BEHAVE WG
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2011

M. Bagnulo
UC3M
A. Sullivan
Shinkuro
P. Matthews
Alcatel-Lucent
I. van Beijnum
IMDEA Networks
July 5, 2010

DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers

draft-ietf-behave-dns64-10

Abstract

DNS64 is a mechanism for synthesizing AAAA records from A records. DNS64 is used with an IPv6/IPv4 translator to enable client-server communication between an IPv6-only client and an IPv4-only server, without requiring any changes to either the IPv6 or the IPv4 node, for the class of applications that work through NATs. This document specifies DNS64, and provides suggestions on how it should be deployed in conjunction with IPv6/IPv4 translators.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document specifies DNS64, a mechanism that is part of the toolbox for IPv6-IPv4 transition and co-existence. DNS64, used together with an IPv6/IPv4 translator such as stateful NAT64 [I-D.ietf-behave-v6v4-xlate-stateful], allows an IPv6-only client to initiate communications by name to an IPv4-only server.

DNS64 is a mechanism for synthesizing AAAA resource records (RRs) from A RRs. A synthetic AAAA RR created by the DNS64 from an original A RR contains the same owner name of the original A RR but it contains an IPv6 address instead of an IPv4 address. The IPv6 address is an IPv6 representation of the IPv4 address contained in the original A RR. The IPv6 representation of the IPv4 address is algorithmically generated from the IPv4 address returned in the A RR and a set of parameters configured in the DNS64 (typically, an IPv6 prefix used by IPv6 representations of IPv4 addresses and optionally other parameters).

Together with an IPv6/IPv4 translator, these two mechanisms allow an IPv6-only client to initiate communications to an IPv4-only server using the FQDN of the server.

These mechanisms are expected to play a critical role in the IPv4-IPv6 transition and co-existence. Due to IPv4 address depletion, it is likely that in the future, many IPv6-only clients will want to connect to IPv4-only servers. In the typical case, the approach only requires the deployment of IPv6/IPv4 translators that connect an IPv6-only network to an IPv4-only network, along with the deployment of one or more DNS64-enabled name servers. However, some advanced features require performing the DNS64 function directly in the end-hosts themselves.

2. Overview

This section provides a non-normative introduction to the DNS64 mechanism.

We assume that we have one or more IPv6/IPv4 translator boxes connecting an IPv4 network and an IPv6 network. The IPv6/IPv4 translator device provides translation services between the two networks enabling communication between IPv4-only hosts and IPv6-only hosts. (NOTE: By IPv6-only hosts we mean hosts running IPv6-only applications, hosts that can only use IPv6, as well as cases where only IPv6 connectivity is available to the client. By IPv4-only servers we mean servers running IPv4-only applications, servers that can only use IPv4, as well as cases where only IPv4 connectivity is

available to the server). Each IPv6/IPv4 translator used in conjunction with DNS64 must allow communications initiated from the IPv6-only host to the IPv4-only host.

To allow an IPv6 initiator to do a standard AAAA RR DNS lookup to learn the address of the responder, DNS64 is used to synthesize a AAAA record from an A record containing a real IPv4 address of the responder, whenever the DNS64 cannot retrieve a AAAA record for the queried name. The DNS64 service appears as a regular DNS server or resolver to the IPv6 initiator. The DNS64 receives a AAAA DNS query generated by the IPv6 initiator. It first attempts a resolution for the requested AAAA records. If there are no AAAA records available for the target node (which is the normal case when the target node is an IPv4-only node), DNS64 performs a query for A records. For each A record discovered, DNS64 creates a synthetic AAAA RR from the

information retrieved in the A RR.

The owner name of a synthetic AAAA RR is the same as that of the original A RR, but an IPv6 representation of the IPv4 address contained in the original A RR is included in the AAAA RR. The IPv6 representation of the IPv4 address is algorithmically generated from the IPv4 address and additional parameters configured in the DNS64. Among those parameters configured in the DNS64, there is at least one IPv6 prefix. If not explicitly mentioned, all prefixes are treated equally and the operations described in this document are performed using the prefixes available. So as to be general, we will call any of these prefixes Pref64::/n, and describe the operations made with the generic prefix Pref64::/n. The IPv6 address representing IPv4 addresses included in the AAAA RR synthesized by the DNS64 contain Pref64::/n and they also embed the original IPv4 address.

The same algorithm and the same Pref64::/n prefix(es) must be configured both in the DNS64 device and the IPv6/IPv4 translator(s), so that both can algorithmically generate the same IPv6 representation for a given IPv4 address. In addition, it is required that IPv6 packets addressed to an IPv6 destination address that contains the Pref64::/n be delivered to an IPv6/IPv4 translator that has that particular Pref64::/n configured, so they can be translated into IPv4 packets.

Once the DNS64 has synthesized the AAAA RRs, the synthetic AAAA RRs are passed back to the IPv6 initiator, which will initiate an IPv6 communication with the IPv6 address associated with the IPv4 receiver. The packet will be routed to an IPv6/IPv4 translator which will forward it to the IPv4 network.

In general, the only shared state between the DNS64 and the IPv6/IPv4 translator is the Pref64::/n and an optional set of static

parameters. The Pref64::/n and the set of static parameters must be configured to be the same on both; there is no communication between the DNS64 device and IPv6/IPv4 translator functions. The mechanism to be used for configuring the parameters of the DNS64 is beyond the scope of this memo.

The prefixes to be used as Pref64::/n and their applicability are discussed in [I-D.ietf-behave-address-format]. There are two types

of prefixes that can be used as Pref64::/n.

The Pref64::/n can be the Well-Known Prefix 64:FF9B::/96 reserved by [I-D.ietf-behave-address-format] for the purpose of representing IPv4 addresses in IPv6 address space.

The Pref64::/n can be a Network-Specific Prefix (NSP). An NSP is an IPv6 prefix assigned by an organization to create IPv6 representations of IPv4 addresses.

The main difference in the nature of the two types of prefixes is that the NSP is a locally assigned prefix that is under control of the organization that is providing the translation services, while the Well-Known Prefix is a prefix that has a global meaning since it has been assigned for the specific purpose of representing IPv4 addresses in IPv6 address space.

The DNS64 function can be performed in any of three places. The terms below are more formally defined in Section 4.

The first option is to locate the DNS64 function in authoritative servers for a zone. In this case, the authoritative server provides synthetic AAAA RRs for an IPv4-only host in its zone. This is one type of DNS64 server.

Another option is to locate the DNS64 function in recursive name servers serving end hosts. In this case, when an IPv6-only host queries the name server for AAAA RRs for an IPv4-only host, the name server can perform the synthesis of AAAA RRs and pass them back to the IPv6-only initiator. The main advantage of this mode is that current IPv6 nodes can use this mechanism without requiring any modification. This mode is called "DNS64 in DNS recursive resolver mode" . This is a second type of DNS64 server, and it is also one type of DNS64 resolver.

The last option is to place the DNS64 function in the end hosts, coupled to the local (stub) resolver. In this case, the stub resolver will try to obtain (real) AAAA RRs and in case they are not available, the DNS64 function will synthesize AAAA RRs for internal usage. This mode is compatible with some advanced functions like

DNSSEC validation in the end host. The main drawback of this mode is

its deployability, since it requires changes in the end hosts. This mode is called "DNS64 in stub-resolver mode". This is the second type of DNS64 resolver.

3. Background to DNS64-DNSSEC interaction

DNSSEC ([RFC4033], [RFC4034], [RFC4035]) presents a special challenge for DNS64, because DNSSEC is designed to detect changes to DNS answers, and DNS64 may alter answers coming from an authoritative server.

A recursive resolver can be security-aware or security-oblivious. Moreover, a security-aware recursive resolver can be validating or non-validating, according to operator policy. In the cases below, the recursive resolver is also performing DNS64, and has a local policy to validate. We call this general case vDNS64, but in all the cases below the DNS64 functionality should be assumed needed.

DNSSEC includes some signaling bits that offer some indicators of what the query originator understands.

If a query arrives at a vDNS64 device with the "DNSSEC OK" (DO) bit set, the query originator is signaling that it understands DNSSEC. The DO bit does not indicate that the query originator will validate the response. It only means that the query originator can understand responses containing DNSSEC data. Conversely, if the DO bit is clear, that is evidence that the querying agent is not aware of DNSSEC.

If a query arrives at a vDNS64 device with the "Checking Disabled" (CD) bit set, it is an indication that the querying agent wants all the validation data so it can do checking itself. By local policy, vDNS64 could still validate, but it must return all data to the querying agent anyway.

Here are the possible cases:

1. A DNS64 (DNSSEC-aware or DNSSEC-oblivious) receives a query with the DO bit clear. In this case, DNSSEC is not a concern, because the querying agent does not understand DNSSEC responses.

2. A security-oblivious DNS64 receives a query with the DO bit set, and the CD bit clear or set. This is just like the case of a non-DNS64 case: the server doesn't support it, so the querying agent is out of luck.

3. A security-aware and non-validating DNS64 receives a query with the DO bit set and the CD bit clear. Such a resolver is not validating responses, likely due to local policy (see [RFC4035], section 4.2). For that reason, this case amounts to the same as the previous case, and no validation happens.

4. A security-aware and non-validating DNS64 receives a query with the DO bit set and the CD bit set. In this case, the resolver is supposed to pass on all the data it gets to the query initiator (see section 3.2.2 of [RFC4035]). This case will not work with DNS64, unless the validating resolver is prepared to do DNS64 itself. If the DNS64 server modifies the record, the client will get the data back and try to validate it, and the data will be invalid as far as the client is concerned.

5. A security-aware and validating DNS64 node receives a query with the DO bit clear and CD clear. In this case, the resolver validates the data. If it fails, it returns RCODE 2 (Server failure); otherwise, it returns the answer. This is the ideal case for vDNS64. The resolver validates the data, and then synthesizes the new record and passes that to the client. The client, which is presumably not validating (else it should have set DO and CD), cannot tell that DNS64 is involved.

6. A security-aware and validating DNS64 node receives a query with the DO bit set and CD clear. This works like the previous case, except that the resolver should also set the "Authentic Data" (AD) bit on the response.

7. A security-aware and validating DNS64 node receives a query with the DO bit set and CD set. This is effectively the same as the case where a security-aware and non-validating recursive resolver receives a similar query, and the same thing will happen: the downstream validator will mark the data as invalid if DNS64 has performed synthesis. The node needs to do DNS64 itself, or else communication will fail.

4. Terminology

This section provides definitions for the special terms used in the document.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Authoritative server: A DNS server that can answer authoritatively a given DNS question.

DNS64: A logical function that synthesizes DNS resource records (e.g AAAA records containing IPv6 addresses) from DNS resource records actually contained in the DNS (e.g., A records containing IPv4 addresses).

DNS64 recursor: A recursive resolver that provides the DNS64 functionality as part of its operation. This is the same thing as "DNS64 in recursive resolver mode".

DNS64 resolver: Any resolver (stub resolver or recursive resolver) that provides the DNS64 function.

DNS64 server: Any server providing the DNS64 function.

Recursive resolver: A DNS server that accepts requests from one resolver, and asks another server (of some description) for the answer on behalf of the first resolver.

Synthetic RR: A DNS resource record (RR) that is not contained in any zone data file, but has been synthesized from other RRs. An example is a synthetic AAAA record created from an A record.

IPv6/IPv4 translator: A device that translates IPv6 packets to IPv4 packets and vice-versa. It is only required that the communication initiated from the IPv6 side be supported.

For a detailed understanding of this document, the reader should also be familiar with DNS terminology from [RFC1034], [RFC1035] and current NAT terminology from [RFC4787]. Some parts of this document assume familiarity with the terminology of the DNS security extensions outlined in [RFC4035]. It is worth emphasizing that while DNS64 is a logical function separate from the DNS, it is nevertheless closely associated with that protocol. It depends on the DNS protocol, and some behavior of DNS64 will interact with regular DNS responses.

5. DNS64 Normative Specification

DNS64 is a logical function that synthesizes AAAA records from A records. The DNS64 function may be implemented in a stub resolver, in a recursive resolver, or in an authoritative name server. It works within those DNS functions, and appears on the network as though it were a "plain" DNS resolver or name server conforming to [RFC1034], and [RFC1035].

The implementation SHOULD support mapping of separate IPv4 address ranges to separate IPv6 prefixes for AAAA record synthesis. This allows handling of special use IPv4 addresses [RFC5735].

DNS64 also responds to PTR queries involving addresses containing any of the IPv6 prefixes it uses for synthesis of AAAA RRs.

5.1. Resolving AAAA queries and the answer section

When the DNS64 receives a query for RRs of type AAAA and class IN, it first attempts to retrieve non-synthetic RRs of this type and class, either by performing a query or, in the case of an authoritative server, by examining its own results. The query may be answered from a local cache, if one is available. DNS64 operation for classes other than IN is undefined, and a DNS64 MUST behave as though no DNS64 function is configured.

5.1.1. The answer when there is AAAA data available

If the query results in one or more AAAA records in the answer section, the result is returned to the requesting client as per normal DNS semantics, except in the case where any of the AAAA records match a special exclusion set of prefixes, considered in Section 5.1.4. If there is (non-excluded) AAAA data available, DNS64 SHOULD NOT include synthetic AAAA RRs in the response (see Appendix A for an analysis of the motivations for and the implications of not complying with this recommendation). By default DNS64 implementations MUST NOT synthesize AAAA RRs when real AAAA RRs exist.

5.1.2. The answer when there is an error

If the query results in a response with RCODE other than 0 (No error condition), then there are two possibilities. A result with RCODE=3 (Name Error) is handled according to normal DNS operation (which is normally to return the error to the client). This stage is still prior to any synthesis having happened, so a response to be returned to the client does not need any special assembly than would usually happen in DNS operation.

Any other RCODE is treated as though the RCODE were 0 and the answer section were empty. This is because of the large number of different responses from deployed name servers when they receive AAAA queries without a AAAA record being available (see [RFC4074]). Note that this means, for practical purposes, that several different classes of error in the DNS are all treated as though a AAAA record is not available for that owner name.

It is important to note that, as of this writing, some servers respond with RCODE=3 to a AAAA query even if there is an A record available for that owner name. Those servers are in clear violation of the meaning of RCODE 3, and it is expected that they will decline in use as IPv6 deployment increases.

5.1.3. Dealing with timeouts

If the query receives no answer before the timeout (which might be the timeout from every authoritative server, depending on whether the DNS64 is in recursive resolver mode), it is treated as RCODE=2 (Server failure). .

5.1.4. Special exclusion set for AAAA records

Some IPv6 addresses are not actually usable by IPv6-only hosts. If they are returned to IPv6-only querying agents as AAAA records, therefore, the goal of decreasing the number of failure modes will not be attained. Examples include AAAA records with addresses in the ::ffff:0:0/96 network, and possibly (depending on the context) AAAA records with the site's Pref::64/n or the Well-Known Prefix (see below for more about the Well-Known Prefix). A DNS64 implementation SHOULD provide a mechanism to specify IPv6 prefix ranges to be treated as though the AAAA containing them were an empty answer. An implementation SHOULD include the ::ffff/96 network in that range by

default. Failure to provide this facility will mean that clients querying the DNS64 function may not be able to communicate with hosts that would be reachable from a dual-stack host.

When the DNS64 performs its initial AAAA query, if it receives an answer with only AAAA records containing addresses in the excluded range(s), then it MUST treat the answer as though it were an empty answer, and proceed accordingly. If it receives an answer with at least one AAAA record containing an address outside any of the excluded range(s), then it MAY build an answer section for a response including only the AAAA record(s) that do not contain any of the addresses inside the excluded ranges. That answer section is used in the assembly of a response as detailed in Section 5.4. Alternatively, it MAY treat the answer as though it were an empty answer, and proceed accordingly. It MUST NOT return the offending AAAA records as part of a response.

5.1.5. Dealing with CNAME and DNAME

If the response contains a CNAME or a DNAME, then the CNAME or DNAME chain is followed until the first terminating A or AAAA record is reached. This may require the DNS64 to ask for an A record, in case the response to the original AAAA query is a CNAME or DNAME without a

AAAA record to follow. The resulting AAAA or A record is treated like any other AAAA or A case, as appropriate.

When assembling the answer section, any chains of CNAME or DNAME RRs are included as part of the answer along with the synthetic AAAA (if appropriate).

5.1.6. Data for the answer when performing synthesis

If the query results in no error but an empty answer section in the response, the DNS64 attempts to retrieve A records for the name in question, either by performing another query or, in the case of an authoritative server, by examining its own results. If this new A RR query results in an empty answer or in an error, then the empty result or error is used as the basis for the answer returned to the querying client. If instead the query results in one or more A RRs, the DNS64 synthesizes AAAA RRs based on the A RRs according to the procedure outlined in Section 5.1.7. The DNS64 returns the

synthesized AAAA records in the answer section, removing the A records that form the basis of the synthesis.

5.1.7. Performing the synthesis

A synthetic AAAA record is created from an A record as follows:

- The NAME field is set to the NAME field from the A record
- The TYPE field is set to 28 (AAAA)
- The CLASS field is set to the original CLASS field, 1. Under this specification, DNS64 for any CLASS other than 1 is undefined.
- The TTL field is set to the minimum of the TTL of the original A RR and the SOA RR for the queried domain. (Note that in order to obtain the TTL of the SOA RR, the DNS64 does not need to perform a new query, but it can remember the TTL from the SOA RR in the negative response to the AAAA query. If the SOA RR was not delivered with the negative response to the AAAA query, then the DNS64 SHOULD use a default value of 600 seconds. It is possible instead to query explicitly for the SOA RR and use the result of that query, but this will increase query load and time to resolution for little additional benefit.) This is in keeping with the approach used in negative caching ([RFC2308]
- The RDLENGTH field is set to 16
- The RDATA field is set to the IPv6 representation of the IPv4 address from the RDATA field of the A record. The DNS64 SHOULD

check each A RR against configured IPv4 address ranges and select the corresponding IPv6 prefix to use in synthesizing the AAAA RR. See Section 5.2 for discussion of the algorithms to be used in effecting the transformation.

5.1.8. Querying in parallel

The DNS64 MAY perform the query for the AAAA RR and for the A RR in parallel, in order to minimize the delay. However, this would result in performing unnecessary A RR queries in the case where no AAAA RR synthesis is required. A possible trade-off would be to perform them

sequentially but with a very short interval between them, so if we obtain a fast reply, we avoid doing the additional query. (Note that this discussion is relevant only if the DNS64 function needs to perform external queries to fetch the RR. If the needed RR information is available locally, as in the case of an authoritative server, the issue is no longer relevant.)

5.2. Generation of the IPv6 representations of IPv4 addresses

DNS64 supports multiple algorithms for the generation of the IPv6 representation of an IPv4 address. The constraints imposed on the generation algorithms are the following:

The same algorithm to create an IPv6 address from an IPv4 address MUST be used by both a DNS64 to create the IPv6 address to be returned in the synthetic AAAA RR from the IPv4 address contained in an original A RR, and by a IPv6/IPv4 translator to create the IPv6 address to be included in the source address field of the outgoing IPv6 packets from the IPv4 address included in the source address field of the incoming IPv4 packet.

The algorithm MUST be reversible; i.e., it MUST be possible to derive the original IPv4 address from the IPv6 representation.

The input for the algorithm MUST be limited to the IPv4 address, the IPv6 prefix (denoted Pref64::/n) used in the IPv6 representations and optionally a set of stable parameters that are configured in the DNS64 and in the NAT64 (such as fixed string to be used as a suffix).

For each prefix Pref64::/n, n MUST be less than or equal to 96. If one or more Pref64::/n are configured in the DNS64 through any means (such as manually configured, or other automatic means not specified in this document), the default algorithm MUST use these prefixes (and not use the Well-Known Prefix). If no prefix is available, the algorithm MUST use the Well-Known Prefix 64:FF9B::/96 defined in

[I-D.ietf-behave-address-format] to represent the IPv4 unicast address range

[[anchor8: Note in document: The value 64:FF9B::/96 is proposed as

the value for the Well-Known prefix and needs to be confirmed whenis published as RFC.]][I-D.ietf-behave-address-format]

A DNS64 MUST support the algorithm for generating IPv6 representations of IPv4 addresses defined in Section 2 of [I-D.ietf-behave-address-format]. Moreover, the aforementioned algorithm MUST be the default algorithm used by the DNS64. While the normative description of the algorithm is provided in [I-D.ietf-behave-address-format], a sample description of the algorithm and its application to different scenarios is provided in Section 7 for illustration purposes.

5.3. Handling other Resource Records and the Additional Section

5.3.1. PTR Resource Record

If a DNS64 server receives a PTR query for a record in the IP6.ARPA domain, it MUST strip the IP6.ARPA labels from the QNAME, reverse the address portion of the QNAME according to the encoding scheme outlined in section 2.5 of [RFC3596], and examine the resulting address to see whether its prefix matches any of the locally-configured Pref64::/n. There are two alternatives for a DNS64 server to respond to such PTR queries. A DNS64 server MUST provide one of these, and SHOULD NOT provide both at the same time unless different IP6.ARPA zones require answers of different sorts:

1. The first option is for the DNS64 server to respond authoritatively for its prefixes. If the address prefix matches any Pref64::/n used in the site, either a NSP or the Well-Known Prefix (i.e. 64:FF9B::/96), then the DNS64 server MAY answer the query using locally-appropriate RDATA. The DNS64 server MAY use the same RDATA for all answers. Note that the requirement is to match any Pref64::/n used at the site, and not merely the locally-configured Pref64::/n. This is because end clients could ask for a PTR record matching an address received through a different (site-provided) DNS64, and if this strategy is in effect, those queries should never be sent to the global DNS. The advantage of this strategy is that it makes plain to the querying client that the prefix is one operated by the (DNS64) site, and that the answers the client is getting are generated by DNS64. The disadvantage is that any useful reverse-tree information that might be in the global DNS is unavailable to the clients querying the DNS64.

2. The second option is for the DNS64 nameserver to synthesize a CNAME mapping the IP6.ARPA namespace to the corresponding IN-ADDR.ARPA name. The rest of the response would be the normal DNS processing. The CNAME can be signed on the fly if need be. The advantage of this approach is that any useful information in the reverse tree is available to the querying client. The disadvantage is that it adds additional load to the DNS64 (because CNAMEs have to be synthesized for each PTR query that matches the Pref64::/n), and that it may require signing on the fly. In addition, the generated CNAME could correspond to an unpopulated in-addr.arpa zone, so the CNAME would provide a reference to a non-existent record.

If the address prefix does not match any Pref64::/n, then the DNS64 server MUST process the query as though it were any other query; i.e. a recursive nameserver MUST attempt to resolve the query as though it were any other (non-A/AAAA) query, and an authoritative server MUST respond authoritatively or with a referral, as appropriate.

5.3.2. Handling the additional section

DNS64 synthesis MUST NOT be performed on any records in the additional section of synthesized answers. The DNS64 MUST pass the additional section unchanged.

It may appear that adding synthetic records to the additional section is desirable, because clients sometimes use the data in the additional section to proceed without having to re-query. There is in general no promise, however, that the additional section will contain all the relevant records, so any client that depends on the additional section being able to satisfy its needs (i.e. without additional queries) is necessarily broken. An IPv6-only client that needs a AAAA record, therefore, will send a query for the necessary AAAA record if it is unable to find such a record in the additional section of an answer it is consuming. For a correctly-functioning client, the effect would be no different if the additional section were empty.

The alternative, of removing the A records in the additional section and replacing them with synthetic AAAA records, may cause a host behind a NAT64 to query directly a nameserver that is unaware of the NAT64 in question. The result in this case will be resolution failure anyway, only later in the resolution operation.

The prohibition on synthetic data in the additional section reduces, but does not eliminate, the possibility of resolution failures due to cached DNS data from behind the DNS64. See Section 6.

5.3.3. Other Resource Records

If the DNS64 is in recursive resolver mode, then considerations outlined in [I-D.ietf-dnsop-default-local-zones] may be relevant.

All other RRs MUST be returned unchanged. This includes responses to queries for A RRs.

5.4. Assembling a synthesized response to a AAAA query

A DNS64 uses different pieces of data to build the response returned to the querying client.

The query that is used as the basis for synthesis results either in an error, an answer that can be used as a basis for synthesis, or an empty (authoritative) answer. If there is an empty answer, then the DNS64 responds to the original querying client with the answer the DNS64 received to the original (initiator's) query. Otherwise, the response is assembled as follows.

The header fields are set according to the usual rules for recursive or authoritative servers, depending on the role that the DNS64 is serving. The question section is copied from the original (initiator's) query. The answer section is populated according to the rules in Section 5.1.7. The authority and additional sections are copied from the response to the final query that the DNS64 performed, and used as the basis for synthesis.

The final response from the DNS64 is subject to all the standard DNS rules, including truncation [RFC1035] and EDNS0 handling [RFC2671].

5.5. DNSSEC processing: DNS64 in recursive resolver mode

We consider the case where a recursive resolver that is performing DNS64 also has a local policy to validate the answers according to the procedures outlined in [RFC4035] Section 5. We call this general case vDNS64.

The vDNS64 uses the presence of the DO and CD bits to make some decisions about what the query originator needs, and can react

accordingly:

1. If CD is not set and DO is not set, vDNS64 SHOULD perform validation and do synthesis as needed. See the next item for rules about how to do validation and synthesis. In this case, however, vDNS64 MUST NOT set the AD bit in any response.

2. If CD is not set and DO is set, then vDNS64 SHOULD perform validation. Whenever vDNS64 performs validation, it MUST validate the negative answer for AAAA queries before proceeding to query for A records for the same name, in order to be sure that there is not a legitimate AAAA record on the Internet. Failing to observe this step would allow an attacker to use DNS64 as a mechanism to circumvent DNSSEC. If the negative response validates, and the response to the A query validates, then the vDNS64 MAY perform synthesis and SHOULD set the AD bit in the answer to the client. This is acceptable, because [RFC4035], section 3.2.3 says that the AD bit is set by the name server side of a security-aware recursive name server if and only if it considers all the RRSets in the Answer and Authority sections to be authentic. In this case, the name server has reason to believe the RRSets are all authentic, so it SHOULD set the AD bit. If the data does not validate, the vDNS64 MUST respond with RCODE=2 (Server failure).
   A security-aware end point might take the presence of the AD bit as an indication that the data is valid, and may pass the DNS (and DNSSEC) data to an application. If the application attempts to validate the synthesized data, of course, the validation will fail. One could argue therefore that this approach is not desirable, but security aware stub resolvers must not place any reliance on data received from resolvers and validated on their behalf without certain criteria established by [RFC4035], section 4.9.3. An application that wants to perform validation on its own should use the CD bit.

3. If the CD bit is set and DO is set, then vDNS64 MAY perform validation, but MUST NOT perform synthesis. It MUST return the data to the query initiator, just like a regular recursive resolver, and depend on the client to do the validation and the synthesis itself.

The disadvantage to this approach is that an end point that is translation-oblivious but security-aware and validating will not be able to use the DNS64 functionality. In this case, the end point will not have the desired benefit of NAT64. In effect, this strategy means that any end point that wishes to do validation in a NAT64 context must be upgraded to be translation-aware as well.

6. Deployment notes

While DNS64 is intended to be part of a strategy for aiding IPv6 deployment in an internetworking environment with some IPv4-only and IPv6-only networks, it is important to realise that it is incompatible with some things that may be deployed in an IPv4-only or

dual-stack context.

6.1. DNS resolvers and DNS64

Full-service resolvers that are unaware of the DNS64 function can be (mis)configured to act as mixed-mode iterative and forwarding resolvers. In a native IPv4 context, this sort of configuration may appear to work. It is impossible to make it work properly without it being aware of the DNS64 function, because it will likely at some point obtain IPv4-only glue records and attempt to use them for resolution. The result that is returned will contain only A records, and without the ability to perform the DNS64 function the resolver will be unable to answer the necessary AAAA queries.

6.2. DNSSEC validators and DNS64

An existing DNSSEC validator (i.e. that is unaware of DNS64) might reject all the data that comes from DNS64 as having been tampered with (even if it did not set CD when querying). If it is necessary to have validation behind the DNS64, then the validator must know how to perform the DNS64 function itself. Alternatively, the validating host may establish a trusted connection with a DNS64, and allow the DNS64 recursor to do all validation on its behalf.

6.3. DNS64 and multihomed and dual-stack hosts

6.3.1. IPv6 multihomed hosts

Synthetic AAAA records may be constructed on the basis of the network context in which they were constructed. If a host sends DNS queries to resolvers in multiple networks, it is possible that some of them will receive answers from a DNS64 without all of them being connected via a NAT64. For instance, suppose a system has two interfaces, i1 and i2. Whereas i1 is connected to the IPv4 Internet via NAT64, i2 has native IPv6 connectivity only. I1 might receive a AAAA answer from a DNS64 that is configured for a particular NAT64; the IPv6 address contained in that AAAA answer will not connect with anything via i2.

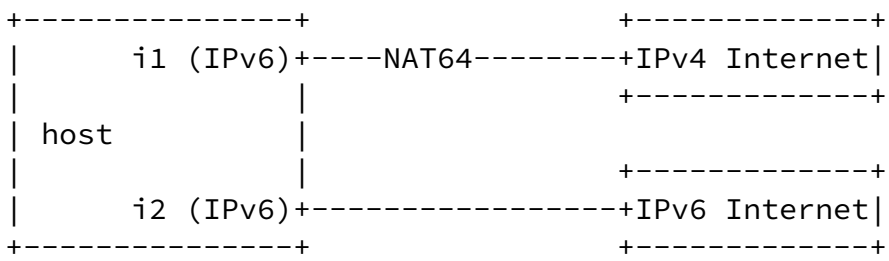


Figure 1: IPv6 multihomed hosts

This example illustrates why it is generally preferable that hosts treat DNS answers from one interface as local to that interface. The answer received on one interface will not work on the other interface. Hosts that attempt to use DNS answers globally may encounter surprising failures in these cases.

Note that the issue is not that there are two interfaces, but that there are two networks involved. The same results could be achieved with a single interface routed to two different networks.

6.3.2. Accidental dual-stack DNS64 use

Similarly, suppose that i1 has IPv6 connectivity and can connect to the IPv4 Internet through NAT64, but i2 has native IPv4 connectivity. In this case, i1 could receive an IPv6 address from a synthetic AAAA that would better be reached via native IPv4. Again, it is worth emphasising that this arises because there are two networks involved.

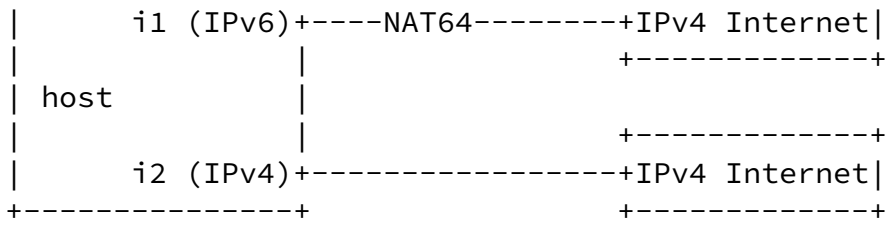


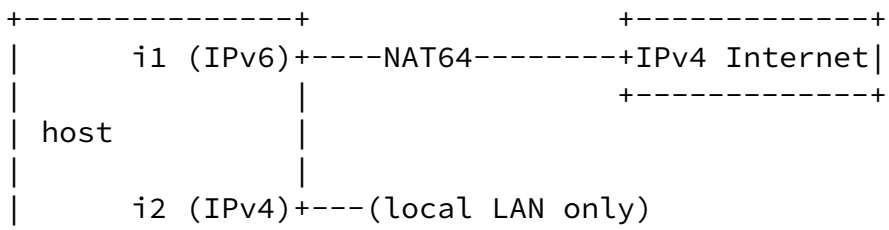
Figure 2: Accidental dual-stack DNS64 use

The default configuration of dual-stack hosts is that IPv6 is preferred over IPv4 ([RFC3484]). In that arrangement the host will often use the NAT64 when native IPv4 would be more desirable. For this reason, hosts with IPv4 connectivity to the Internet should avoid using DNS64. This can be partly resolved by ISPs when providing DNS resolvers to clients, but that is not a guarantee that the NAT64 will never be used when a native IPv4 connection should be used. There is no general-purpose mechanism to ensure that native IPv4 transit will always be preferred, because to a DNS64-oblivious host, the DNS64 looks just like an ordinary DNS server. Operators of a NAT64 should expect traffic to pass through the NAT64 even when it is not necessary.

6.3.3. Intentional dual-stack DNS64 use

Finally, consider the case where the IPv4 connectivity on i2 is only with a LAN, and not with the IPv4 Internet. The IPv4 Internet is only accessible using the NAT64. In this case, it is critical that the DNS64 not synthesize AAAA responses for hosts in the LAN, or else that the DNS64 be aware of hosts in the LAN and provide context-

sensitive answers ("split view" DNS answers) for hosts inside the LAN. As with any split view DNS arrangement, operators must be prepared for data to leak from one context to another, and for failures to occur because nodes accessible from one context are not accessible from the other.



```
            +---------------+
```

Figure 3: Intentional dual-stack DNS64 use

It is important for deployers of DNS64 to realise that, in some circumstances, making the DNS64 available to a dual-stack host will cause the host to prefer to send packets via NAT64 instead of via native IPv4, with the associated loss of performance or functionality (or both) entailed by the NAT. At the same time, some hosts are not able to learn about DNS servers provisioned on IPv6 addresses, or simply cannot send DNS packets over IPv6.

7. Deployment scenarios and examples

In this section, we walk through some sample scenarios that are expected to be common deployment cases. It should be noted that this is provided for illustrative purposes and this section is not normative. The normative definition of DNS64 is provided in Section 5 and the normative definition of the address transformation algorithm is provided in [I-D.ietf-behave-address-format].

In this section we illustrate how the DNS64 behaves in different scenarios that are expected to be common. In particular we will consider the following scenarios defined in [I-D.ietf-behave-v6v4-framework]: the an-IPv6-network-to-IPv4-Internet scenario (both with DNS64 in DNS server mode and in stub-resolver mode) and the IPv6-Internet-to-an-IPv4-network setup (with DNS64 in DNS server mode only).

In all the examples below, there is a IPv6/IPv4 translator connecting the IPv6 domain to the IPv4 one. Also there is a name server that is a dual-stack node, so it can communicate with IPv6 hosts using IPv6 and with IPv4 nodes using IPv4. In addition, we assume that in the examples, the DNS64 function learns which IPv6 prefix it needs to use to map the IPv4 address space through manual configuration.

7.1. Example of An-IPv6-network-to-IPv4-Internet setup with DNS64 in DNS server mode

In this example, we consider an IPv6 node located in an IPv6-only site that initiates a communication to an IPv4 node located in the

IPv4 Internet.

The scenario for this case is depicted in the following figure:

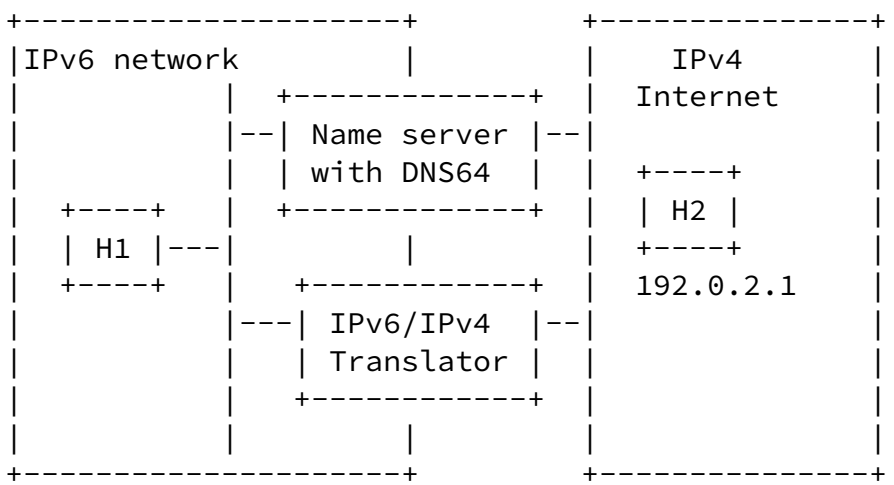


Figure 4: An-IPv6-network-to-IPv4-Internet setup with DNS64 in DNS server mode

The figure shows an IPv6 node H1 and an IPv4 node H2 with IPv4 address 192.0.2.1 and FQDN h2.example.com

The IPv6/IPv4 Translator has an IPv4 address 203.0.113.1 assigned to its IPv4 interface and it is using the WKP 64:FF9B::/96 to create IPv6 representations of IPv4 addresses. The same prefix is configured in the DNS64 function in the local name server.

For this example, assume the typical DNS situation where IPv6 hosts have only stub resolvers, and they are configured with the IP address of a name server that they always have to query and that performs recursive lookups (henceforth called "the recursive nameserver").

The steps by which H1 establishes communication with H2 are:

1. H1 does a DNS lookup for h2.example.com. H1 does this by sending a DNS query for a AAAA record for H2 to the recursive name server. The recursive name server implements DNS64 functionality.

2. The recursive name server resolves the query, and discovers that there are no AAAA records for H2.

3. The recursive name server performs an A-record query for H2 and gets back an RRset containing a single A record with the IPv4 address 192.0.2.1. The name server then synthesizes a AAAA record. The IPv6 address in the AAAA record contains the prefix assigned to the IPv6/IPv4 Translator in the upper 96 bits and the received IPv4 address in the lower 32 bits i.e. the resulting IPv6 address is 64:FF9B::192.0.2.1

4. H1 receives the synthetic AAAA record and sends a packet towards H2. The packet is sent to the destination address 64:FF9B:: 192.0.2.1.

5. The packet is routed to the IPv6 interface of the IPv6/IPv4 translator and the subsequent communication flows by means of the IPv6/IPv4 translator mechanisms.

7.2. An example of an-IPv6-network-to-IPv4-Internet setup with DNS64 in stub-resolver mode

This case is depicted in the following figure:

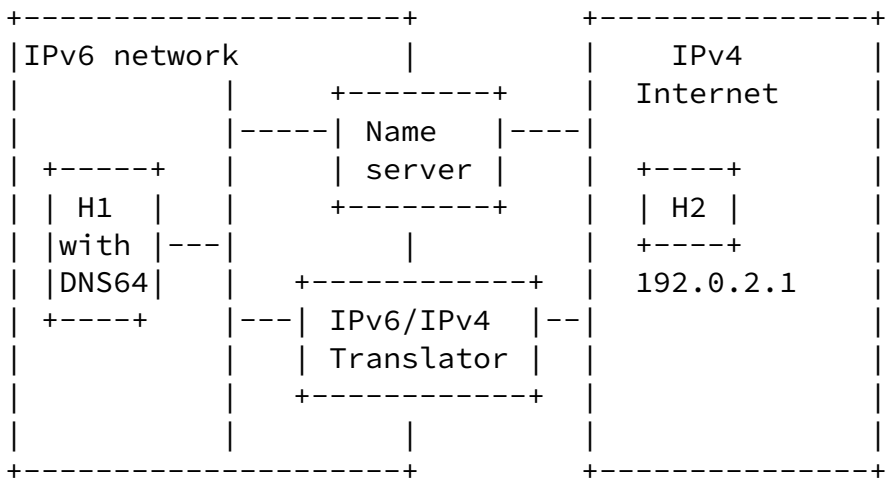


Figure 5: An-IPv6-network-to-IPv4-Internet setup with DNS64 in stub-resolver mode

The figure shows an IPv6 node H1 implementing the DNS64 function and an IPv4 node H2 with IPv4 address 192.0.2.1 and FQDN h2.example.com

The IPv6/IPv4 Translator has an IPv4 address 203.0.113.1 assigned to its IPv4 interface and it is using the WKP 64:FF9B::/96 to create

IPv6 representations of IPv4 addresses. The same prefix is configured in the DNS64 function in H1.

For this example, assume the typical DNS situation where IPv6 hosts have only stub resolvers, and they are configured with the IP address of a name server that they always have to query and that performs recursive lookups (henceforth called "the recursive nameserver"). The recursive name server does not perform the DNS64 function.

The steps by which H1 establishes communication with H2 are:

1. H1 does a DNS lookup for h2.example.com. H1 does this by sending a DNS query for a AAAA record for H2 to the recursive name server.

2. The recursive DNS server resolves the query, and returns the answer to H1. Because there are no AAAA records in the global DNS for H2, the answer is empty.

3. The stub resolver at H1 then queries for an A record for H2 and gets back an A record containing the IPv4 address 192.0.2.1. The DNS64 function within H1 then synthesizes a AAAA record. The IPv6 address in the AAAA record contains the prefix assigned to the IPv6/IPv4 translator in the upper 96 bits, then the received IPv4 address i.e. the resulting IPv6 address is 64:FF9B:: 192.0.2.1.

4. H1 sends a packet towards H2. The packet is sent to the destination address 64:FF9B::192.0.2.1.

5. The packet is routed to the IPv6 interface of the IPv6/IPv4 translator and the subsequent communication flows using the IPv6/ IPv4 translator mechanisms.

7.3. Example of IPv6-Internet-to-an-IPv4-network setup DNS64 in DNS server mode

In this example, we consider an IPv6 node located in the IPv6 Internet that initiates a communication to an IPv4 node located in the IPv4 site.

In some cases, this scenario can be addressed without using any form

of DNS64 function. This is so because it is possible to assign a fixed IPv6 address to each of the IPv4 nodes. Such an IPv6 address would be constructed using the address transformation algorithm defined in [I-D.ietf-behave-address-format] that takes as input the Pref64::/96 and the IPv4 address of the IPv4 node. Note that the IPv4 address can be a public or a private address; the latter does

not present any additional difficulty, since an NSP must be used as Pref64::/96 (in this scenario the usage of the Well-Known prefix is not supported as discussed in [I-D.ietf-behave-address-format]). Once these IPv6 addresses have been assigned to represent the IPv4 nodes in the IPv6 Internet, real AAAA RRs containing these addresses can be published in the DNS under the site's domain. This is the recommended approach to handle this scenario, because it does not involve synthesizing AAAA records at the time of query.

However, there are some more dynamic scenarios, where synthesizing AAAA RRs in this setup may be needed. In particular, when DNS Update [RFC2136] is used in the IPv4 site to update the A RRs for the IPv4 nodes, there are two options: One option is to modify the DNS server that receives the dynamic DNS updates. That would normally be the authoritative server for the zone. So the authoritative zone would have normal AAAA RRs that are synthesized as dynamic updates occur. The other option is modify all the authoritative servers to generate synthetic AAAA records for a zone, possibly based on additional constraints, upon the receipt of a DNS query for the AAAA RR. The first option -- in which the AAAA is synthesized when the DNS update message is received, and the data published in the relevant zone -- is recommended over the second option (i.e. the synthesis upon receipt of the AAAA DNS query). This is because it is usually easier to solve problems of misconfiguration when the DNS responses are not being generated dynamically. However, it may be the case where the primary server (that receives all the updates) cannot be upgraded for whatever reason, but where a secondary can be upgraded in order to handle the (comparatively small amount) of AAAA queries. In such case, it is possible to use the DNS64 as described next. The DNS64 behavior that we describe in this section covers the case of synthesizing the AAAA RR when the DNS query arrives.

The scenario for this case is depicted in the following figure:

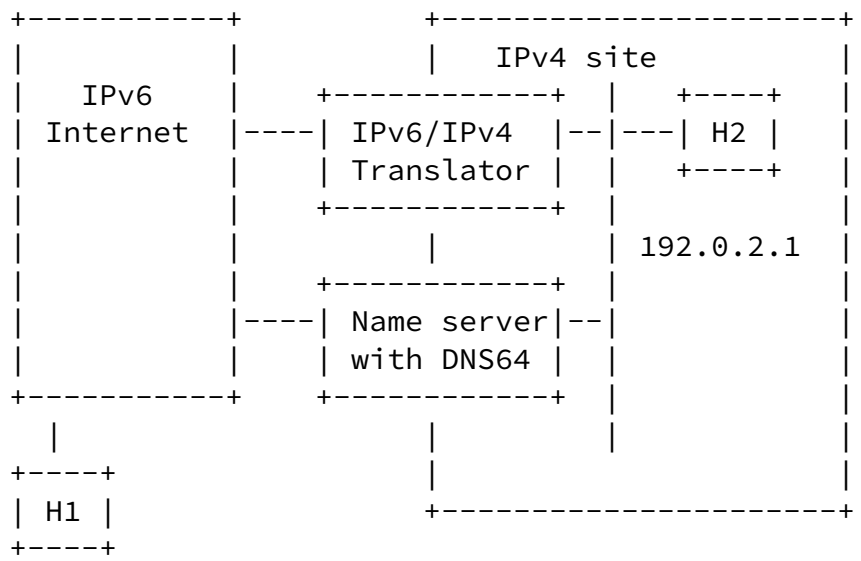


Figure 6: IPv6-Internet-to-an-IPv4-network setup DNS64 in DNS server mode

The figure shows an IPv6 node H1 and an IPv4 node H2 with IPv4 address 192.0.2.1 and FQDN h2.example.com.

The IPv6/IPv4 Translator is using a NSP 2001:DB8::/96 to create IPv6 representations of IPv4 addresses. The same prefix is configured in the DNS64 function in the local name server. The name server that implements the DNS64 function is the authoritative name server for the local domain.

The steps by which H1 establishes communication with H2 are:

1. H1 does a DNS lookup for h2.example.com. H1 does this by sending a DNS query for a AAAA record for H2. The query is eventually forwarded to the server in the IPv4 site.

2. The local DNS server resolves the query (locally), and discovers that there are no AAAA records for H2.

3. The name server verifies that h2.example.com and its A RR are among those that the local policy defines as allowed to generate a AAAA RR from. If that is the case, the name server synthesizes a AAAA record from the A RR and the prefix 2001:DB8::/96. The IPv6 address in the AAAA record is 2001:DB8::192.0.2.1.

4. H1 receives the synthetic AAAA record and sends a packet towards H2. The packet is sent to the destination address 2001:DB8::192.0.2.1.

5. The packet is routed through the IPv6 Internet to the IPv6 interface of the IPv6/IPv4 translator and the communication flows using the IPv6/IPv4 translator mechanisms.

8. Security Considerations

DNS64 operates in combination with the DNS, and is therefore subject to whatever security considerations are appropriate to the DNS mode in which the DNS64 is operating (i.e. authoritative, recursive, or stub resolver mode).

DNS64 has the potential to interfere with the functioning of DNSSEC, because DNS64 modifies DNS answers, and DNSSEC is designed to detect such modification and to treat modified answers as bogus. See the discussion above in Section 3, Section 5.5, and Section 6.2.

9. IANA Considerations

This memo makes no request of IANA.

10. Contributors

Dave Thaler

Microsoft

dthaler@windows.microsoft.com

11. Acknowledgements

This draft contains the result of discussions involving many people, including the participants of the IETF BEHAVE Working Group. The following IETF participants made specific contributions to parts of the text, and their help is gratefully acknowledged: Jaap Akkerhuis, Mark Andrews, Jari Arkko, Rob Austein, Timothy Baldwin, Fred Baker, Doug Barton, Marc Blanchet, Cameron Byrne, Brian Carpenter, Zhen Cao, Hui Deng, Francis Dupont, Patrik Faltstrom, David Harrington, Ed Jankiewicz, Peter Koch, Suresh Krishnan, Martti Kuparinen, Ed Lewis, Xing Li, Bill Manning, Matthijs Mekking, Hiroshi Miyata, Simon Perrault, Teemu Savolainen, Jyrki Soini, Dave Thaler, Mark Townsley, Rick van Rein, Stig Venaas, Magnus Westerlund, Jeff Westhead, Florian Weimer, Dan Wing, Xu Xiaohu, Xiangsong Cui.

Marcelo Bagnulo and Iljitsch van Beijnum are partly funded by

Trilogy, a research project supported by the European Commission under its Seventh Framework Program.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

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

[I-D.ietf-behave-address-format] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", draft-ietf-behave-address-format-08 (work in progress), May 2010.

12.2. Informative References

[I-D.ietf-behave-v6v4-xlate-stateful] Bagnulo, M., Matthews, P., and I. Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", draft-ietf-behave-v6v4-xlate-stateful-11 (work in progress), March 2010.

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

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.

[RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.

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

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

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

[RFC5735] Cotton, M. and L. Vegoda, "Special Use IPv4 Addresses", BCP 153, RFC 5735, January 2010.

[I-D.ietf-behave-v6v4-framework] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", draft-ietf-behave-v6v4-framework-09 (work in progress), May 2010.

[I-D.ietf-dnsop-default-local-zones] Andrews, M., "Locally-served DNS Zones", draft-ietf-dnsop-default-local-zones-13 (work in progress), April 2010.

Appendix A. Motivations and Implications of synthesizing AAAA Resource Records when real AAAA Resource Records exist

The motivation for synthesizing AAAA RRs when real AAAA RRs exist is to support the following scenario:

An IPv4-only server application (e.g. web server software) is running on a dual-stack host. There may also be dual-stack server applications running on the same host. That host has fully routable IPv4 and IPv6 addresses and hence the authoritative DNS server has an A and a AAAA record.

An IPv6-only client (regardless of whether the client application is IPv6-only, the client stack is IPv6-only, or it only has an

IPv6 address) wants to access the above server.

The client issues a DNS query to a DNS64 resolver.

If the DNS64 only generates a synthetic AAAA if there's no real AAAA, then the communication will fail. Even though there's a real AAAA, the only way for communication to succeed is with the translated address. So, in order to support this scenario, the administrator of a DNS64 service may want to enable the synthesis of AAAA RRs even when real AAAA RRs exist.

The implication of including synthetic AAAA RRs when real AAAA RRs exist is that translated connectivity may be preferred over native connectivity in some cases where the DNS64 is operated in DNS server mode.

RFC3484 [RFC3484] rules use longest prefix match to select the preferred destination address to use. So, if the DNS64 resolver returns both the synthetic AAAA RRs and the real AAAA RRs, then if the DNS64 is operated by the same domain as the initiating host, and a global unicast prefix (called an NSP in [I-D.ietf-behave-address-format]) is used, then a synthetic AAAA RR is likely to be preferred.

This means that without further configuration:

In the "An IPv6 network to the IPv4 Internet" scenario, the host will prefer translated connectivity if an NSP is used. If the Well-Known Prefix defined in [I-D.ietf-behave-address-format] is used, it will probably prefer native connectivity.

In the "IPv6 Internet to an IPv4 network" scenario, it is possible to bias the selection towards the real AAAA RR if the DNS64 resolver returns the real AAAA first in the DNS reply, when an NSP is used (the Well-Known Prefix usage is not supported in this case)

In the "An IPv6 network to IPv4 network" scenario, for local destinations (i.e., target hosts inside the local site), it is likely that the NSP and the destination prefix are the same, so we can use the order of RR in the DNS reply to bias the selection through native connectivity. If the Well-Known Prefix is used, the longest prefix match rule will select native connectivity.

The problem can be solved by properly configuring the RFC3484 [RFC3484] policy table.

Authors' Addresses

Marcelo Bagnulo
UC3M
Av. Universidad 30
Leganes, Madrid 28911
Spain

Phone: +34-91-6249500
Fax:
Email: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es/marcelo

Andrew Sullivan
Shinkuro
4922 Fairmont Avenue, Suite 250
Bethesda, MD 20814
USA

Phone: +1 301 961 3131
Email: ajs@shinkuro.com

Philip Matthews
Unaffiliated
600 March Road
Ottawa, Ontario
Canada

Phone: +1 613-592-4343 x224
Fax:
Email: philip_matthews@magma.ca
URI:

Iljitsch van Beijnum
IMDEA Networks
Av. Universidad 30
Leganes, Madrid 28911
Spain

Phone: +34-91-6246245

Email: iljitsch@muada.com