DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers
draft-ietf-behave-dns64-00

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on January 5, 2010.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
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.

Table of Contents

1. Introduction .................................................. 3
2. Overview ...................................................... 3
3. Overview and rationale for DNS64 - DNSSEC interaction .......... 5
4. Terminology ..................................................... 7
5. DNS64 Normative Specification .................................. 8
   5.1. Resolving AAAA queries and the answer section ............ 8
   5.2. Handling the additional section ........................... 9
   5.3. Performing the synthesis ................................... 9
   5.3.1. Generation of the IPv6 representations of IPv4 addresses ................................................. 10
   5.4. Handling other RRs ......................................... 11
   5.4.1. PTR queries ............................................ 11
   5.5. DNSSEC processing: DNS64 in recursive server mode ....... 12
6. Appendixes ...................................................... 13
   6.1. Deployment scenarios and examples .......................... 13
   6.1.1. Embed and Zero-Pad algorithm description .............. 14
   6.1.2. An-IPv6-network-to-IPv4-Internet setup with DNS64 in DNS server mode ................................. 15
   6.1.3. An-IPv6-network-to-IPv4-Internet setup with DNS64 in stub-resolver mode ............................. 17
   6.1.4. IPv6-Internet-to-an-IPv4-network setup DNS64 in DNS server mode ...................................... 18
6.2. Motivations and Implications of synthesizing AAAA RR when real AAAA RR exists .................................. 20
7. Security Considerations ........................................ 22
8. IANA Considerations ............................................ 22
9. Contributors ..................................................... 22
10. Acknowledgements ............................................... 22
11. References ...................................................... 22
   11.1. Normative References ..................................... 22
   11.2. Informative References ................................... 23
Authors’ Addresses ................................................ 25
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 NAT64 [I-D.bagnulo-behave-nat64], 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 FQDN 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 a 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’s likely that in the future, a lot of 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 by the end-hosts themselves.

2. Overview

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

We assume that we have an IPv6/IPv4 translator box 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, IPv6-only hosts, as well as the cases where only IPv6-only connectivity is available to the client. By IPv4-only servers we mean IPv4-only servers, as well as the cases where only IPv4 connectivity is available to the server). The 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 service cannot retrieve a AAAA record for the requested host name. The DNS64 device appears as a regular recursive resolver for the IPv6 initiator. The DNS64 box receives an AAAA DNS query generated by the IPv6 initiator. It first attempts a recursive resolution for the requested AAAA records. If there is no AAAA record 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. If any A records are discovered, DNS64 creates a synthetic AAAA RR from the information retrieved in each A RR.

The FQDN 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 an IPv6 prefix, called Pref64::/n. The IPv6 address representing IPv4 addresses included in the AAAA RR synthesized by the DNS64 function contain Pref64::/n and they also embed the original IPv4 address.

The same algorithm and the same Pref64::/n prefix must be configured both in the DNS64 device and the IPv6/IPv4 translator, 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 that contains the Pref64::/n be delivered to the IPv6/IPv4 translator, so they can be translated into IPv4 packets.

Once the DNS64 has synthesized the AAAA RR, the synthetic AAAA RR is 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 the 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 DNS64 function can be performed in two places.
One 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 server mode".

The other option is to place the DNS64 function in the end hosts themselves, 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".

3. Overview and rationale for DNS64 - DNSSEC interaction

DNSSEC 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 name server can be validating or non-validating, according to operator policy. In the cases below, the recursive server 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 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 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 security-oblivious recursive resolver 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 recursive resolver receives a query with the DO bit set, and the CD bit clear. 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 recursive resolver 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 recursive resolver 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 (this is in section 3.2.2 of [RFC4035]). This is a case will be problematic for vNAT64. If it 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 recursive resolver 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 (SERVFAIL); 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.

6. A security-aware and validating recursive resolver receives a query with the DO bit set and CD clear. In principle, this ought to work like the previous case, except that the resolver should also set the AD bit on the response.

7. A security-aware and validating recursive resolver receives a query with the DO bit set and CD set. This is effectively the same as the case where an 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.
4. Terminology

This section provides a definitive reference for all the terms used in 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].

The following terms are used in this document:

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

DNS64: A logical function that synthesizes AAAA records (containing IPv6 addresses) from A records (containing IPv4 addresses).

DNS64 recursor: A recursive resolver that provides the DNS64 functionality as part of its operation.

Recursive resolver: A DNS server that accepts requests from one resolver, and asks another resolver for the answer on behalf of the first resolver. In the context of this document, "the recursive resolver" means a recursive resolver immediately next in the DNS resolution chain from an end point. The end point usually has only a stub resolver available.

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.

Stub resolver: A resolver with minimum functionality, typically for use in end points that depend on a recursive resolver. Most end points on the Internet as of this writing use stub resolvers.

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 [RFC1035] and current NAT terminology [RFC4787]. Some parts of this document assume familiarity with the terminology of the DNS security extensions outlined in [RFC4035].
5. DNS64 Normative Specification

A 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.

5.1. Resolving AAAA queries and the answer section

When the DNS64 recursor 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 recursive query or, in the case of an authoritative server, by examining its own results.

If this query results in one or more AAAA records in the answer section or in an error condition, this result is returned to the requesting client as per normal DNS semantics (except in the case where the AAAA falls in the ::ffff/96 network; see below for treatment of that network.) In this case, DNS64 SHOULD NOT include synthetic AAAA RRs in the response (see Section 6.2 for an analysis of the motivations and the implications of not complying with this recommendation). By default DNS64 implementations MUST NOT synthesize AAAA RRs when real AAAA RRs exist.

If the query results in no error but an empty answer section in the response, the DNS64 resolver attempts to retrieve A records for the name in question. If this new A RR query results in an empty result or in an error, this result is returned to the client. If the query results in one or more A RRs, the DNS64 synthesizes AAAA RRs based on the A RRs. The DNS64 resolver then returns the synthesized AAAA records in the answer section to the client, removing the A records that form the basis of the synthesis.

As an exception to the general rule about always returning the AAAA records if they are returned in the answer, AAAA records with addresses in the ::ffff/96 network are treated just like the case where there is no error and an empty answer section. This is because a real IPv6-only node will not be any more able to reach the addresses in ::ffff/96 than it is able to reach an IPv4 address without assistance. An implementation MAY use the address in ::ffff/96 as the basis of synthesis without querying for an A record, by using the last 32 bits of the address provided in the AAAA record.

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 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. Handling the additional section

Whenever the DNS64 is answering a query for any RRTYPE other than AAAA, it SHOULD examine the additional section and perform synthesis there if necessary, too. Because A records may appear in the additional section, and because such records could result in a failure to resolve the host in question, a DNS64 server SHOULD synthesize AAAA records for every A record in the additional section if there is no AAAA record already in the additional section for that A record’s RNAME. In this case, the DNS64 MUST add the AAAA record to the additional section without removing the A record. The DNS64 SHOULD sort the additional section with AAAA records first. Only then should the DNS64 truncate the response to the DNS query, if needed. Because of the additional message size that might result from adding the synthesized AAAA records to the response, DNS64 nodes MUST support EDNS0 with buffer sizes adequate to support the additional message sizes (ideally, 4096). If a DNS64 node truncates the additional section, it MUST set the TC bit on the response it provides.

A response to a AAAA query MUST NOT have its additional section manipulated, even if that could result in a resolution failure. This stricture is in place on the principle that an IPv6-only host should prefer native connectivity, and it is therefore important not to adjust AAAA responses in any way.

It is nevertheless worth noting that the above approach may cause IPv6 connections to flow via synthesized addresses even if there is a native IPv6 connection actually available on the target host. This is because a DNS64 could get an answer from a cache where the AAAA records that would otherwise have appeared in the additional section have already been removed. There is no way to remedy the data available from an upstream cache, however, and this approach at least allows connections to succeed. At the same time, one should be aware that caches inside the DNS64 boundary could cause AAAA records in the additional section to be truncated, which could cause resolution failures. Operating a cache behind a DNS64 name server is NOT RECOMMENDED.

5.3. Performing the synthesis

A synthetic AAAA record is created from an A record as follows:
o The NAME field is set to the NAME field from the A record

o The TYPE field is set to 28 (AAAA)

o The CLASS field is set to 1 (IN)

o 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).

o The RDLENGTH field is set to 16

o The RDATA field is set to the IPv6 representation of the IPv4 address from the RDATA field of the A record, see section Section 5.3.1.

### 5.3.1. Generation of the IPv6 representations of IPv4 addresses

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

The same algorithm to create an IPv6 address from an IPv4 address MUST be used by:

- The DNS64 to create the IPv6 address to be returned in the synthetic AAA RR from the IPv4 address contained in original A RR, and,

- The IPv6/IPv4 translator to create the IPv6 address to be included in the destination address field of the outgoing IPv6 packets from the IPv4 address included in the destination address field of the incoming IPv4 packet.

The algorithm MUST be reversible, i.e. it MUST be possible to extract 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 (such as fixed string to be used as a suffix).

If we note n the length of the prefix Pref64::/n, then n MUST the less or equal than 96. If a Pref64::/n is configured through any means in the DNS64 (such as manually configured, or
other automatic means not specified in this document), the default algorithm MUST use this prefix. If no prefix is available, the algorithm MUST use the Well-Known prefix (include here the prefix to be assigned by IANA) defined in [I-D.thaler-behave-translator-addressing]

DNS64 MUST support the following algorithms for generating IPv6 representations of IPv4 addresses defined in [I-D.thaler-behave-translator-addressing]:

- Zero-Pad And Embed, defined in section 3.2.3 of [I-D.thaler-behave-translator-addressing]
- Compensation-Pad And Embed, defined in section of 3.2.4 of [I-D.thaler-behave-translator-addressing]
- Embed And Zero-Pad, defined in section of 3.2.5 of [I-D.thaler-behave-translator-addressing]
- Preconfigured Mapping Table, defined in section of 3.2.6 of [I-D.thaler-behave-translator-addressing]

The default algorithm used by DNS64 must be Embed and Zero-Pad. While the normative description of the algorithms is provided in [I-D.thaler-behave-translator-addressing], an sample description of the algorithm and its application to different scenarios is provided in Section 6.1 for illustration purposes.

5.4. Handling other RRs

5.4.1. PTR queries

If a DNS64 nameserver 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 the locally-configured Pref64::/n. If the address prefix matches any Pref64::/n used in the site, either a LIR prefix or a well-known prefix used for NAT64 as defined in [I-D.thaler-behave-translator-addressing], then the DNS64 server SHOULD 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 those queries should never be sent to the global DNS.
If the address prefix does not match any of the 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.

If the DNS64 is in recursive resolver mode, then it SHOULD also serve the zones specified in [I-D.ietf-dnsop-default-local-zones], rather than forwarding those queries elsewhere to be handled.

All other RRs MUST be returned unchanged, with the exception of the handling of the additional section as outlined in Section 5.2.

5.5. DNSSEC processing: DNS64 in recursive server mode

We consider the case where the recursive server 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.

2. If CD is not set and DO is set, then vDNS64 SHOULD perform validation. If the answer validates, 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 RRsSets in the Answer and Authority sections to be authentic. In this case, the name server has reason to believe the RRsSets 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. Therefore, 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 hand the data back 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 simply won’t 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. Appendixes

6.1. Deployment scenarios and examples

In this section, we first provide a description of the default address transformation algorithm and then we walk through some sample scenarios that are expected to be common deployment cases. It should be noted that 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.thaler-behave-translator-addressing].

There are two main different setups where DNS64 is expected to be used (other setups are possible as well, but these two are the main ones identified at the time of this writing).

One possible setup that is expected to be common is the case of an end site or an ISP that is providing IPv6-only connectivity or connectivity to IPv6-only hosts that wants to allow the communication from these IPv6-only connected hosts to the IPv4 Internet. This case is called An-IPv6-network-to-IPv4-Internet. In this case, the IPv6/IPv4 Translator is used to connect the end site or the ISP to the IPv4 Internet and the DNS64 function is provided by the end site or the ISP.

The other possible setup that is expected is an IPv4 site that wants that its IPv4 servers to be reachable from the IPv6 Internet. This case is called IPv6-Internet-to-an-IPv4-network. It should be noted that the IPv4 addresses used in the IPv4 site can be either public or private. In this case, the IPv6/IPv4 Translator is used to connect the IPv4 end site to the IPv6 Internet and the DNS64 function is provided by the end site itself.
In this section we illustrate how the DNS64 behaves in the different scenarios that are expected to be common. We consider then 3 possible scenarios, namely:

1. An-IPv6-network-to-IPv4-Internet setup with DNS64 in DNS server mode
2. An-IPv6-network-to-IPv4-Internet setup with DNS64 in stub-resolver mode
3. IPv6-Internet-to-an-IPv4-network setup with DNS64 in DNS server mode

The notation used is the following: upper case letters are IPv4 addresses; upper case letters with a prime(’) are IPv6 addresses; lower case letters are ports; prefixes are indicated by "P::X", which is an IPv6 address built from an IPv4 address X by adding the prefix P, mappings are indicated as "(X,x) <--> (Y’,y)".

6.1.1 Embed and Zero-Pad algorithm description

In this section we describe the default algorithm for the generation of IPv6 address from IPv4 address to be implemented in the DNS64.

The only parameter required by the default algorithm is an IPv6 prefix. This prefix is used to map IPv4 addresses into IPv6 addresses, and is denoted Pref64. If we note n the length of the prefix Pref64, then n must the less or equal than 96. If an Pref64 is configured through any means in the DNS64 (such as manually configured, or other automatic mean not specified in this document), the default algorithm must use this prefix. If no prefix is available the algorithm must use the Well-Know prefix (include here the prefix to be assigned by IANA) defined in

[I-D.thaler-behave-translator-addressing]

The input for the algorithm are:

- The IPv4 address: X
- The IPv6 prefix: Pref64::/n

The IPv6 address is generated by concatenating the prefix Pref64::/n, the IPv4 address X and optionally (in case n is strictly smaller than 96) an all-zero suffix. So, the resulting IPv6 address would be Pref64:X::

Reverse algorithm
We next describe the reverse algorithm of the algorithm described in the previous section. This algorithm allows to generate an IPv4 address from an IPv6 address. This reverse algorithm is NOT implemented by the DNS64 but it is implemented in the IPv6/IPv4 translator that is serving the same domain the DNS64.

The only parameter required by the default algorithm is an IPv6 prefix. This prefix is the one originally used to map IPv4 addresses into IPv6 addresses, and is denoted Pref64.

The input for the algorithm are:

The IPv6 address: X'

The IPv6 prefix: Pref64::/n

First, the algorithm checks that the first n bits of the IPv6 address X' match with the prefix Pref64::/n i.e. verifies that Pref64::/n = X'/n.

If this is not the case, the algorithm ends and no IPv4 address is generated.

If the verification is successful, then the bits between the n+1 and the n+32 of the IPv6 address X' are extracted to form the IPv4 address.

6.1.2. 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:
The figure shows an IPv6 node H1 which has an IPv6 address Y’ and an IPv4 node H2 with IPv4 address X.

A IPv6/IPv4 Translator connects the IPv6 network to the IPv4 Internet. This IPv6/IPv4 Translator has a prefix (called Pref64::/n) an IPv4 address T assigned to its IPv4 interface.

The other element involved is the local name server. The name server is a dual-stack node, so that H1 can contact it via IPv6, while it can contact IPv4-only name servers via IPv4.

The local name server needs to know the prefix assigned to the local IPv6/IPv4 Translator (Pref64::/n). For the purpose of this example, we assume it learns this through manual configuration.

For this example, assume the typical DNS situation where IPv6 hosts have only stub resolvers, and always query a name server 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 FQDN(H2). H1 does this by sending a DNS query for an 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 queries for an A record for H2 and gets back an A record containing the IPv4 address X. The name server then synthesizes an AAAA record. The IPv6 address in the AAAA record contains the prefix assigned to the IPv6/IPv4 Translator in the upper n bits then the IPv4 address X and then an all-zero padding i.e. the resulting IPv6 address is Pref64::X::

4. H1 receives the synthetic AAAA record and sends a packet towards H2. The packet is sent from a source transport address of (Y’,y)
to a destination transport address of (Pref64::X::,X), where y and x are ports chosen by H2.

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.

6.1.3. An-IPv6-network-to-IPv4-Internet setup with DNS64 in stub-resolver mode

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

```
+---------------------------------------+         +-----------+
|IPv6 site                              |         |IP addr: T |
| +----------+ | Name | +-------+  T | IPv4 |
| | H1 with DNS64 | Server | 64Trans|-------| Internet |
| +----------+ +-------+ +-------+ +-------+ +-----------+
| | IP addr: Y' |         | IP addr: X |
| +-------------------------+       +----+
| +----------------------------+       | H2 |
```

The figure shows an IPv6 node H1 which has an IPv6 address Y' and an IPv4 node H2 with IPv4 address X. Node H1 is implementing the DNS64 function.

A IPv6/IPv4 Translator connects the IPv6 network to the IPv4 Internet. This IPv6/IPv4 Translator has a prefix (called Pref64::/n) and an IPv4 address T assigned to its IPv4 interface.

H1 needs to know the prefix assigned to the local IPv6/IPv4 Translator (Pref64::/n). For the purpose of this example, we assume it learns this through manual configuration.

Also shown is a name server. For the purpose of this example, we assume that the name server is a dual-stack node, so that H1 can contact it via IPv6, while it can contact IPv4-only name servers via IPv4.

For this example, assume the typical situation where IPv6 hosts have only stub resolvers and always query a name server that provides recursive lookups (henceforth called "the recursive name server"). The recursive name server does not perform the DNS64 function.

The steps by which H1 establishes communication with H2 are:

---

Several lines of text are truncated due to the length of the representation.
1. H1 does a DNS lookup for FQDN(H2). 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 X. 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 n bits, then the IPv4 address X and then an all-zero padding i.e. the resulting IPv6 address is Pref64:X::.

4. H1 sends a packet towards H2. The packet is sent from a source transport address of (Y’,y) to a destination transport address of (Pref64:X::,x), where y and x are ports chosen by H2.

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.

6.1.4. 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 site that initiates a communication to a IPv4 node located in the IPv4 site.

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 servers. Such an IPv6 address would be constructed as the Pref64::/n concatenated with the IPv4 address of the IPv4 server and an all-zero padding. Note that the IPv4 address can be a public or a private address; the latter does not present any additional difficulty, since the LIR prefix must be used a Pref64 (in this scenario the usage of the WK prefix is not supported). Once these IPv6 addresses have been assigned to represent the IPv4 servers 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. Such a configuration is easier to troubleshoot in the event of problems, because it always provides the same answer to every 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
servers, there are two options: One option is to modify the 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 to modify the authoritative server 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 and so on when the DNS responses are not being generated dynamically. For completeness, the DNS64 behavior that we describe in this section covers the case of synthesizing the AAAA RR when the DNS query arrives. Nevertheless, such a configuration is NOT RECOMMENDED. Troubleshooting configurations that change the data depending on the query they receive is notoriously hard, and the IPv4/IPv6 translation scenario is complicated enough without adding additional opportunities for possible malfunction.

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

```
+-----------+          +----------------------------------------+
|           |          |   IPv4 site            +-------------+ |
|   IPv6    |      +-------+      +----+        | Name server | |
| Internet  |------|64Trans|      | H2 |        | with DNS64  | |
+-----------+      +-------+      +----+        +-------------+ |
| IP addr: Y'         |  |         |IP addr: X     |           |
+----+                 | -----------------------------------    |
| H1 |                 +----------------------------------------+
+----+
```

The figure shows an IPv6 node H1 which has an IPv6 address Y' and an IPv4 node H2 with IPv4 address X.

A IPv6/IPv4 Translator connects the IPv4 network to the IPv6 Internet. This IPv6/IPv4 Translator has a prefix (called Pref64::/n).

Also shown is the authoritative name server for the local domain with DNS64 functionality. For the purpose of this example, we assume that the name server is a dual-stack node, so that H1 or a recursive resolver acting on the request of H1 can contact it via IPv6, while it can be contacted by IPv4-only nodes to receive dynamic DNS updates via IPv4.
The local name server needs to know the prefix assigned to the local IPv6/IPv4 Translator (Pref64::/n). For the purpose of this example, we assume it learns this through manual configuration.

The steps by which H1 establishes communication with H2 are:

1. H1 does a DNS lookup for FQDN(H2). H1 does this by sending a DNS query for an 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 FQDN(H2) 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 an AAAA record from the A RR and the relevant Pref64::/n. The IPv6 address in the AAAA record contains the prefix assigned to the IPv6/IPv4 Translator in the first n bits and the IPv4 address X and then an all-zero padding.

4. H1 receives the synthetic AAAA record and sends a packet towards H2. The packet is sent from a source transport address of (Y’,y) to a destination transport address of (Pref64:X::,x), where y and x are ports chosen by H2.

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.

6.2 Motivations and Implications of synthesizing AAAA RR when real AAAA RR exists

The motivation for synthesizing AAAA RR when a real AAAA RR exists 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 also 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 as a result.

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 recursor.
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 RR even when real AAAA RR exist.

The implication of including synthetic AAAA RR when real AAAA RR 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 which is the preferred destination address to use. So, if the DNS64 recursor returns both the synthetic AAAA RR and the real AAAA RR, then if the DNS64 is operated by the same domain as the initiating host, and a global unicast prefix (called the LIR prefix as defined in [I-D.thaler-behave-translator-addressing]) is used, then the synthetic AAAA RR is likely to be preferred.

This means that without further configuration:

In the case of an IPv6 network to the IPv4 internet, the host will prefer translated connectivity if LIR prefix is used. If the Well-Known (WK) prefix defined in [I-D.thaler-behave-translator-addressing] is used, it will probably prefer native connectivity.

In the case of the IPv6 Internet to an IPv4 network, it is possible to bias the selection towards the real AAAA RR if the DNS64 recursor returns the real AAAA first in the DNS reply, when the LIR prefix is used (the WK prefix usage is not recommended in this case).

In the case of the IPv6 to IPv4 in the same network, for local destinations (i.e., target hosts inside the local site), it is likely that the LIR prefix 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 a WK prefix is used, the longest prefix match rule will select native connectivity.

So this option introduces problems in the following cases:

An IPv6 network to the IPv4 internet with the LIR prefix

IPv6 to IPv4 in the same network when reaching external destinations and the LIR prefix is used.
In any case, the problem can be solved by properly configuring the RFC3484 [RFC3484] policy table, but this requires effort on the part of the site operator.

7. Security Considerations

See the discussion on the usage of DNSSEC and DNS64 described in the document.

8. IANA Considerations

This draft has no actions for IANA.

9. Contributors

Dave Thaler
Microsoft
dthaler@windows.microsoft.com

10. Acknowledgements

This draft contains the result of discussions involving many people, including: Dan Wing, Jari Arkko, Mark Townsley, Fred Baker, Xing Li, Hiroshi Miyata, Brian Carpenter, Ed Jankiewicz, Magnus Westerlund, Ed Lewis, Rob Austein, Matthijs Mekking, Peter Koch, Mark Andrews, Simon Perrault.

Marcelo Bagnulo and Iljitsch van Beijnum are partly funded by Trilogy, a research project supported by the European Commission under its Seventh Framework Program.

11. References

11.1. Normative References


11.2. Informative References

[I-D.bagnulo-behave-nat64]


[RFC1858]  Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858,
October 1995.


draft-wing-behave-learn-prefix-02 (work in progress),
May 2009.

Miyata, H. and M. Bagnulo, "PREFIX64 Comparison",
draft-miyata-behave-prefix64-02 (work in progress),
March 2009.

Andrews, M., "Locally-served DNS Zones",
draft-ietf-dnsop-default-local-zones-08 (work in progress),
February 2009.

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