codes? Are the standard two letter language codes also sufficient?

<31>

- o Is uniqueness based on the name/facet-set/service tuple sufficient?

<32>

- o If it is, is there a requirement that the results of a query be exhaustive? This requirement would create a situation where all service providers would have to be discoverable.

<33>

- o Is there a real requirement for supporting the trademark law concepts of name scoping by geographic and category boundaries? If so then requirements for the location and category facets need to be investigated further.

3. Usage Scenarios

<34>

Since it is much easier to discuss these goals in terms of specific usage scenarios instead of vague general desires, the discussion above is framed in terms of the following examples.

3.1 Transition from DNS to something else

<35>

In order to deploy anything at Layer 2, a transition method would have to be put in place to allow for users to a) use domain-names within the system and b) use Layer 2 names in place of domain-names. A usage example would look like this:

<36>

A user at a web browser wants to go to the web site for "CNN". Their browser has rudimentary software installed that can handle the term "CNN" as Layer 2 service name instead of a munged Layer 1 domain-name but only by 'acting' as a shim above that browsers Layer 1 interface.

Therefore the users query are specified so that the results are nothing more than pointers to regular DNS records. If the results contain more than one answer then the user is given the disambiguation step. The results are kept in a cache but as far as the browser is concerned, it has still received a simple 'A' record.

<37>

The same could be done for an enhanced email address. In this case a seemingly normal email address is decomposed using regular RFC 822 [RFC0822] rules. The same shim layer is called on the Right Hand Side (RHS) of the address only per RFC 822's rules. The query is for an MX or A record. If there is a disambiguation step required the user is given that choice. The result of the query will be an MX or A record that is handed back to the user's application.

<38>

This is simply a scenario. If a solution is deployed that actually enables it then extreme care must be taken so that recursive resolution doesn't happen between a full implementation of the service and this 'shim'. I.e. if the result of a full enabled client query is then input into an 'shim' then serious usability/ interoperability problems can occur.

3.2 Web Browsing

<39>

The following scenarios discuss the use of an SLS like system for naming services used via web browsers.

3.2.1 Never resolved but the name is known

<40>

One of the main uses of Layer 2 style names is that they help solve the 'guessing' function that many users are forced to do with DNS names. With DNS a guess is required because the most likely name is often already taken, causing other services to have to pick sub-optimal domain-names. This causes the user to have to guess at that sub-optimal name in order to find the other services. This scenario involves the case where the user is attempting to browse the public page of a service they have heard about but never actually visited or queried for.

<41>

In this case the user enters the Layer 2 name "McDonald's". This name, plus defaults provided by the browser that the user previously configured (location, locale, interests, etc), are sent to the users configured SLS servers. The servers respond with the various results and those results are displayed to the user. In this particular case the user lives in an area that has a locksmith business called "McDonald's Doors" as well as a computer upgrade and repair business called "McDonald's and Associates". All of these companies and the fairly famous restaurant with over a billion served both have the use of the tradename "McDonald's". The user is presented with these

results:

McDonald's - a worldwide system of restaurants which prepare,
assemble, package and sell a limited menu of value-priced
foods.
http://www.mcdonalds.com/

McDonald & Associates - current pricing on standard memory modules, a list
of proprietary upgrades available, a memory FAQ, and articles on
upgrading memory and types of memory.
http://www.buymemory.com/

McDonalds Doors - offers security, safety, and protection in doors and
locking systems.
http://www.mcdonaldsdoors.co.uk/

<42>
Based on this list of results the user selects the locksmith at which point their browser is told to request that particular URI. At the same time, that record is also inserted into the users local cache of service records.

3.2.2 The name is known and it has been resolved before

<43>
In this scenario the user is requesting the same Layer 2 name as before: "McDonald's". At this point the user interface designer has three main choices:

<44>
- o immediately follow the service description found in the first hit in the users local cache

<45>
- o re-query to the current list of SLS service providers, but displaying the users locally cached records first

<46>
- o ignore the users local cache entirely

The usability issue here is the tradeoff between the easiest way for the user to use frequently used names without needing to disambiguate yet again and allowing the user to signal that they wish to do a novel name lookup regardless of what they have done in the past. In this scenario we will assume that the browser handles this situation to the user's satisfaction. In this case the user never sees the disambiguation step and thus is immediately sent back to the same service as before.

3.2.3 The name is not known but other characteristics are known

<47>
In this scenario the user does _not_ know the actual name of the

service she is looking for but she does know that it is a locksmith in her local area. In this case the query is not for an Layer 2 name but instead is sent to a local Layer 3 service such as a local yellow pages provider. The results are sent back in the same basic form as what SLS provides but augmented with additional values that helps the user differentiate between which locksmith they're looking for. These records also contain the Layer 2 name for each service, thus populating the users local-cache with the correct names for future queries.

<48>

The key point here is that there is a general requirement that Layer 2 and Layer 3 service be interoperable to some degree.

3.3 Sending email

<49>

In these scenarios, Layer 2 names are used as components of SLS-enhanced email addresses.

3.3.1 LHS and RHS names are known

<50>

In this scenario the user has been presented with the SLS enhanced email address of a friend on their business card. The address is "Ima Sample@Example Technologies, Inc". The user enters this address into their SLS enhanced mailer which then decomposes the email address into its Right and Left Hand Side components. The RHS is sent to the same SLS providers as above and the results are provided to the user. In this case the user knows that there are several companies with that name but she is aware that this particular one is an aerospace contractor in England. In some cases the results contain a referral to an SLS service that is specific to that email service. The user picks a record that has such a referral and the mail agent then sends an SLS query for the LHS of the address to the SLS service found in that referral. That local service then sends the matches for "Ima Sample" back to the user. These records contain pointers to 'real' RFC 822 addresses that the user's mail client can actually send email to.

3.3.2 Full human address known and has been bookmarked

<51>

In the case where the address or the RHS of a previous addresss has been previously queried for the same behavior above can be inserted. In the case where the RHS is in the cache but the LHS is not the disambiguation step (if needed) would have to be done. In either case, the results are again inserted into the users local cache and used according to the user interface requirements.

4. A Strawman Proposal: The Service Lookup System (SLS)

4.1 Strawman Introduction

<52>

The strawman proposal discussed here offers several options with respect to data representation and transport. Both revolve around the desire for a small, almost DNS-like, network footprint. The data representation discussion stems from the use of XML and whether or not its possible to build small packet with it. The transport discussion looks at the use of UDP and how to deal with the probability that responses will extend beyond the typical 512 byte limit.

4.2 Network Service Record (NSR)

<53>

The earlier services vs identifier discussion illustrates that it is necessary for any Layer 2 service to return more information than simply the identifier that maps to the label. In many cases this information will be task specific due to different referral models and service types. This strawman introduces the concept of a Network Service Record (NSR) which acts as the "glue" between real world entities and network services. They do not replace the DNS in form or function. These are administrative records, containing information that will allow users to identify (recognize) real world entities and for systems to express services as opposed to simple, undecorated endpoints. They can be used on an occasional basis to obtain specific network (machine interpretable) identifiers.

<54>

NSRs are a different, higher level, concept than URIs, which are machine interpretable names and addresses providing specific identification. In fact, the network identifiers provided in the NSR are URIs.

<55>

The results of an NSR lookup may be stored in user software (e.g., bookmark lists, caches, mail address books, buddy lists). Done right, the NSR label will be interpretable by human users (perhaps even attaining the elusive goal of "human friendliness") while DNS and other network identifiers continue to evolve to meet technical needs (necessarily not being "human-friendly" to the bulk of the world's population).

<56>

The format of an NSR is undefined here since it is more likely to be dependent on the requirements of the service used to look them up. In the strawman SLS proposal below the format is evolved from the XML based ResourceDescriptor element from CNRP.

4.3 Basic NSR Semantics

<57>

The NSR contains, minimally, a label that contains the Layer 2 name. Any other elements are descriptive information such as the service's

location, language, etc. These elements are called "facets" of the network service. Additionally, the NSR contains identifiers for the specific network elements used to express the service.

<58>

NSRs are globally unique across the label AND descriptive facet data. That is, many NSRs may have the same label, if they differ in the values of other facet data.

4.4 NSR Population

<59>

NSRs are registered on an opt-in basis. An organization or individual wishing to identify their network service(s) through a particular label may register the label and associated facet information with any NSR registry service, pursuant to the uniqueness criteria mentioned above.

<60>

It is not expected that domain name holders, organizations, or individuals will register an NSR for each host name within their domain. Rather, the NSR is independent of network devices. One service (e.g., what we today know of as an HTTP server operating for a particular domain) may have several NSRs to reflect different labels for the service entity. And, that may be the only "machine" within an organizations network that has an NSR registered to identify its services. All network services are accessible through the traditional, existing network identifiers (host+port+protocol, URIs, etc).

4.5 Service Lookup System (SLS): Looking up NSRs

<61>

NSRs are the basic element of operation for the Service Lookup System in much the same way that Resource Records are the basic operating element of the DNS. A query for NSRs sent to an SLS provider consists of the following parameters:

<62>

- NSR label (required, whole string)

<63>

- NSR descriptive facets (optional, substring allowed)

<64>

- Target Service (required, from designated list of possible)

<65>

- Additional descriptive data about the user's linguistic and geographic preferences (optional)

<66>

The NSR label is required, in full, since this is a lookup service not a data mine. That being said, individual NSR directory services may apply local matching heuristics to retrieve NSRs that are "like" what the user is looking for, at their discretion, and in order to

accommodate potential difficulties in matching transcriptions. Additionally, NSR directory services may use the additional user descriptive information (language, locale, etc) to determine a match against the set of NSRs it has.

<67>

The response to an NSR lookup request will be 0 or more NSRs.

4.6 Services

<68>

SLS has the concept of a 'target service'. This is the intent or purpose that the NSR is going to be used for. This is analogous to a Resource Record (RR) Type in DNS. In the DNS the RR type is used to designate the resulting content type as well the function that content will be used for. In SLS these two functions are broken out into the 'target service' type and the result URI. The following list of target services outlines the service name and how it is expected to be used.

<69>

<70>

'dns' -- A request for a DNS record type in the form of the 'dns:' URI scheme [I-D.josefsson-dns-url]. The service facet in the query for the NSR(s) is specified in the form of 'dns:<classtype>:<querytype>'. For example, to request an MX record the service would be 'dns:1:15'.

<71>

'web' -- The request is for the URI of a web page used for browsing by a user. The result SHOULD either be a URI with the 'http' scheme or a 'dns:' URI pointing to the A record(s) for the web server.

<72>

'email' -- In general, the NSR is targeted at identifying network services as a whole. This is useful in solving today's problem of trying to support catchy phrases for identifying a corporation's main website, but is not useful for replacing e-mail addresses on business cards.

Insofar as e-mail addresses comprise identification of particulars (string on the lefthand side of the "@") at a particular service (SMTP), it is not a far stretch to think of developing a companion standard to identify particulars within a given service. That is, the NSR could be used to find the network location of an SLS service that can map the local part of the original identifier into the final NSR record(s).

Although the conventions for expressing NSR labels and the particular identifier (e.g., on a business card) are well beyond the scope of this document, consider for example:

Leslie Daigle@Le Chat Pensant

is not a valid RFC 822 address. But using an SLS service it can be mapped into one. The SLS service would provide a DNS URI that identifies either an MX or A record for the relevant SMTP service (thinkingcat.com) as well as a referral to another SLS service that can map "Leslie Daigle" to some value that is valid for that SMTP service (in this case 'leslie'), yielding 'leslie@thinkingcat.com' to be stored in an e-mail address book.

4.7 Protocol Options

<73>

This section deals with mapping the above semantics and NSR contents to actual on-the-wire protocols. All of the options below utilize the CNRP encoding of SLS as the basis for discussion since the XML found within CNRP closely (by no accident) matches SLS already.

4.7.1 Mapping the SLS onto CNRP

<74>

As part of the proposal the SLS is mapped onto the Common Name Resolution Protocol (CNRP) [I-D.ietf-cnrp]. CNRP was designed to handle services with many of the same requirements and thus makes an easy match for discussing particular aspects of the proposal. One important issue is that operational requirements may require that the XML encoding and HTTP transports be dropped in favor of something with a smaller network 'footprint'.

4.7.1.1 An Introduction to the Common Name Resolution Protocol (CNRP)

<75>

CNRP is a protocol that is encoded in XML and transported via HTTP (as mandatory to implement, other transports are valid). The basic component of CNRP is the 'Common Name'. This is the item that is being looked up. In addition to the Common Name, a query can contain Properties. Properties have names and types. A Property type is an identifier for which controlled vocabulary the value is drawn from.

<76>

CNRP general feature list includes:

<77>

- o Unicode -- While standard XML conventions allow for specifying additional language and character set values, CNRP is required to be expressed in Unicode using the encoding specified in the XML document header.

<78>

- o Referral support -- A CNRP server can send a message to the client which tells the client what server and possible dataset an answer might be found in.

<79>

- No requirements on the CN -- CNRP makes no other requirements on the CN other than being expressed in Unicode.

<80>

- No requirements on match semantics -- CNRP puts no requirements on a service provider as to what match semantics they may or may not use. The query is series of hints only. It is up to other standards to define services using CNRP that adhere to specific rules.

<81>

- Only three Properties defined -- CNRP defines the Location, Language and Category properties in addition to a process for defining new Properties.

<82>

Results within CNRP are encoded as ordered sets of either referrals, status codes or ResourceDescriptors. It is the ResourceDescriptor which is used as the encoding of the NSR. The following is an example of a ResourceDescriptor acting as an NSR returned in response to a query for the name 'Joe's Example Mart':

```
<results>
     <service id="i0">
          <serviceuri>http://sls.bar.com/</serviceuri>
     </service>
     <resourcedescriptor id="i1">
          <commonname>Joe's Example Mart</commonname>
          <id>foo.com:234364</id>
          <resourceuri>http://acme.example.com/~joe/examples/</resourceuri>
          <serviceref ref="i0" />
          <description>A purveyor of fine examples</description>
          <property name="locale" type="rfc1766">en-uk</property>
          <property name="location" type="sls">gb-ham</property>
          <property name="service">web</property>
          <property name="category" type="nice">380023</property>
     </resourcedescriptor>
</results>
```

4.7.1.2 CNRP Service Definition

4.7.1.2.1 CNRP Properties as Facets

<83>

The concept of facets is handled with CNRP properties. Properties have both a name and a type. Properties can be valid for either queries or results or both.

<84>
The location property has a new type defined that is hierarchical in nature with each level separated by a "-". The first level is taken from ISO-3166-1 two letter country codes. The second level is taken from ISO-3166-2. Third and subsequent levels are defined by the previous level. For example, the city of Lubbock, Texas would use: us-tx-lubbock.

<85>
The language property is restricted to the values found in RFC 3066 [RFC3066]

<86>
The type of the category property is 'nice' which designates the classification of goods and services found in the Nice Agreement on International Classification of Products and Services [NICE].

<87>
The service property is the type of service being requested. The list of services is made up of the complete list of DNS QTYPEs and QCLASS-es plus specific services defined in Section 4.6. The format of the service designator is defined by each service.

<88>
The source service ID is a required CNRP property but it is listed here to be sure to note that uniqueness discussed earlier includes the source of the results as one of the facets that determine uniqueness.

4.7.1.2.2 Service Object XML

<89>
SLS defines a new CNRP property called 'cnrp-service-type' which is used to notify the client that this service adheres to the SLS standard. This is why the service object doesn't actually need to define all of the SLS facets as CNRP properties.

```
<?xml version="1.0"?>
<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
"http://ietf.org/dtd/cnrp-1.0.dtd">
<cnrp>
 <results>
    <service ttl="43200">
      <serviceuri>urn:foo:bar</serviceuri>
        <servers>
          <server>
             <serveruri>http://host1.example.com:4321</serveruri>
          </server>
          <server>
             <serveruri>mailto:user@example.com</serveruri>
          </server>
        </servers>
        <description>This is the ExampleCorp SLS Service</description>
        <!-- This property means that this service is a SLS compliant -->
        <!-- This could probably be sufficient but we'll list all     -->
        <!-- of the properties anyway just for completeness           -->
             <property name="cnrp-service-type">sls</property>
             <propertyschema>
                  <propertydeclaration id="i1">
                       <propertyname>cnrp-service-type</propertyname>
                       <propertytype default="yes">iana</propertytype>
                  </propertydeclaration>
                  <!-- CNRP defines the location, language and category -->
                  <!-- properties for us                                -->
             </propertyschema>
    </service>
  </results>
</cnrp>
```


4.7.1.2.3 Contextual Uniqueness

<90>

A CNRP service MUST have one and only one answer for any COMPLETE set of facets. This includes the facet that is the service name itself. This means that essentially uniqueness of a given name is at the service level. Thus, if a query is sent to more than one service, each one may send back valid answers. These are considered different NSRs (because they differ in the service facet).

<91>

Also, if a particular facet is set to a higher level of some hierarchical value or set to a wildcard type match semantic, it is also possible to get multiple answers for the query. What this means and how applications should deal with it is up for discussion since this behavior is the one aspect of Layer 2 that directly affects

usability.

4.7.1.2.4 Results Restrictions

<92>

Results are in the form of URIs. Unlike a generic CNRP service the schemes that can be returned are explicitly defined to match the Service facet in the request. See Section 4.6 for the list of Service to Results URI matchings and the semantics of those matches.

4.7.2 Data Representation Alternatives

<93>

This section discusses the various alternatives to CNRP in terms of data representation. The reason for this discussion is that CNRP's XML content is considered rather large in comparison to the size of a DNS packet. The question is whether or no the disparity is actually worth the price paid in the size of the network footprint.

4.7.2.1 Full XML

<94>

Obviously, one alternative is to use CNRP as-is. The argument here is that the power and features of XML actually solve problems. Character set problems already have solutions. Problems with extensible schemas and expressing those schemas can be solved with various XML schema languages. There are XML based query languages, compression algorithms, databases, and translation methods (XSLT). The cost of a larger network footprint may be outweighed by the robustness and problem solving abilities of the data representation.

4.7.2.2 Compressed XML

<95>

If the value of XML is evident but the size of the responses and queries is still unacceptable, the data could be compressed. There are three approaches:

<96>

o existing, non-schema specific stream compression

<97>

o XML compression that is not directly parse-able

<98>

o XML compression that can be directly parsed via a stylesheet

<99>

The advantage of non-smart compression is that its fast and easy. The main drawback is that performance is probably not good enough for large response sets. The various XML-specific compressions techniques provide extremely good performance, especially when the compression engines are provided with a description (either XML Schema or a DTD). The best XML-specific compression methods are the ones that are still directly parse-able from their compressed form

via a stylesheet. In other words, there is no real 'compression' or 'de-compression' since the smaller form is still a proper, parse-able representation of the XML infoset.

4.7.2.3 BLOBed XML

<100>

This alternative attempts to turn the higher container elements in the XML into a binary encoding. The author chooses the Binary Low Overhead Block (BLOB) [I-D.ietf-rescap-blob] encoding but this is only for illustrative purposes. The idea here is to maintain the XML encoding of the actual data in order to preserve some of the advantages expressed in Section 4.7.2.1 without wasting bandwidth on generic, top-level, tags such as <referral> or &ltresourcedescriptor>. This method does impact the ability to extend the schema using namespaces since the overall container relationships become hardwired.

4.7.2.4 BLOB Only

<101>

This concept uses the full power of the BLOB format in order to get as close as possible to the DNS packet size. There are no XML concepts at all and all SLS components are encoded as strings and integer arguments in a various BLOB data structures. One interesting concept with this approach is that it becomes remarkably similar to a binary encoding of the same XML document. In other words, this approach and the XML compression approach are actually the same idea only slightly restated.

4.7.3 Transport Alternatives

<102>

The desire to approximate DNS' network footprint suggests that selection of the actual network transport is very important. While CNRP uses HTTP 1.1, SLS may restrict features enough that HTTP may not be the correct solution. The following discussions outline various features that may or may not be necessary.

4.7.3.1 Full HTTP

<103>

As with the full XML suggestion above, the idea is that the HTTP transport provided with CNRP solves enough problems as is that, while it is a chatty, TCP session protocol, the price is worth paying. HTTP's network impacts are well understood. Caching and proxy solutions exist. Off the shelf software exists and is easily intergrated. In many cases the transport software already exists on both the client and server platforms. Security, error reporting, character set negotiation and XML integration problems are already solved. The question to answer is how many of those features are actually needed?

4.7.3.2 Beep

<104>

The Blocks Extensible Exchange Protocol (BEEP) [RFC3080] defines a generic application protocol kernel for connection-oriented, asynchronous interactions. Its application to SLS is that it provides a transport for XML (or BLOBs) that provides most of the items from HTTP 1.1 that SLS might use without the multitude of other features that SLS would not need. While it is much tighter than HTTP, BEEP is still mapped onto TCP [RFC3081] and thus has a connection context when it may be the case that SLS is idempotent and thus does not require a connection. It is also probable that SLS would never require more than one BEEP channel. In this case BEEP may still provide much more than SLS needs.

4.7.3.3 UDP with TCP fallback

<105>

Encoding SLS for transport via a UDP packet probably gets as close as possible to DNS' network profile. The problem is that responses will tend to be large enough that they will routinely be larger than 512 bytes and possibly larger than 1500 bytes (depending on the data representation chosen). The two suggested possibilities are to follow DNS' lead and respond with a "Requery with TCP" error response in the case where a response is to large or to negotiate a maximum UDP packet size during the query. If either technique fails then the client requeries via a TCP connection.

<106>

The upside of this approach is that it follows DNS' behavior as it is today. But that is also its downside. In many cases today, DNS packet sizes are routinely blowing out the 512 byte limitation. Throw in more responses with more information and the likelihood of that occurring more often increases. There comes a point very quickly where the liklihood of requering becomes so great that it is more efficient to simply connect via TCP by default. If that becomes the case then HTTP or BEEP become just as reasonable.

4.7.3.4 UDP with reconstructed packets

<107>

Assuming that SLS is idempotent, it seems a waste to not be able to use UDP when it matches the expected use cases and requirements so well. The only issue with using UDP is how to deal with the larger than normal amounts of data. The authors propose that, assuming the other requirements remain in line with the idempotency and minimal transport requirements, SLS could be sent via a series of UDP packets, all limited to 512 bytes and which begin with a sequence number. There is no packet retransmission since the cost to the server outweighs the cost of simply re-querying. The network impact is almost identical to IP fragmentation so the network impact is minimal. Each 512 byte packet beings with a single octet that

contains the sequence number for that packet. The first packet in the sequence also contains a second octet that contains the total number of packets to expect. The last packet will be the only packet that is less than 512 bytes.

<108>

The actual encoding of the protocol into those UDP packets is largely dependent on exactly what the data representation is. If it is purely XML then some of the techniques found in BEEP's channel 0 could be used. If it is some mixture of BLOB's then the BLOB itself would become the base packet encoding method. For an example of how this might look see [I-D.ietf-rescap-rc].

4.8 SLS Example Scenarios

<109>

The following scenarios show how a few services might be used in 'real world' situations.

4.8.1 The DNS Service

<110>

The DNS SLS service is meant more as a method for moving from the currently deployed infrastructure to new, SLS based systems. Imagine an English speaking user living in Lubbock, Texas who is attempting to browse the CNN web site. The user has pre-configured two SLS providers but her implementation does not understand any services beyond the 'dns' service. The first provider is scoped to her metropolitan area and the second handles names with a more global scope. The user attempts to ask for the 'dns:1:1' service for the name 'CNN' with their location set to 'us-tx-lubbock', their language (locale) set to 'en-us'. They leave the category blank. The query is sent to both the locally and globally scoped services. The locally scoped service returns no results and the global one returns the URI 'dns:www.cnn.com;type=a'.

<111>

The same scenario could work for leveraging legacy services such as ftp, instant messaging and even email (if applied carefully).

<112>

The exact transaction between the client and server looks like this. The client connects to the server (over some transport) and issues this request:

```
C:<?xml version="1.0"?>
C:  <!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
C:    "http://ietf.org/dtd/cnrp-1.0.dtd">
C:   <cnrp>
C:    <query>
C:       <commonname>cnn</commonname>
C:       <property name="geography" type="sls">us-tx-lubbock</property>
C:       <property name="locale"    type="posix">en-us</property>
C:       <property name="category"  type="nice"></property>
C:       <property name="service"   type="sls">dns:1:1</property>
C:    </query>
C:   </cnrp>

S:<?xml version="1.0"?>
S:<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
S:  "http://ietf.org/dtd/cnrp-1.0.dtd">
S:<cnrp>
S:  <results>
S:     <service id="i0">
S:         <serviceuri>http://example.com</serviceuri>
S:     </service>
S:     <resourcedescriptor>
S:       <commonname>CNN</commonname>
S:       <id>1333459455</id>
S:       <resourceuri>dns:www.cnn.com;type=A</resourceuri>
S:       <serviceref ref="i0" />
S:       <description>The Cable News Network (tm)</description>
S:       <property name="geography" type="sls">global</property>
S:       <property name="locale"    type="posix">en-us</property>
S:       <property name="category"  type="nice">380012</property>
S:       <property name="service"   type="sls">web</property>
S:     </resourcedescriptor>
S:  </results>
S:</cnrp>
```

4.8.2 The Web Service

<113>

The end goal is a more task specific service query. Take the previous scenario as a starting point but instead the user's client can understand the 'web' service. In this case the user is interested in the 'CNN Travel' name. They send the same query to both services and again the locally scoped one returns nothing but the globally scoped one returns the URI 'http://www.cnn.com/TRAVEL/'. Note how the name given by the user is all lower case but it matches the upper case. This can safely be done because the locale specifies sorting and matching algorithms specifically. The entity that

registered the name can specify whether or not the name is case sensitive or not.

<114>

Again, the actual XML sent looks like this:


```
C:<?xml version="1.0"?>
C:  <!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
C:    "http://ietf.org/dtd/cnrp-1.0.dtd">
C:   <cnrp>
C:    <query>
C:       <commonname>cnn travel</commonname>
C:       <property name="geography" type="sls">us-tx-lubbock</property>
C:       <property name="locale"    type="posix">en-us</property>
C:       <property name="category"  type="nice"></property>
C:       <property name="service"   type="sls">web</property>
C:    </query>
C:   </cnrp>

S:<?xml version="1.0"?>
S:<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
S:  "http://ietf.org/dtd/cnrp-1.0.dtd">
S:<cnrp>
S:  <results>
S:     <service id="i0">
S:         <serviceuri>http://example.com</serviceuri>
S:     </service>
S:     <resourcedescriptor>
S:       <commonname>CNN Travel</commonname>
S:       <id>1333459455</id>
S:       <resourceuri>http://www.cnn.com/TRAVEL/</resourceuri>
S:       <serviceref ref="i0" />
S:       <description>The Cable News Network: Travel
S:                 Section(tm)</description>
S:       <property name="geography" type="sls">global</property>
S:       <property name="locale"    type="posix">en-us</property>
S:       <property name="category"  type="nice">380012</property>
S:       <property name="service"   type="sls">web</property>
S:     </resourcedescriptor>
S:  </results>
S:</cnrp>
```


4.8.3 The Web Service With Ambiguous Results

<115>

Now, imagine the last scenario but with the name as "John's Computer Repair". In this case the user still asks for the 'web' service but the locally scoped provider returns one result and the globally

scoped one also returns a result. The one returned by the locally scoped provider is for a computer repair company just down the street from the user. The one from the globally scoped provider is for a computer repair company that advertises around the world. The user's client presents the user with a choice between the two and the user chooses.

<116>

In this case the exact same query is sent to both servers:

```
C:<?xml version="1.0"?>
C:  <!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
C:    "http://ietf.org/dtd/cnrp-1.0.dtd">
C:   <cnrp>
C:    <query>
C:       <commonname>john's computer repair</commonname>
C:       <property name="geography" type="sls">us-tx-lubbock</property>
C:       <property name="locale"    type="posix">en-us</property>
C:       <property name="category"  type="nice"></property>
C:       <property name="service"   type="sls">web</property>
C:    </query>
C:   </cnrp>
```

The locally scopped server returns this:

```
S:<?xml version="1.0"?>
S:<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
S:  "http://ietf.org/dtd/cnrp-1.0.dtd">
S:<cnrp>
S:  <results>
S:     <service id="i0">
S:         <serviceuri>http://lubbock-tx-example.com</serviceuri>
S:     </service>
S:     <resourcedescriptor>
S:        <commonname>John's Computer Repair</commonname>
S:        <id>1333459455</id>
S:        <resourceuri>http://www.lubbocknet/~john/</resourceuri>
S:        <serviceref ref="i0" />
S:        <description>Serving the Lubbock, TX computer user since 1948
S:                 </description>
S:       <property name="geography" type="sls">us-tx-lubbock</property>
S:       <property name="locale"    type="posix">en-us</property>
S:       <property name="category"  type="nice">370166</property>
S:       <property name="service"   type="sls">web</property>
S:     </resourcedescriptor>
S:  </results>
S:</cnrp>
```

while the globally scoped one returns this:

```
S:<?xml version="1.0"?>
S:<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
S:  "http://ietf.org/dtd/cnrp-1.0.dtd">
S:<cnrp>
S:  <results>
S:     <service id="i0">
S:         <serviceuri>http://example.com</serviceuri>
S:     </service>
S:     <resourcedescriptor>
S:        <commonname>John's Computer Repair</commonname>
S:        <id>1333459455</id>
S:        <resourceuri>http://www.computer-repair.biz/</resourceuri>
S:        <serviceref ref="i0" />
S:        <description>Worldwide hardware repair and software consulting
S:                 via mail order</description>
S:       <property name="geography" type="sls">global</property>
S:       <property name="locale"    type="posix">en-us</property>
S:       <property name="category"  type="nice">370166</property>
S:       <property name="service"   type="sls">web</property>
S:     </resourcedescriptor>
S:  </results>
S:</cnrp>
```

4.8.4 The Web Service With Ambiguous Query and Results

<117>

The previous example can also happen when the user specifies an ambiguous, blank or multivalued facet. For example, since the user never specified a category, "John's Computer Repair" could have matched several different NSRs that had the same name but different facet values. A more likely example would be 'Genesis' (the band and the hydraulics company). If the user were to specify a query for Genesis and left the category blank then the user could consievably get a large number of answers back:

```
C:<?xml version="1.0"?>
C:  <!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
C:    "http://ietf.org/dtd/cnrp-1.0.dtd">
C:   <cnrp>
C:    <query>
C:       <commonname>Gensis</commonname>
C:       <property name="geography" type="sls">us-tx-lubbock</property>
C:       <property name="locale"    type="posix">en-us</property>
C:       <property name="category"  type="nice"></property>
C:       <property name="service"   type="sls">web</property>
C:    </query>
C:   </cnrp>
```

which would return in a series of results:

```
S:<?xml version="1.0"?>
S:<!DOCTYPE cnrp PUBLIC "-//IETF//DTD CNRP 1.0//EN"
S:  "http://ietf.org/dtd/cnrp-1.0.dtd">
S:<cnrp>
S:  <results>
S:     <service id="i0">
S:         <serviceuri>http://example.com</serviceuri>
S:     </service>
S:     <resourcedescriptor>
S:       <commonname>Genesis</commonname>
S:       <id>1333459455</id>
S:       <resourceuri>http://www.sony.com/genesis</resourceuri>
S:       <serviceref ref="i0" />
S:       <description>The band</description>
S:       <property name="geography" type="sls">global</property>
S:       <property name="locale"    type="posix">en-us</property>
S:       <property name="category"  type="nice">410023</property>
S:       <property name="service"   type="sls">web</property>
S:     </resourcedescriptor>
S:     <resourcedescriptor>
S:        <commonname>Genesis</commonname>
S:        <id>2345432</id>
S:        <resourceuri>http://www.genesis-hydraulics.com/genesis
S:          </resourceuri>
S:        <serviceref ref="i0" />
S:        <description>Providing world wide hydraulics engineering
S:              services sinde 1973</description>
S:       <property name="geography" type="sls">global</property>
S:       <property name="locale"    type="posix">en-us</property>
S:       <property name="category"  type="nice">370166</property>
S:       <property name="service"   type="sls">web</property>
```

```
S:     </resourcedescriptor>
S:     <resourcedescriptor>
S:              .... other results from other categories
S:     </resourcedescriptor>
S:  </results>
S:</cnrp>
```

References

[I-D.ietf-cnrp] Mealling, M., Popp, N. and M. Moseley, "Common Name Resolution Protocol (CNRP)", draft-ietf-cnrp-12 (work in progress), February 2002.

[I-D.ietf-rescap-blob] Moore, K., "The Binary Low-Overhead Block Presentation Protocol", draft-ietf-rescap-blob-01 (work in progress), March 2002.

[I-D.ietf-rescap-rc] Moore, K., "The Resource Catalog", draft-ietf-rescap-rc-01 (work in progress), March 2002.

[I-D.josefsson-dns-url] Josefsson, S., "Domain Name System URI Scheme and MIME Media Types", draft-josefsson-dns-url-05 (work in progress), May 2002.

[I-D.klensin-dns-role] Klensin, J., "Role of the Domain Name System", draft-klensin-dns-role-03 (work in progress), June 2002.

[I-D.klensin-dns-search] Klensin, J., "A Search-based access model for the DNS", draft-klensin-dns-search-03 (work in progress), March 2002.

[I-D.klensin-i18n-newclass] Klensin, J., "Internationalizing the DNS -- A New Class", draft-klensin-i18n-newclass-02 (work in progress), June 2002.

[NICE] World Intellectual Property Organization, "Nice Agreement concerning the International Classification of Goods and Services for the Purposes of the Registration of Marks", June 1957.

[RFC0822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC3066] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[RFC3081] Rose, M., "Mapping the BEEP Core onto TCP", RFC 3081, March 2001.

Authors' Addresses

Michael Mealling
VeriSign, Inc.
21345 Ridgetop Circle
Sterling, VA 20166
US

EMail: michael@verisignlabs.com
URI: http://www.verisign.com

Leslie Daigle
VeriSign, Inc.
21345 Ridgetop Circle
Sterling, VA 20166
US

EMail: leslie@verisignlabs.com
URI: http://www.verisign.com

Appendix A. Version History

A.1 Version 00 to 01

<118>
<119>
- o Added the Usage Scenarios section

<120>
- o Added the appendis on service provider discovery

A.2 Version 01 to 02

<121>

<122>

o Re-organized the requirements analysis

<123>

o Added discussions on data representation and transport

<124>

o Removed the Og story

Appendix B. SLS Service Provider Discovery

<125>

One of the hardest parts of the SLS system is how to discover all of the available SLS providers in a way that is not mired in political/ social issues of ownership over spaces. The proposed solution is a peer-to-peer style service announcement mechanism where anyone can announce their intention to provide an SLS service. The key to such a mechanism is being able to trust that the service has not censored the list of advertisements. The following is a description of such a discovery mechanism:

<126>

Components of the system:

<127>

Assertions opaque bits of data inserted into the system by Asserters. The system makes no statements about Assertsions or checks their validity.

<128>

Clients Those that are looking for the list of assertions. They do not normally do verification

<129>

Asserters Systems that insert Assertions into the system and periodically verify their existence

<130>

Advertisers The systems that handle the requests for and the maintenance of the current set of all Advertisers as well as all of the Assertions

<131>

The key here is how to make sure no one is maliciously changing the data. The problem is that the servers advertising the data can't be trusted with the responsibility of verifying it. The only one that can really do it is the one who cares and who knows what the original assertion was. Thus if Alice wants to advertise assertion X then Alice has to request that assertion from a random number of servers in the system at random times to verify that the assertion is 'correct' and actually being advertised.

<132>

The important case is reliably getting the "list of all servers" from

some subset of those servers in a way that makes it so that some malicious subset can't cut the rest off. A server could filter this list and create a constrained 'view' of the state for queriers. The key is to apply the same method from above for server to server verification. Servers will request the list of servers from each other on a random basis. If they don't find themselves in a particular server's list then they complain to other servers. The goal here is to utilize the sense of self preservation: Mutual Assured Destruction at a distributed systems level.

<133>

The only cryptographic part is proving that someone has done a bad thing. This is handled by signature matches and _VERY_ specific cases where this is done. The only two that are glaringly obvious are:

<134>

1. a client claims that server n is not advertising the client's assertion X.

<135>

2. server n claims that some other server is not advertising it in the list of servers.

<136>

How do you prove these cases? Pretty easily actually. All requests to add an assertion or to join the system of servers generate a response in the form of the "assertion" message being timestamped and signed by the server that handled the request. (a question here is do assertions have expirations or are they 'deleted' on request). In case #1 the client merely has to approach another server (or servers) and say "see, I have the signed "ok" message". The servers then go check that server to see if it is or is not advertising the assertion. If it is they tell the client so. If not then they notify the server and delist it and also notify all of the other servers. All servers verify these notifications themselves (because they don't trust each other).

<137>

The second case is a special case of the first. Here each server has the responsibility to randomly check the other servers to make sure they are all giving out the entire list of servers. Again, when a server wishes to join the party it receives a signed and timestamped response to its request. If at any time it notices that some other server is not including it in its list it sends this signed acknowledgment to the other servers requesting that they verify this as well. They do so and if its true then they delist the offending server.

<138>

There are some boundary cases here that can cause the system's sense of trust (or distrust) to fall apart. A random time out needs to be done on delisting so that servers who have been behind a network

outage can re-synchronize. It could be engineered so that if a network outage does make a server inaccessible the other servers could be considered in a 'non-conclusive' state. A response to the "you're not doing what you're supposed to do" claim would be "I am now!". That should be fine. Should anything happen if that's consistently a problem? Can a server be delisted for having bad connectivity?

<139>

This discussion is very preliminary and needs review by distributed systems and security experts.

Full Copyright Statement

Acknowledgement

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