Network Working Group
Request for Comments: 4472
Category: Informational

A. Durand
Comcast
J. Ihren
Autonomica
P. Savola
CSC/FUNET

April 2006

Operational Considerations and Issues with IPv6 DNS

Status of This Memo

Copyright Notice

Abstract

This memo presents operational considerations and issues with IPv6 Domain Name System (DNS), including a summary of special IPv6 addresses, documentation of known DNS implementation misbehavior, recommendations and considerations on how to perform DNS naming for service provisioning and for DNS resolver IPv6 support, considerations for DNS updates for both the forward and reverse trees, and miscellaneous issues. This memo is aimed to include a summary of information about IPv6 DNS considerations for those who have experience with IPv4 DNS.

Table of Contents

1. Introduction

This memo presents operational considerations and issues with IPv6 DNS; it is meant to be an extensive summary and a list of pointers for more information about IPv6 DNS considerations for those with experience with IPv4 DNS.

The purpose of this document is to give information about various issues and considerations related to DNS operations with IPv6; it is not meant to be a normative specification or standard for IPv6 DNS.

The first section gives a brief overview of how IPv6 addresses and names are represented in the DNS, how transport protocols and resource records (don't) relate, and what IPv4/IPv6 name space fragmentation means and how to avoid it; all of these are described at more length in other documents.

The second section summarizes the special IPv6 address types and how they relate to DNS. The third section describes observed DNS implementation misbehaviors that have a varying effect on the use of IPv6 records with DNS. The fourth section lists recommendations and considerations for provisioning services with DNS. The fifth section in turn looks at recommendations and considerations about providing IPv6 support in the resolvers. The sixth and seventh sections describe considerations with forward and reverse DNS updates, respectively. The eighth section introduces several miscellaneous IPv6 issues relating to DNS for which no better place has been found in this memo. Appendix A looks briefly at the requirements for unique local addressing. Appendix B discusses additional data.

1.1. Representing IPv6 Addresses in DNS Records

In the forward zones, IPv6 addresses are represented using AAAA records. In the reverse zones, IPv6 address are represented using PTR records in the nibble format under the ip6.arpa. tree. See [RFC3596] for more about IPv6 DNS usage, and [RFC3363] or [RFC3152] for background information.

In particular, one should note that the use of A6 records in the forward tree or Bitlabels in the reverse tree is not recommended [RFC3363]. Using DNAME records is not recommended in the reverse tree in conjunction with A6 records; the document did not mean to take a stance on any other use of DNAME records [RFC3364].

1.2. Independence of DNS Transport and DNS Records

DNS has been designed to present a single, globally unique name space [RFC2826]. This property should be maintained, as described here and in Section 1.3.

The IP version used to transport the DNS queries and responses is independent of the records being queried: AAAA records can be queried over IPv4, and A records over IPv6. The DNS servers must not make any assumptions about what data to return for Answer and Authority sections based on the underlying transport used in a query.

However, there is some debate whether the addresses in Additional section could be selected or filtered using hints obtained from which transport was being used; this has some obvious problems because in many cases the transport protocol does not correlate with the requests, and because a "bad" answer is in a way worse than no answer at all (consider the case where the client is led to believe that a name received in the additional record does not have any AAAA records at all).

As stated in [RFC3596]:

> The IP protocol version used for querying resource records is independent of the protocol version of the resource records; e.g., IPv4 transport can be used to query IPv6 records and vice versa.

1.3. Avoiding IPv4/IPv6 Name Space Fragmentation

To avoid the DNS name space from fragmenting into parts where some parts of DNS are only visible using IPv4 (or IPv6) transport, the recommendation is to always keep at least one authoritative server IPv4-enabled, and to ensure that recursive DNS servers support IPv4. See DNS IPv6 transport guidelines [RFC3901] for more information.

1.4. Query Type '*' and A/AAAA Records

QTYPE=* is typically only used for debugging or management purposes; it is worth keeping in mind that QTYPE=* ("ANY" queries) only return any available RRsets, not *all* the RRsets, because the caches do not necessarily have all the RRsets and have no way of guaranteeing that they have all the RRsets. Therefore, to get both A and AAAA records reliably, two separate queries must be made.

2. DNS Considerations about Special IPv6 Addresses

There are a couple of IPv6 address types that are somewhat special; these are considered here.

2.1. Limited-Scope Addresses

The IPv6 addressing architecture [RFC4291] includes two kinds of local-use addresses: link-local (fe80::/10) and site-local (fec0::/10). The site-local addresses have been deprecated [RFC3879] but are discussed with unique local addresses in Appendix A.

Link-local addresses should never be published in DNS (whether in forward or reverse tree), because they have only local (to the connected link) significance [WIP-DC2005].

2.2. Temporary Addresses

Temporary addresses defined in RFC 3041 [RFC3041] (sometimes called "privacy addresses") use a random number as the interface identifier. Having DNS AAAA records that are updated to always contain the current value of a node's temporary address would defeat the purpose of the mechanism and is not recommended. However, it would still be possible to return a non-identifiable name (e.g., the IPv6 address in hexadecimal format), as described in [RFC3041].

2.3. 6to4 Addresses

6to4 [RFC3056] specifies an automatic tunneling mechanism that maps a public IPv4 address V4ADDR to an IPv6 prefix 2002:V4ADDR::/48.

If the reverse DNS population would be desirable (see Section 7.1 for applicability), there are a number of possible ways to do so.

[WIP-H2005] aims to design an autonomous reverse-delegation system that anyone being capable of communicating using a specific 6to4 address would be able to set up a reverse delegation to the corresponding 6to4 prefix. This could be deployed by, e.g., Regional Internet Registries (RIRs). This is a practical solution, but may have some scalability concerns.

2.4. Other Transition Mechanisms

6to4 is mentioned as a case of an IPv6 transition mechanism requiring special considerations. In general, mechanisms that include a special prefix may need a custom solution; otherwise, for example, when IPv4 address is embedded as the suffix or not embedded at all, special solutions are likely not needed.

Note that it does not seem feasible to provide reverse DNS with another automatic tunneling mechanism, Teredo [RFC4380]; this is because the IPv6 address is based on the IPv4 address and UDP port of the current Network Address Translation (NAT) mapping, which is likely to be relatively short-lived.

3. Observed DNS Implementation Misbehavior

Several classes of misbehavior in DNS servers, load-balancers, and resolvers have been observed. Most of these are rather generic, not only applicable to IPv6 -- but in some cases, the consequences of this misbehavior are extremely severe in IPv6 environments and deserve to be mentioned.

3.1. Misbehavior of DNS Servers and Load-balancers

There are several classes of misbehavior in certain DNS servers and load-balancers that have been noticed and documented [RFC4074]: some implementations silently drop queries for unimplemented DNS records types, or provide wrong answers to such queries (instead of a proper negative reply). While typically these issues are not limited to AAAA records, the problems are aggravated by the fact that AAAA records are being queried instead of (mainly) A records.

The problems are serious because when looking up a DNS name, typical getaddrinfo() implementations, with AF_UNSPEC hint given, first try to query the AAAA records of the name, and after receiving a response, query the A records. This is done in a serial fashion -- if the first query is never responded to (instead of properly returning a negative answer), significant time-outs will occur.

In consequence, this is an enormous problem for IPv6 deployments, and in some cases, IPv6 support in the software has even been disabled due to these problems.

The solution is to fix or retire those misbehaving implementations, but that is likely not going to be effective. There are some possible ways to mitigate the problem, e.g., by performing the lookups somewhat in parallel and reducing the time-out as long as at least one answer has been received, but such methods remain to be investigated; slightly more on this is included in Section 5.

3.2. Misbehavior of DNS Resolvers

Several classes of misbehavior have also been noticed in DNS resolvers [WIP-LB2005]. However, these do not seem to directly impair IPv6 use, and are only referred to for completeness.

4. Recommendations for Service Provisioning Using DNS

When names are added in the DNS to facilitate a service, there are several general guidelines to consider to be able to do it as smoothly as possible.

4.1. Use of Service Names instead of Node Names

It makes sense to keep information about separate services logically separate in the DNS by using a different DNS hostname for each service. There are several reasons for doing this, for example:

- It allows more flexibility and ease for migration of (only a part of) services from one node to another,
- It allows configuring different properties (e.g., Time to Live (TTL)) for each service, and
- It allows deciding separately for each service whether or not to publish the IPv6 addresses (in cases where some services are more IPv6-ready than others).

Using SRV records [RFC2782] would avoid these problems. Unfortunately, those are not sufficiently widely used to be applicable in most cases. Hence an operation technique is to use service names instead of node names (or "hostnames"). This operational technique is not specific to IPv6, but required to understand the considerations described in Section 4.2 and Section 4.3.

For example, assume a node named "pobox.example.com" provides both SMTP and IMAP service. Instead of configuring the MX records to point at "pobox.example.com", and configuring the mail clients to look up the mail via IMAP from "pobox.example.com", one could use, e.g., "smtp.example.com" for SMTP (for both message submission and mail relaying between SMTP servers) and "imap.example.com" for IMAP. Note that in the specific case of SMTP relaying, the server itself must typically also be configured to know all its names to ensure that loops do not occur. DNS can provide a layer of indirection between service names and where the service actually is, and using which addresses. (Obviously, when wanting to reach a specific node, one should use the hostname rather than a service name.)

4.2. Separate vs. the Same Service Names for IPv4 and IPv6

The service naming can be achieved in basically two ways: when a service is named "service.example.com" for IPv4, the IPv6-enabled service could either be added to "service.example.com" or added separately under a different name, e.g., in a sub-domain like "service.ipv6.example.com".

These two methods have different characteristics. Using a different name allows for easier service piloting, minimizing the disturbance to the "regular" users of IPv4 service; however, the service would not be used transparently, without the user/application explicitly finding it and asking for it -- which would be a disadvantage in most cases. When the different name is under a sub-domain, if the services are deployed within a restricted network (e.g., inside an enterprise), it's possible to prefer them transparently, at least to a degree, by modifying the DNS search path; however, this is a suboptimal solution. Using the same service name is the "long-term" solution, but may degrade performance for those clients whose IPv6 performance is lower than IPv4, or does not work as well (see Section 4.3 for more).

In most cases, it makes sense to pilot or test a service using separate service names, and move to the use of the same name when confident enough that the service level will not degrade for the users unaware of IPv6.

4.3. Adding the Records Only When Fully IPv6-enabled

The recommendation is that AAAA records for a service should not be added to the DNS until all of following are true:

1. The address is assigned to the interface on the node.

2. The address is configured on the interface.

3. The interface is on a link that is connected to the IPv6 infrastructure.

In addition, if the AAAA record is added for the node, instead of service as recommended, all the services of the node should be IPv6-enabled prior to adding the resource record.

For example, if an IPv6 node is isolated from an IPv6 perspective (e.g., it is not connected to IPv6 Internet) constraint #3 would mean that it should not have an address in the DNS.

Consider the case of two dual-stack nodes, which both are IPv6-enabled, but the server does not have (global) IPv6 connectivity. As the client looks up the server's name, only A records are returned (if the recommendations above are followed), and no IPv6 communication, which would have been unsuccessful, is even attempted.

The issues are not always so black-and-white. Usually, it's important that the service offered using both protocols is of roughly equal quality, using the appropriate metrics for the service (e.g., latency, throughput, low packet loss, general reliability, etc.). This is typically very important especially for interactive or real-time services. In many cases, the quality of IPv6 connectivity may not yet be equal to that of IPv4, at least globally; this has to be taken into consideration when enabling services.

4.4. The Use of TTL for IPv4 and IPv6 RRs

The behavior of DNS caching when different TTL values are used for different RRsets of the same name calls for explicit discussion. For example, let's consider two unrelated zone fragments:

```
example.com.        300    IN    MX     foo.example.com.
foo.example.com.    300    IN    A      192.0.2.1
foo.example.com.    100    IN    AAAA   2001:db8::1
```

...

```
child.example.com.    300  IN    NS     ns.child.example.com.
ns.child.example.com. 300  IN    A      192.0.2.1
ns.child.example.com. 100  IN    AAAA   2001:db8::1
```

In the former case, we have "courtesy" additional data; in the latter, we have "critical" additional data. See more extensive background discussion of additional data handling in Appendix B.

4.4.1. TTL with Courtesy Additional Data

When a caching resolver asks for the MX record of example.com, it gets back "foo.example.com". It may also get back either one or both of the A and AAAA records in the additional section. The resolver must explicitly query for both A and AAAA records [RFC2821].

After 100 seconds, the AAAA record is removed from the cache(s) because its TTL expired. It could be argued to be useful for the caching resolvers to discard the A record when the shorter TTL (in this case, for the AAAA record) expires; this would avoid the situation where there would be a window of 200 seconds when incomplete information is returned from the cache. Further argument

for discarding is that in the normal operation, the TTL values are so high that very likely the incurred additional queries would not be noticeable, compared to the obtained performance optimization. The behavior in this scenario is unspecified.

4.4.2. TTL with Critical Additional Data

The difference to courtesy additional data is that the A/AAAA records served by the parent zone cannot be queried explicitly. Therefore, after 100 seconds the AAAA record is removed from the cache(s), but the A record remains. Queries for the remaining 200 seconds (provided that there are no further queries from the parent that could refresh the caches) only return the A record, leading to a potential operational situation with unreachable servers.

Similar cache flushing strategies apply in this scenario; the behavior is likewise unspecified.

4.5. IPv6 Transport Guidelines for DNS Servers

As described in Section 1.3 and [RFC3901], there should continue to be at least one authoritative IPv4 DNS server for every zone, even if the zone has only IPv6 records. (Note that obviously, having more servers with robust connectivity would be preferable, but this is the minimum recommendation; also see [RFC2182].)

5. Recommendations for DNS Resolver IPv6 Support

When IPv6 is enabled on a node, there are several things to consider to ensure that the process is as smooth as possible.

5.1. DNS Lookups May Query IPv6 Records Prematurely

The system library that implements the getaddrinfo() function for looking up names is a critical piece when considering the robustness of enabling IPv6; it may come in basically three flavors:

1. The system library does not know whether IPv6 has been enabled in the kernel of the operating system: it may start looking up AAAA records with getaddrinfo() and AF_UNSPEC hint when the system is upgraded to a system library version that supports IPv6.

2. The system library might start to perform IPv6 queries with getaddrinfo() only when IPv6 has been enabled in the kernel. However, this does not guarantee that there exists any useful IPv6 connectivity (e.g., the node could be isolated from the other IPv6 networks, only having link-local addresses).

3. The system library might implement a toggle that would apply some heuristics to the "IPv6-readiness" of the node before starting to perform queries; for example, it could check whether only link-local IPv6 address(es) exists, or if at least one global IPv6 address exists.

First, let us consider generic implications of unnecessary queries for AAAA records: when looking up all the records in the DNS, AAAA records are typically tried first, and then A records. These are done in serial, and the A query is not performed until a response is received to the AAAA query. Considering the misbehavior of DNS servers and load-balancers, as described in Section 3.1, the lookup delay for AAAA may incur additional unnecessary latency, and introduce a component of unreliability.

One option here could be to do the queries partially in parallel; for example, if the final response to the AAAA query is not received in 0.5 seconds, start performing the A query while waiting for the result. (Immediate parallelism might not be optimal, at least without information-sharing between the lookup threads, as that would probably lead to duplicate non-cached delegation chain lookups.)

An additional concern is the address selection, which may, in some circumstances, prefer AAAA records over A records even when the node does not have any IPv6 connectivity [WIP-RDP2004]. In some cases, the implementation may attempt to connect or send a datagram on a physical link [WIP-R2006], incurring very long protocol time-outs, instead of quickly falling back to IPv4.

Now, we can consider the issues specific to each of the three possibilities:

In the first case, the node performs a number of completely useless DNS lookups as it will not be able to use the returned AAAA records anyway. (The only exception is where the application desires to know what's in the DNS, but not use the result for communication.) One should be able to disable these unnecessary queries, for both latency and reliability reasons. However, as IPv6 has not been enabled, the connections to IPv6 addresses fail immediately, and if the application is programmed properly, the application can fall gracefully back to IPv4 [RFC4038].

The second case is similar to the first, except it happens to a smaller set of nodes when IPv6 has been enabled but connectivity has not been provided yet. Similar considerations apply, with the exception that IPv6 records, when returned, will be actually tried first, which may typically lead to long time-outs.

The third case is a bit more complex: optimizing away the DNS lookups with only link-locals is probably safe (but may be desirable with different lookup services that getaddrinfo() may support), as the link-locals are typically automatically generated when IPv6 is enabled, and do not indicate any form of IPv6 connectivity. That is, performing DNS lookups only when a non-link-local address has been configured on any interface could be beneficial -- this would be an indication that the address has been configured either from a router advertisement, Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [RFC3315], or manually. Each would indicate at least some form of IPv6 connectivity, even though there would not be guarantees of it.

These issues should be analyzed at more depth, and the fixes found consensus on, perhaps in a separate document.

5.2. Obtaining a List of DNS Recursive Resolvers

In scenarios where DHCPv6 is available, a host can discover a list of DNS recursive resolvers through the DHCPv6 "DNS Recursive Name Server" option [RFC3646]. This option can be passed to a host through a subset of DHCPv6 [RFC3736].

The IETF is considering the development of alternative mechanisms for obtaining the list of DNS recursive name servers when DHCPv6 is unavailable or inappropriate. No decision about taking on this development work has been reached as of this writing [RFC4339].

In scenarios where DHCPv6 is unavailable or inappropriate, mechanisms under consideration for development include the use of [WIP-O2004] and the use of Router Advertisements to convey the information [WIP-J2006].

Note that even though IPv6 DNS resolver discovery is a recommended procedure, it is not required for dual-stack nodes in dual-stack networks as IPv6 DNS records can be queried over IPv4 as well as IPv6. Obviously, nodes that are meant to function without manual configuration in IPv6-only networks must implement the DNS resolver discovery function.

5.3. IPv6 Transport Guidelines for Resolvers

As described in Section 1.3 and [RFC3901], the recursive resolvers should be IPv4-only or dual-stack to be able to reach any IPv4-only DNS server. Note that this requirement is also fulfilled by an IPv6-only stub resolver pointing to a dual-stack recursive DNS resolver.

6. Considerations about Forward DNS Updating

While the topic of how to enable updating the forward DNS, i.e., the mapping from names to the correct new addresses, is not specific to IPv6, it should be considered especially due to the advent of Stateless Address Autoconfiguration [RFC2462].

Typically, forward DNS updates are more manageable than doing them in the reverse DNS, because the updater can often be assumed to "own" a certain DNS name -- and we can create a form of security relationship with the DNS name and the node that is allowed to update it to point to a new address.

A more complex form of DNS updates -- adding a whole new name into a DNS zone, instead of updating an existing name -- is considered out of scope for this memo as it could require zone-wide authentication. Adding a new name in the forward zone is a problem that is still being explored with IPv4, and IPv6 does not seem to add much new in that area.

6.1. Manual or Custom DNS Updates

The DNS mappings can also be maintained by hand, in a semi-automatic fashion or by running non-standardized protocols. These are not considered at more length in this memo.

6.2. Dynamic DNS

Dynamic DNS updates (DDNS) [RFC2136] [RFC3007] is a standardized mechanism for dynamically updating the DNS. It works equally well with Stateless Address Autoconfiguration (SLAAC), DHCPv6, or manual address configuration. It is important to consider how each of these behave if IP address-based authentication, instead of stronger mechanisms [RFC3007], was used in the updates.

1. Manual addresses are static and can be configured.

2. DHCPv6 addresses could be reasonably static or dynamic, depending on the deployment, and could or could not be configured on the DNS server for the long term.

3. SLAAC addresses are typically stable for a long time, but could require work to be configured and maintained.

As relying on IP addresses for Dynamic DNS is rather insecure at best, stronger authentication should always be used; however, this requires that the authorization keying will be explicitly configured using unspecified operational methods.

Note that with DHCP it is also possible that the DHCP server updates the DNS, not the host. The host might only indicate in the DHCP exchange which hostname it would prefer, and the DHCP server would make the appropriate updates. Nonetheless, while this makes setting up a secure channel between the updater and the DNS server easier, it does not help much with "content" security, i.e., whether the hostname was acceptable -- if the DNS server does not include policies, they must be included in the DHCP server (e.g., a regular host should not be able to state that its name is "www.example.com"). DHCP-initiated DDNS updates have been extensively described in [WIP-SV2005], [WIP-S2005a], and [WIP-S2005b].

The nodes must somehow be configured with the information about the servers where they will attempt to update their addresses, sufficient security material for authenticating themselves to the server, and the hostname they will be updating. Unless otherwise configured, the first could be obtained by looking up the authoritative name servers for the hostname; the second must be configured explicitly unless one chooses to trust the IP address-based authentication (not a good idea); and lastly, the nodename is typically pre-configured somehow on the node, e.g., at install time.

Care should be observed when updating the addresses not to use longer TTLs for addresses than are preferred lifetimes for the addresses, so that if the node is renumbered in a managed fashion, the amount of stale DNS information is kept to the minimum. That is, if the preferred lifetime of an address expires, the TTL of the record needs to be modified unless it was already done before the expiration. For better flexibility, the DNS TTL should be much shorter (e.g., a half or a third) than the lifetime of an address; that way, the node can start lowering the DNS TTL if it seems like the address has not been renewed/refreshed in a while. Some discussion on how an administrator could manage the DNS TTL is included in [RFC4192]; this could be applied to (smart) hosts as well.

7. Considerations about Reverse DNS Updating

Updating the reverse DNS zone may be difficult because of the split authority over an address. However, first we have to consider the applicability of reverse DNS in the first place.

7.1. Applicability of Reverse DNS

Today, some applications use reverse DNS either to look up some hints about the topological information associated with an address (e.g., resolving web server access logs) or (as a weak form of a security check) to get a feel whether the user's network administrator has

"authorized" the use of the address (on the premise that adding a reverse record for an address would signal some form of authorization).

One additional, maybe slightly more useful usage is ensuring that the reverse and forward DNS contents match (by looking up the pointer to the name by the IP address from the reverse tree, and ensuring that a record under the name in the forward tree points to the IP address) and correspond to a configured name or domain. As a security check, it is typically accompanied by other mechanisms, such as a user/password login; the main purpose of the reverse+forward DNS check is to weed out the majority of unauthorized users, and if someone managed to bypass the checks, he would still need to authenticate "properly".

It may also be desirable to store IPsec keying material corresponding to an IP address in the reverse DNS, as justified and described in [RFC4025].

It is not clear whether it makes sense to require or recommend that reverse DNS records be updated. In many cases, it would just make more sense to use proper mechanisms for security (or topological information lookup) in the first place. At minimum, the applications that use it as a generic authorization (in the sense that a record exists at all) should be modified as soon as possible to avoid such lookups completely.

The applicability is discussed at more length in [WIP-S2005c].

7.2. Manual or Custom DNS Updates

Reverse DNS can of course be updated using manual or custom methods. These are not further described here, except for one special case.

One way to deploy reverse DNS would be to use wildcard records, for example, by configuring one name for a subnet (/64) or a site (/48). As a concrete example, a site (or the site's ISP) could configure the reverses of the prefix 2001:db8:f00::/48 to point to one name using a wildcard record like "*.0.0.f.0.8.b.d.0.1.0.0.2.ip6.arpa. IN PTR site.example.com.". Naturally, such a name could not be verified from the forward DNS, but would at least provide some form of "topological information" or "weak authorization" if that is really considered to be useful. Note that this is not actually updating the DNS as such, as the whole point is to avoid DNS updates completely by manually configuring a generic name.

7.3. DDNS with Stateless Address Autoconfiguration

Dynamic reverse DNS with SLAAC is simpler than forward DNS updates in some regard, while being more difficult in another, as described below.

The address space administrator decides whether or not the hosts are trusted to update their reverse DNS records. If they are trusted and deployed at the same site (e.g., not across the Internet), a simple address-based authorization is typically sufficient (i.e., check that the DNS update is done from the same IP address as the record being updated); stronger security can also be used [RFC3007]. If they aren't allowed to update the reverses, no update can occur. However, such address-based update authorization operationally requires that ingress filtering [RFC3704] has been set up at the border of the site where the updates occur, and as close to the updater as possible.

Address-based authorization is simpler with reverse DNS (as there is a connection between the record and the address) than with forward DNS. However, when a stronger form of security is used, forward DNS updates are simpler to manage because the host can be assumed to have an association with the domain. Note that the user may roam to different networks and does not necessarily have any association with the owner of that address space. So, assuming a stronger form of authorization for reverse DNS updates than an address association is generally infeasible.

Moreover, the reverse zones must be cleaned up by an unspecified janitorial process: the node does not typically know a priori that it will be disconnected, and it cannot send a DNS update using the correct source address to remove a record.

A problem with defining the clean-up process is that it is difficult to ensure that a specific IP address and the corresponding record are no longer being used. Considering the huge address space, and the unlikelihood of collision within 64 bits of the interface identifiers, a process that would remove the record after no traffic has been seen from a node in a long period of time (e.g., a month or year) might be one possible approach.

To insert or update the record, the node must discover the DNS server to send the update to somehow, similar to as discussed in Section 6.2. One way to automate this is looking up the DNS server authoritative (e.g., through SOA record) for the IP address being updated, but the security material (unless the IP address-based authorization is trusted) must also be established by some other means.

One should note that Cryptographically Generated Addresses (CGAs) [RFC3972] may require a slightly different kind of treatment. CGAs are addresses where the interface identifier is calculated from a public key, a modifier (used as a nonce), the subnet prefix, and other data. Depending on the usage profile, CGAs might or might not be changed periodically due to, e.g., privacy reasons. As the CGA address is not predictable, a reverse record can only reasonably be inserted in the DNS by the node that generates the address.

7.4. DDNS with DHCP

With DHCPv4, the reverse DNS name is typically already inserted to the DNS that reflects the name (e.g., "dhcp-67.example.com"). One can assume similar practice may become commonplace with DHCPv6 as well; all such mappings would be pre-configured and would require no updating.

If a more explicit control is required, similar considerations as with SLAAC apply, except for the fact that typically one must update a reverse DNS record instead of inserting one (if an address assignment policy that reassigns disused addresses is adopted) and updating a record seems like a slightly more difficult thing to secure. However, it is yet uncertain how DHCPv6 is going to be used for address assignment.

Note that when using DHCP, either the host or the DHCP server could perform the DNS updates; see the implications in Section 6.2.

If disused addresses were to be reassigned, host-based DDNS reverse updates would need policy considerations for DNS record modification, as noted above. On the other hand, if disused address were not to be assigned, host-based DNS reverse updates would have similar considerations as SLAAC in Section 7.3. Server-based updates have similar properties except that the janitorial process could be integrated with DHCP address assignment.

7.5. DDNS with Dynamic Prefix Delegation

In cases where a prefix, instead of an address, is being used and updated, one should consider what is the location of the server where DDNS updates are made. That is, where the DNS server is located:

1. At the same organization as the prefix delegator.

2. At the site where the prefixes are delegated to. In this case, the authority of the DNS reverse zone corresponding to the delegated prefix is also delegated to the site.

3. Elsewhere; this implies a relationship between the site and where the DNS server is located, and such a relationship should be rather straightforward to secure as well. Like in the previous case, the authority of the DNS reverse zone is also delegated.

In the first case, managing the reverse DNS (delegation) is simpler as the DNS server and the prefix delegator are in the same administrative domain (as there is no need to delegate anything at all); alternatively, the prefix delegator might forgo DDNS reverse capability altogether, and use, e.g., wildcard records (as described in Section 7.2). In the other cases, it can be slightly more difficult, particularly as the site will have to configure the DNS server to be authoritative for the delegated reverse zone, implying automatic configuration of the DNS server -- as the prefix may be dynamic.

Managing the DDNS reverse updates is typically simple in the second case, as the updated server is located at the local site, and arguably IP address-based authentication could be sufficient (or if not, setting up security relationships would be simpler). As there is an explicit (security) relationship between the parties in the third case, setting up the security relationships to allow reverse DDNS updates should be rather straightforward as well (but IP address-based authentication might not be acceptable). In the first case, however, setting up and managing such relationships might be a lot more difficult.

8. Miscellaneous DNS Considerations

This section describes miscellaneous considerations about DNS that seem related to IPv6, for which no better place has been found in this document.

8.1. NAT-PT with DNS-ALG

The DNS-ALG component of NAT-PT [RFC2766] mangles A records to look like AAAA records to the IPv6-only nodes. Numerous problems have been identified with [WIP-AD2005]. This is a strong reason not to use NAT-PT in the first place.

8.2. Renumbering Procedures and Applications' Use of DNS

One of the most difficult problems of systematic IP address renumbering procedures [RFC4192] is that an application that looks up a DNS name disregards information such as TTL, and uses the result obtained from DNS as long as it happens to be stored in the memory of the application. For applications that run for a long time, this

could be days, weeks, or even months. Some applications may be clever enough to organize the data structures and functions in such a manner that lookups get refreshed now and then.

While the issue appears to have a clear solution, "fix the applications", practically, this is not reasonable immediate advice. The TTL information is not typically available in the APIs and libraries (so, the advice becomes "fix the applications, APIs, and libraries"), and a lot more analysis is needed on how to practically go about to achieve the ultimate goal of avoiding using the names longer than expected.

9. Acknowledgements

Some recommendations (Section 4.3, Section 5.1) about IPv6 service provisioning were moved here from [RFC4213] by Erik Nordmark and Bob Gilligan. Havard Eidnes and Michael Patton provided useful feedback and improvements. Scott Rose, Rob Austein, Masataka Ohta, and Mark Andrews helped in clarifying the issues regarding additional data and the use of TTL. Jefsey Morfin, Ralph Droms, Peter Koch, Jinmei Tatuya, Iljitsch van Beijnum, Edward Lewis, and Rob Austein provided useful feedback during the WG last call. Thomas Narten provided extensive feedback during the IESG evaluation.

10. Security Considerations

This document reviews the operational procedures for IPv6 DNS operations and does not have security considerations in itself.

However, it is worth noting that in particular with Dynamic DNS updates, security models based on the source address validation are very weak and cannot be recommended -- they could only be considered in the environments where ingress filtering [RFC3704] has been deployed. On the other hand, it should be noted that setting up an authorization mechanism (e.g., a shared secret, or public-private keys) between a node and the DNS server has to be done manually, and may require quite a bit of time and expertise.

To re-emphasize what was already stated, the reverse+forward DNS check provides very weak security at best, and the only (questionable) security-related use for them may be in conjunction with other mechanisms when authenticating a user.

11. References

11.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC2182] Elz, R., Bush, R., Bradner, S., and M. Patton, "Selection and Operation of Secondary DNS Servers", BCP 16, RFC 2182, July 1997.

[RFC2462] Thomson, S. and T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, December 1998.

[RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, August 1999.

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, November 2000.

[RFC3041] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, January 2001.

[RFC3056] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, February 2001.

[RFC3152] Bush, R., "Delegation of IP6.ARPA", BCP 49, RFC 3152, August 2001.

[RFC3315] Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, July 2003.

[RFC3363] Bush, R., Durand, A., Fink, B., Gudmundsson, O., and T. Hain, "Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS)", RFC 3363, August 2002.

[RFC3364] Austein, R., "Tradeoffs in Domain Name System (DNS) Support for Internet Protocol version 6 (IPv6)", RFC 3364, August 2002.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", RFC 3596, October 2003.

[RFC3646] Droms, R., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, December 2003.

[RFC3736] Droms, R., "Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6", RFC 3736, April 2004.

[RFC3879] Huitema, C. and B. Carpenter, "Deprecating Site Local Addresses", RFC 3879, September 2004.

[RFC3901] Durand, A. and J. Ihren, "DNS IPv6 Transport Operational Guidelines", BCP 91, RFC 3901, September 2004.

[RFC4038] Shin, M-K., Hong, Y-G., Hagino, J., Savola, P., and E. Castro, "Application Aspects of IPv6 Transition", RFC 4038, March 2005.

[RFC4074] Morishita, Y. and T. Jinmei, "Common Misbehavior Against DNS Queries for IPv6 Addresses", RFC 4074, May 2005.

[RFC4192] Baker, F., Lear, E., and R. Droms, "Procedures for Renumbering an IPv6 Network without a Flag Day", RFC 4192, September 2005.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4339] Jeong, J., Ed., "IPv6 Host Configuration of DNS Server Information Approaches", RFC 4339, February 2006.

11.2. Informative References

[RFC2766] Tsirtsis, G. and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)", RFC 2766, February 2000.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC2826] Internet Architecture Board, "IAB Technical Comment on the Unique DNS Root", RFC 2826, May 2000.

[RFC3704] Baker, F. and P. Savola, "Ingress Filtering for Multihomed Networks", BCP 84, RFC 3704, March 2004.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.

[RFC4025] Richardson, M., "A Method for Storing IPsec Keying Material in DNS", RFC 4025, March 2005.

[RFC4213] Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.

[RFC4215] Wiljakka, J., "Analysis on IPv6 Transition in Third Generation Partnership Project (3GPP) Networks", RFC 4215, October 2005.

[RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, February 2006.

[TC-TEST] Jinmei, T., "Thread "RFC2181 section 9.1: TC bit handling and additional data" on DNSEXT mailing list, Message-Id:y7vek9j9hyo.wl%jinmei@isl.rdc.toshiba.co.jp", August 1, 2005, <http://ops.ietf.org/lists/namedroppers/namedroppers.2005/msg01102.html>.

[WIP-AD2005] Aoun, C. and E. Davies, "Reasons to Move NAT-PT to Experimental", Work in Progress, October 2005.

[WIP-DC2005] Durand, A. and T. Chown, "To publish, or not to publish, that is the question", Work in Progress, October 2005.

[WIP-H2005] Huston, G., "6to4 Reverse DNS Delegation Specification", Work in Progress, November 2005.

[WIP-J2006] Jeong, J., "IPv6 Router Advertisement Option for DNS Configuration", Work in Progress, January 2006.

[WIP-LB2005] Larson, M. and P. Barber, "Observed DNS Resolution Misbehavior", Work in Progress, February 2006.

[WIP-O2004] Ohta, M., "Preconfigured DNS Server Addresses", Work in Progress, February 2004.

[WIP-R2006] Roy, S., "IPv6 Neighbor Discovery On-Link Assumption Considered Harmful", Work in Progress, January 2006.

[WIP-RDP2004] Roy, S., Durand, A., and J. Paugh, "Issues with Dual Stack IPv6 on by Default", Work in Progress, July 2004.

[WIP-S2005a] Stapp, M., "The DHCP Client FQDN Option", Work in Progress, March 2006.

[WIP-S2005b] Stapp, M., "A DNS RR for Encoding DHCP Information (DHCID RR)", Work in Progress, March 2006.

[WIP-S2005c] Senie, D., "Encouraging the use of DNS IN-ADDR Mapping", Work in Progress, August 2005.

[WIP-SV2005] Stapp, M. and B. Volz, "Resolution of FQDN Conflicts among DHCP Clients", Work in Progress, March 2006.

Appendix A. Unique Local Addressing Considerations for DNS

Unique local addresses [RFC4193] have replaced the now-deprecated site-local addresses [RFC3879]. From the perspective of the DNS, the locally generated unique local addresses (LUL) and site-local addresses have similar properties.

The interactions with DNS come in two flavors: forward and reverse DNS.

To actually use local addresses within a site, this implies the deployment of a "split-faced" or a fragmented DNS name space, for the zones internal to the site, and the outsiders' view to it. The procedures to achieve this are not elaborated here. The implication is that local addresses must not be published in the public DNS.

To facilitate reverse DNS (if desired) with local addresses, the stub resolvers must look for DNS information from the local DNS servers, not, e.g., starting from the root servers, so that the local information may be provided locally. Note that the experience of private addresses in IPv4 has shown that the root servers get loaded for requests for private address lookups in any case. This requirement is discussed in [RFC4193].

Appendix B. Behavior of Additional Data in IPv4/IPv6 Environments

DNS responses do not always fit in a single UDP packet. We'll examine the cases that happen when this is due to too much data in the Additional section.

B.1. Description of Additional Data Scenarios

There are two kinds of additional data:

1. "critical" additional data; this must be included in all scenarios, with all the RRsets, and

2. "courtesy" additional data; this could be sent in full, with only a few RRsets, or with no RRsets, and can be fetched separately as well, but at the cost of additional queries.

The responding server can algorithmically determine which type the additional data is by checking whether it's at or below a zone cut.

Only those additional data records (even if sometimes carelessly termed "glue") are considered "critical" or real "glue" if and only if they meet the above-mentioned condition, as specified in Section 4.2.1 of [RFC1034].

Remember that resource record sets (RRsets) are never "broken up", so if a name has 4 A records and 5 AAAA records, you can either return all 9, all 4 A records, all 5 AAAA records, or nothing. In particular, notice that for the "critical" additional data getting all the RRsets can be critical.

In particular, [RFC2181] specifies (in Section 9) that:

a. if all the "critical" RRsets do not fit, the sender should set the TC bit, and the recipient should discard the whole response and retry using mechanism allowing larger responses such as TCP.

b. "courtesy" additional data should not cause the setting of the TC bit, but instead all the non-fitting additional data RRsets should be removed.

An example of the "courtesy" additional data is A/AAAA records in conjunction with MX records as shown in Section 4.4; an example of the "critical" additional data is shown below (where getting both the A and AAAA RRsets is critical with respect to the NS RR):

```
child.example.com.    IN   NS ns.child.example.com.
ns.child.example.com. IN    A 192.0.2.1
ns.child.example.com. IN AAAA 2001:db8::1
```

When there is too much "courtesy" additional data, at least the non-fitting RRsets should be removed [RFC2181]; however, as the additional data is not critical, even all of it could be safely removed.

When there is too much "critical" additional data, TC bit will have to be set, and the recipient should ignore the response and retry using TCP; if some data were to be left in the UDP response, the issue is which data could be retained.

However, the practice may differ from the specification. Testing and code analysis of three recent implementations [TC-TEST] confirm this. None of the tested implementations have a strict separation of critical and courtesy additional data, while some forms of additional data may be treated preferably. All the implementations remove some (critical or courtesy) additional data RRsets without setting the TC bit if the response would not otherwise fit.

Failing to discard the response with the TC bit or omitting critical information but not setting the TC bit lead to an unrecoverable problem. Omitting only some of the RRsets if all would not fit (but not setting the TC bit) leads to a performance problem. These are discussed in the next two subsections.

B.2. Which Additional Data to Keep, If Any?

NOTE: omitting some critical additional data instead of setting the TC bit violates a 'should' in Section 9 of RFC2181. However, as many implementations still do that [TC-TEST], operators need to understand its implications, and we describe that behavior as well.

If the implementation decides to keep as much data (whether "critical" or "courtesy") as possible in the UDP responses, it might be tempting to use the transport of the DNS query as a hint in either of these cases: return the AAAA records if the query was done over IPv6, or return the A records if the query was done over IPv4. However, this breaks the model of independence of DNS transport and resource records, as noted in Section 1.2.

With courtesy additional data, as long as enough RRsets will be removed so that TC will not be set, it is allowed to send as many complete RRsets as the implementations prefers. However, the implementations are also free to omit all such RRsets, even if complete. Omitting all the RRsets (when removing only some would suffice) may create a performance penalty, whereby the client may need to issue one or more additional queries to obtain necessary and/or consistent information.

With critical additional data, the alternatives are either returning nothing (and absolutely requiring a retry with TCP) or returning something (working also in the case if the recipient does not discard the response and retry using TCP) in addition to setting the TC bit. If the process for selecting "something" from the critical data would otherwise be practically "flipping the coin" between A and AAAA records, it could be argued that if one looked at the transport of the query, it would have a larger possibility of being right than just 50/50. In other words, if the returned critical additional data would have to be selected somehow, using something more sophisticated than a random process would seem justifiable.

That is, leaving in some intelligently selected critical additional data is a trade-off between creating an optimization for those resolvers that ignore the "should discard" recommendation and causing a protocol problem by propagating inconsistent information about "critical" records in the caches.

Similarly, leaving in the complete courtesy additional data RRsets instead of removing all the RRsets is a performance trade-off as described in the next section.

B.3. Discussion of the Potential Problems

As noted above, the temptation for omitting only some of the additional data could be problematic. This is discussed more below.

For courtesy additional data, this causes a potential performance problem as this requires that the clients issue re-queries for the potentially omitted RRsets. For critical additional data, this causes a potential unrecoverable problem if the response is not discarded and the query not re-tried with TCP, as the nameservers might be reachable only through the omitted RRsets.

If an implementation would look at the transport used for the query, it is worth remembering that often the host using the records is different from the node requesting them from the authoritative DNS server (or even a caching resolver). So, whichever version the requestor (e.g., a recursive server in the middle) uses makes no difference to the ultimate user of the records, whose transport capabilities might differ from those of the requestor. This might result in, e.g., inappropriately returning A records to an IPv6-only node, going through a translation, or opening up another IP-level session (e.g., a Packet Data Protocol (PDP) context [RFC4215]). Therefore, at least in many scenarios, it would be very useful if the information returned would be consistent and complete -- or if that is not feasible, leave it to the client to query again.

The problem of too much additional data seems to be an operational one: the zone administrator entering too many records that will be returned truncated (or missing some RRsets, depending on implementations) to the users. A protocol fix for this is using Extension Mechanisms for DNS (EDNS0) [RFC2671] to signal the capacity for larger UDP packet sizes, pushing up the relevant threshold. Further, DNS server implementations should omit courtesy additional data completely rather than including only some RRsets [RFC2181]. An operational fix for this is having the DNS server implementations return a warning when the administrators create zones that would result in too much additional data being returned. Further, DNS server implementations should warn of or disallow such zone configurations that are recursive or otherwise difficult to manage by the protocol.

Authors' Addresses

Alain Durand
Comcast
1500 Market St.
Philadelphia, PA 19102
USA

EMail: Alain_Durand@cable.comcast.com

Johan Ihren
Autonomica
Bellmansgatan 30
SE-118 47 Stockholm
Sweden

EMail: johani@autonomica.se

Pekka Savola
CSC/FUNET
Espoo
Finland

EMail: psavola@funet.fi

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).