BEHAVE WG
Internet-Draft
Intended status: Standards Track
Expires: September 8, 2009

M. Bagnulo
UC3M
P. Matthews
Alcatel-Lucent
I. van Beijnum
IMDEA Networks
March 7, 2009

NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers
draft-bagnulo-behave-nat64-03

Status of this Memo

Copyright Notice

Abstract

NAT64 is a mechanism for translating IPv6 packets to IPv4 packets and vice-versa. DNS64 is a mechanism for synthesizing AAAA records from A records. These two mechanisms together 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. They also enable peer-to-peer communication between an IPv4 and an IPv6 node, where the communication can be initiated by either end using existing, NAT-traversing, peer-to-peer communication techniques. This document specifies NAT64, and gives suggestions on how they should be deployed.

Table of Contents

1. Introduction

This document specifies NAT64, a mechanism for IPv6-IPv4 transition and co-existence. Together with DNS64 [I-D.bagnulo-behave-dns64], these two mechanisms allow a IPv6-only client to initiate communications to an IPv4-only server, and also allow peer-to-peer communication between IPv6-only and IPv4-only hosts.

NAT64 is a mechanism for translating IPv6 packets to IPv4 packets. The translation is done by translating the packet headers according to SIIT [RFC2765], translating the IPv4 server address by adding or removing a /96 prefix, and translating the IPv6 client address by installing mappings in the normal NAT manner.

DNS64 is a mechanism for synthesizing AAAA resource records (RR) from A RR. The synthesis is done by adding a /96 prefix to the IPv4 address to create an IPv6 address, where the /96 prefix is assigned to a NAT64 device.

Together, these two mechanisms allow a IPv6-only client to initiate communications to an IPv4-only 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. The NAT64 and DNS64 mechanisms are easily deployable, since they require no changes to either the IPv6 client nor the IPv4 server. For basic functionality, the approach only requires the deployment of NAT64-enabled devices connecting an IPv6-only network to the IPv4-only Internet, along with the deployment of a few DNS64-enabled name servers in the IPv6-only network. However, some advanced features require software updates to the IPv6-only hosts.

The NAT64 and DNS64 mechanisms are related to the NAT-PT mechanism defined in [RFC2766], but significant differences exist. First, NAT64 does not define the NATPT mechanisms used to support IPv6 only servers to be contacted by IPv4 only clients, but only defines the mechanisms for IPv6 clients to contact IPv4 servers and its potential reuse to support peer to peer communications through standard NAT traversal techniques. Second, NAT64 includes a set of features that overcomes many of the reasons the original NAT-PT specification was moved to historic status [RFC4966].

1.1. Features of NAT64

The features of NAT64 and DNS64 are:

- It enables IPv6-only nodes to initiate a client-server connection with an IPv4-only server, without needing any changes on either IPv4 or IPv6 nodes. This works for the same class of applications that work through IPv4-to-IPv4 NATs.

- It supports peer-to-peer communication between IPv4 and IPv6 nodes, including the ability for IPv4 nodes to initiate communcation with IPv6 nodes using peer-to-peer techniques (i.e., using a rendezvous server and ICE). To this end, NAT64 is compliant with the recommendations for how NATs should handle UDP [RFC4787], TCP [I-D.ietf-behave-tcp], and ICMP [I-D.ietf-behave-nat-icmp].

- Compatible with ICE.

- Supports additional features with some changes on nodes. These features include:

  * Support for DNSSec

  * Some forms of IPSec support

1.2. Overview

This section provides a non-normative introduction to the mechanisms of NAT64.

NAT64 mechanism is implemented in an NAT64 box which has two interfaces, an IPv4 interface connected to the the IPv4 network, and an IPv6 interface connected to the IPv6 network. Packets generated in the IPv6 network for a receiver located in the IPv4 network will be routed within the IPv6 network towards the NAT64 box. The NAT64 box will translate them and forward them as IPv4 packets through the IPv4 network to the IPv4 receiver. The reverse takes place for packets generated in the IPv4 network for an IPv6 receiver. NAT64, however, is not symmetric. In order to be able to perform IPv6 - IPv4 translation NAT64 requires state, binding an IPv6 address and port (hereafter called an IPv6 transport address) to an IPv4 address and port (hereafter called an IPv4 transport address).

Such binding state is created when the first packet flowing from the IPv6 network to the IPv4 network is translated. After the binding state has been created, packets flowing in either direction on that particular flow are translated. The result is that NAT64 only supports communications initiated by the IPv6-only node towards an IPv4-only node. Some additional mechanisms, like ICE, can be used in combination with NAT64 to provide support for communications initiated by the IPv4-only node to the IPv6-only node. The

specification of such mechanisms, however, is out of the scope of this document.

1.2.1. NAT64 solution elements

In this section we describe the different elements involved in the NAT64 approach.

The main component of the proposed solution is the translator itself. The translator has essentially two main parts, the address translation mechanism and the protocol translation mechanism.

Protocol translation from IPv4 packet header to IPv6 packet header and vice-versa is performed according to SIIT [RFC2765].

Address translation maps IPv6 transport addresses to IPv4 transport addresses and vice-versa. In order to create these mappings the NAT64 box has two pools of addresses i.e. an IPv6 address pool (to represent IPv4 addresses in the IPv6 network) and an IPv4 address pool (to represent IPv6 addresses in the IPv4 network). Since there is enough IPv6 address space, it is possible to map every IPv4 address into a different IPv6 address.

NAT64 creates the required mappings by using as the IPv6 address pool a /96 IPv6 prefix (hereafter called Pref64::/96). This allows each IPv4 address to be mapped into a different IPv6 address by simply concatenating the /96 prefix assigned as the IPv6 address pool of the NAT64, with the IPv4 address being mapped (i.e. an IPv4 address X is mapped into the IPv6 address Pref64:X). The NAT64 prefix Pref64::/96 is assigned by the administrator of the NAT64 box from the global unicast IPv6 address block assigned to the site. It should be noted that the the prefix used as the IPv6 address pool is assigned to a specific NAT64 box and if there are multiple NAT64 boxes, each box is allocated a different prefix. Assigning the same prefix to multiple boxes may lead to communication failures due to internal routing fluctuations.

The IPv4 address pool, however, is a set of IPv4 addresses, normally a small prefix assigned by the local administrator to the NAT64's external (IPv4) interface. Since IPv4 address space is a scarce resource, the IPv4 address pool is small and typically not sufficient to establish permanent one-to-one mappings with IPv6 addresses. So, mappings using the IPv4 address pool will be created and released dynamically. Moreover, because of the IPv4 address scarcity, the usual practice for NAT64 is likely to be the mapping of IPv6 transport addresses into IPv4 transport addresses, instead of IPv6 addresses into IPv4 addresses directly, which enable a higher utilization of the limited IPv4 address pool.

Because of the dynamic nature of the IPv6 to IPv4 address mapping and the static nature of the IPv4 to IPv6 address mapping, it is easy to understand that it is far simpler to allow communication initiated from the IPv6 side toward an IPv4 node, which address is permanently mapped into an IPv6 address, than communications initiated from IPv4-only nodes to an IPv6 node in which case IPv4 address needs to be associated with it dynamically. For this reason NAT64 supports only communications initiated from the IPv6 side.

An IPv6 initiator can know or derive in advance the IPv6 address representing the IPv4 target and send packets to that address. The packets are intercepted by the NAT64 device, which associates an IPv4 transport address of its IPv4 pool to the IPv6 transport address of the initiator, creating binding state, so that reply packets can be translated and forwarded back to the initiator. The binding state is kept while packets are flowing. Once the flow stops, and based on a timer, the IPv4 transport address is returned to the IPv4 address pool so that it can be reused for other communications.

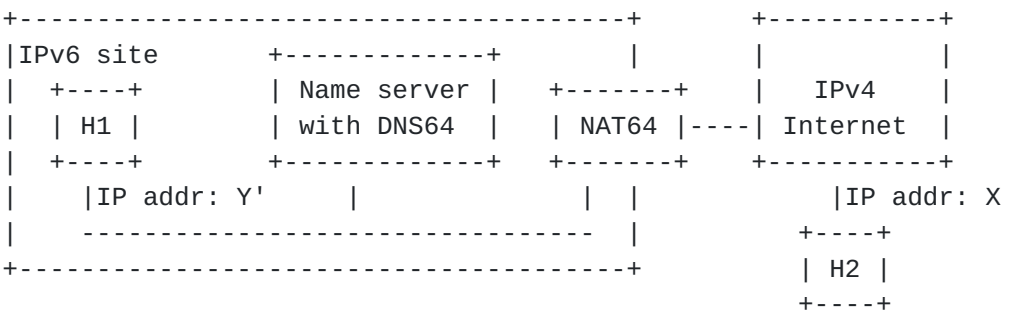
To allow an IPv6 initiator to do the standard DNS lookup to learn the address of the responder, DNS64 [I-D.bagnulo-behave-dns64] is used to synthesize an AAAA record from the A record (containing the real IPv4 address of the responder). DNS64 receives the DNS queries generated by the IPv6 initiator. 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 the A record. If an A record is discovered, DNS64 creates a synthetic AAAA RR by adding the Pref64::/96 of a NAT64 to the responder's IPv4 address (i.e. if the IPv4 node has IPv4 address X, then the synthetic AAAA RR will contain the IPv6 address formed as Pref64:X). The synthetic AAAA RR is passed back to the IPv6 initiator, which will initiate an IPv6 communication with the IPv6 address associated to the IPv4 receiver. The packet will be routed to the NAT64 device, which will create the IPv6 to IPv4 address mapping as described before.

1.2.2. Walkthough

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

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 a IPv6 address built from an IPv4 address X by adding the prefix P, mappings are indicated as "(X,x) <--> (Y',y)".

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



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

A NAT64 connects the IPv6 network to the IPv4 Internet. This NAT64 has a /96 prefix (called Pref64::/96) associated to its IPv6 interface and an IPv4 address T assigned to its IPv4 interface.

Also shown is a local name server with DNS64 functionality. 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.

The local name server needs to know the /96 prefix assigned to the local NAT64 (Pref64::/96). 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 the local name server does the recursive lookups.

The steps by which H1 establishes communication with H2 are:

1. H1 performs a DNS query for FQDN(H2) and receives the synthetic AAAA RR from the local name server that implements the DNS64 functionality. The AAAA record contains an IPv6 address formed by the PRefix64::/96 associated to the NAT64 box and the IPv4 address of H2 in the lower 32 bits.

2. H1 sends a packet to 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 H1.

3. The packet is routed to the IPv6 interface of the NAT64 (since Pref64::/96 has been associated to this interface).

4. The NAT64 receives the packet and performs the following actions:

* The NAT64 selects an unused port t on its IPv4 address T and creates the mapping entry (Y',y) <--> (T,t)

* The NAT64 translates the IPv6 header into an IPv4 header using SIIT.

* The NAT64 includes (T,t) as source transport address in the packet and (X,x) as destination transport address in the packet. Note that X is extracted directly from the lower 32 bits of the destination IPv6 address of the received IPv6 packet that is being translated.

The NAT64 sends the translated packet out its IPv4 interface and the packet arrives at H2.

5. H2 node responds by sending a packet with destination transport address (T,t) and source transport address (X,x).

6. The packet is routed to the NAT64 box, which will look for an existing mapping containing (T,t). Since the mapping (Y',y) <--> (T,t) exists, the NAT64 performs the following operations:

   * The NAT64 translates the IPv4 header into an IPv6 header using SIIT.

   * The NAT64 includes (Y',y) as destination transport address in the packet and (Pref64:X,x) as source transport address in the packet. Note that X is extracted directly from the source IPv4 address of the received IPv4 packet that is being translated.

   The translated packet is sent out the IPv6 interface to H1.

The packet exchange between H1 and H2 continues and packets are translated in the different directions as previously described.

It is important to note that the translation still works if the IPv6 initiator H1 learns the IPv4 address through some scheme other than a DNS look-up. This is because the DNS64 processing does NOT result in any state installed in the NAT64 box and because the mapping of the IPv4 address into an IPv6 address is the result of concatenating the prefix defined within the site for this purpose (called Pref64::/96 in this document) to the original IPv4 address.

1.2.3. Filtering

A NAT64 box may do filtering, which means that it only allows a packet in through an interface if the appropriate permission exists.

A NAT64 may do no filtering, or it may filter on its IPv4 interface. Filtering on the IPv6 interface is not supported, as mappings are only created by packets traveling in the IPv6 --> IPv4 direction.

If a NAT64 performs address-dependent filtering according to RFC4787 [RFC4787] on its IPv4 interface, then an incoming packet is dropped unless a packet has been recently sent out the interface with a destination IP address equal to the source IP address of the incoming packet.

NAT64 filtering is consistent with the recommendations of RFC 4787 [RFC4787].

2. 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:

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

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.

NAT64: A device that translates IPv6 packets to IPv4 packets and vice-versa, with the provision that the communication must be initiated from the IPv6 side. The translation involves not only the IP header, but also the transport header (TCP or UDP).

Session: A TCP or UDP session. In other words, the bi-directional flow of packets between two ports on two different hosts. In NAT64, typically one host is an IPv4 host, and the other one is an IPv6 host.

5-Tuple: The tuple (source IP address, source port, destination IP address, destination port, transport protocol). A 5-tuple uniquely identifies a session. When a session flows through a NAT64, each session has two different 5-tuples: one with IPv4 addresses and one with IPv6 addresses.

Session table: A table of sessions kept by a NAT64. Each NAT64 has two session tables, one for TCP and one for UDP.

Transport Address: The combination of an IPv6 or IPv4 address and a port. Typically written as (IP address, port); e.g. (192.0.2.15, 8001).

Mapping: A mapping between an IPv6 transport address and a IPv4 transport address. Used to translate the addresses and ports of packets flowing between the IPv6 host and the IPv4 host. In NAT64, the IPv4 transport address is always a transport address assigned to the NAT64 itself, while the IPv6 transport address belongs to some IPv6 host.

BIB: Binding Information Base. A table of mappings kept by a NAT64. Each NAT64 has two BIBs, one for TCP and one for UDP.

Endpoint-Independent Mapping: In NAT64, using the same mapping for all sessions between an IPv6 that have the same IPv6 transport address endpoint. Endpoint-independent mapping is important for peer-to-peer communication. See [RFC4787] for the definition of the different types of mappings in IPv4-to-IPv4 NATs.

Hairpinning: Having a packet do a "U-turn" inside a NAT and come back out the same interface as it arrived on. Hairpinning support is important for peer-to-peer applications, as there are cases when two different hosts on the same side of a NAT can only communicate using sessions that hairpin though the NAT.

For a detailed understanding of this document, the reader should also be familiar with DNS terminology [RFC1035] and current NAT terminology [RFC4787].

3. NAT64 Normative Specification

A NAT64 is a device with one IPv6 interface and one IPv4 interface. The IPv6 interface MUST have a unicast /96 IPv6 prefix assigned to it, denoted Pref64::/96. The IPv4 interface MUST have one or more unicast IPv4 addresses assigned to it.

A NAT64 uses the following dynamic data structures:

- UDP BIB
- UDP Session Table

- TCP BIB

- TCP Session Table

A NAT64 has two Binding Information Bases: one for TCP and one for UDP. Each BIB entry specifies a mapping between an IPv6 transport address and an IPv4 transport address:

(X',x) <--> (T,t)

where X' is some IPv6 address, T is an IPv4 address, and x and t are ports. T will always be one of the IPv4 addresses assigned to the IPv4 interface of the NAT64. A given IPv6 or IPv4 transport address can appear in at most one entry in a BIB: for example, (2001:db8::17, 4) can appear in at most one TCP and at most one UDP BIB entry. TCP and UDP have separate BIBs because the port number space for TCP and UDP are distinct.

A NAT64 also has two session tables: one for TCP sessions and one for UDP sessions. Each entry keeps information on the state of the corresponding session: see Section 3.2. The NAT64 uses the session state information to determine when the session is completed, and also uses session information for ingress filtering. A session can be uniquely identified by either an incoming 5-tuple or an outgoing 5-tuple.

For each session, there is a corresponding BIB entry, uniquely specified by either the source IPv6 transport address (in the IPv6 --> IPv4 direction) or the destination IPv4 transport address (in the IPv4 --> IPv6 direction). However, a single BIB entry can have multiple corresponding sessions. When the last corresponding session is deleted, the BIB entry is deleted.

The processing of an incoming IP packet takes the following steps:

1. Determining the incoming 5-tuple

2. Filtering and updating session information

3. Computing the outgoing 5-tuple

4. Translating the packet

5. Handling hairpinning

The details of these steps are specified in the following subsections.

This breakdown of the NAT64 behavior into processing steps is done for ease of presentation. A NAT64 MAY perform the steps in a different order, or MAY perform different steps, as long as the externally visible outcome in the same.

TBD: Add support for ICMP Query packets. (ICMP Error packets are handled).

3.1. Determining the Incoming 5-tuple

This step associates a incoming 5-tuple (source IP address, source port, destination IP address, destination port, transport protocol) with every incoming IP packet for use in subsequent steps.

If the incoming IP packet contains a complete (un-fragmented) UDP or TCP protocol packet, then the 5-tuple is computed by extracting the appropriate fields from the packet.

If the incoming IP packet contains a complete (un-fragmented) ICMP message, then the 5-tuple is computed by extracting the appropriate fields from the IP packet embedded inside the ICMP message. However, the role of source and destination is swapped when doing this: the embedded source IP address becomes the destination IP address in the 5-tuple, the embedded source port becomes the destination port in the 5-tuple, etc. If it is not possible to determine the 5-tuple (perhaps because not enough of the embedded packet is reproduced inside the ICMP message), then the incoming IP packet is silently discarded.

> NOTE: The transport protocol is always one of TCP or UDP, even if the IP packet contains an ICMP message.

If the incoming IP packet contains a fragment, then more processing may be needed. This specification leaves open the exact details of how a NAT64 handles incoming IP packets containing fragments, and simply requires that a NAT64 handle fragments arriving out-of-order. A NAT64 MAY elect to queue the fragments as they arrive and translate all fragments at the same time. Alternatively, a NAT64 MAY translate the fragments as they arrive, by storing information that allows it to compute the 5-tuple for fragments other than the first. In the latter case, the NAT64 will still need to handle the situation where subsequent fragments arrive before the first.

Implementors of NAT64 should be aware that there are a number of well-known attacks against IP fragmentation; see [RFC1858] and [RFC3128].

Assuming it otherwise has sufficient resources, a NAT64 MUST allow

the fragments to arrive over a time interval of at least 10 seconds. A NAT64 MAY require that the UDP, TCP, or ICMP header be completely contained within the first fragment.

3.2. Filtering and Updating Session Information

This step updates the per-session information stored in the appropriate session table. This affects the lifetime of the session, which in turn affects the lifetime of the corresponding BIB entry. This step may also filter incoming packets, if desired.

The details of this step depend on the transport protocol (UDP or TCP).

3.2.1. UDP Session Handling

The state information stored for a UDP session is a timer that tracks the remaining lifetime of the UDP session. The NAT64 decrements this timer at regular intervals. When the timer expires, the UDP session is deleted.

The incoming packet is processed as follows:

1. If the packet arrived on the IPv4 interface and the NAT64 filters on its IPv4 interface, then the NAT64 checks to see if the incoming packet is allowed according to the address-dependent filtering rule. To do this, it searches for a session table entry with a destination IPv4 address equal to the source IPv4 address in the incoming 5-tuple. If such an entry is found (there may be more than one), packet processing continues. Otherwise, the packet is discarded. If the packet is discarded, then an ICMP message SHOULD be sent to the original sender of the packet, unless the discarded packet is itself an ICMP message. The ICMP message, if sent, has a type of 3 (Destination Unreachable) and a code of 13 (Communication Administratively Prohibited).

2. The NAT64 searches for the session table entry corresponding to the incoming 5-tuple. If no such entry if found, a new entry is created.

3. The NAT64 sets or resets the timer in the session table entry to maximum session lifetime. By default, the maximum session lifetime is 5 minutes, but for specific destination ports in the Well-Known port range (0..1023), the NAT64 MAY use a smaller maximum lifetime.

3.2.2. TCP Session Handling

TBD: Describe the state machine required to track the state of the TCP session. This is a simplified version of the state machine used by the endpoints.

3.2.3. Computing the Outgoing 5-Tuple

This step computes the outgoing 5-tuple by translating the addresses and ports in the incoming 5-tuple. The transport protocol in the outgoing 5-tuple is always the same as that in the incoming 5-tuple.

In the text below, a reference to the "the BIB" means either the TCP BIB or the UDP BIB as appropriate, as determined by the transport protocol in the 5-tuple.

> NOTE: Not all addresses are translated using the BIB. BIB entries are used to translate IPv6 source transport addresses to IPv4 source transport addresses, and IPv4 destination transport addresses to IPv6 destination transport addresses. They are NOT used to translate IPv6 destination transport addresses to IPv4 destination transport addresses, nor to translate IPv4 source transport addresses to IPv6 source transport addresses. The latter cases are handled by adding or removing the /96 prefix. This distinction is important; without it, hairpinning doesn't work correctly.

When translating in the IPv6 --> IPv4 direction, let the incoming source and destination transport addresses in the 5-tuple be (S',s) and (D',d) respectively. The outgoing source transport address is computed as follows:

> If the BIB contains a entry (S',s) <--> (T,t), then the outgoing source transport address is (T,t).
>
> Otherwise, create a new BIB entry (S',s) <--> (T,t) as described below. The outgoing source transport address is (T,t).

The outgoing destination address is computed as follows:

> If D' is composed of the NAT64's prefix followed by an IPv4 address D, then the outgoing destination transport address is (D,d).
>
> Otherwise, discard the packet.

When translating in the IPv4 --> IPv6 direction, let the incoming source and destination transport addresses in the 5-tuple be (S,s)

and (D,d) respectively. The outgoing source transport address is computed as follows:

The outgoing source transport address is (Pref64::S,s).

The outgoing destination transport address is computed as follows:

If the BIB contains an entry (X',x) <--> (D,d), then the outgoing destination transport address is (X',x).

Otherwise, discard the packet.

If the rules specify that a new BIB entry is created for a source transport address of (S',s), then the NAT64 allocates an IPv4 transport address for this BIB entry as follows:

If there exists some other BIB entry containing S' as the IPv6 address and mapping it to some IPv4 address T, then use T as the IPv4 address. Otherwise, use any IPv4 address assigned to the IPv4 interface.

If the port s is in the Well-Known port range 0..1023, then allocate a port t from this same range. Otherwise, if the port s is in the range 1024..65535, then allocate a port t from this range. Furthermore, if port s is even, then t must be even, and if port s is odd, then t must be odd.

In all cases, the allocated IPv4 transport address (T,t) MUST NOT be in use in another entry in the same BIB, but MAY be in use in the other BIB.

If it is not possible to allocate an appropriate IPv4 transport address or create a BIB entry for some reason, then the packet is discarded.

TBD: Do we delete the session entry if we cannot create a BIB entry?

If the rules specify that the packet is discarded, then the NAT64 SHOULD send an ICMP reply to the original sender, unless the packet being translated contains an ICMP message. The type should be 3 (Destination Unreachable) and the code should be 0 (Network Unreachable in IPv4, and No Route to Destination in IPv6).

3.2.4. Translating the Packet

This step translates the packet from IPv6 to IPv4 or vica-versa.

The translation of the packet is as specified in section 3 and

section 4 of SIIT [RFC2765], with the following modifications:

- When translating an IP header (sections 3.1 and 4.1), the source and destination IP address fields are set to the source and destination IP addresses from the outgoing 5-tuple.

- When the protocol following the IP header is TCP or UDP, then the source and destination ports are modified to the source and destination ports from the 5-tuple. In addition, the TCP or UDP checksum must also be updated to reflect the translated addresses and ports; note that the TCP and UDP checksum covers the pseudo-header which contains the source and destination IP addresses. An algorithm for efficently updating these checksums is described in [RFC3022].

- When the protocol following the IP header is ICMP (sections 3.4 and 4.4) the source and destination transport addresses in the embedded packet are set to the destination and source transport addresses from the outgoing 5-tuple (note the swap of source and destination).

3.2.5. Handling Hairpinning

This step handles hairpinning if necessary.

If the destination IP address is an address assigned to the NAT64 itself (i.e., is one of the IPv4 addresses assigned to the IPv4 interface, or is covered by the /96 prefix assigned to the IPv6 interface), then the packet is a hairpin packet. The outgoing 5-tuple becomes the incoming 5-tuple, and the packet is treated as if it was received on the outgoing interface. Processing of the packet continues at step 2.

TBD: Is there such a thing as a hairpin loop (likely not naturally, but perhaps through a special-crafted attack packet with a spoofed source address)? If so, need to drop packets that hairpin more than once.

3.3. FTP ALG

TBD: Describe the FTP ALG, a mechanism for translating the embedded IP addresses inside FTP commands, that enables FTP sessions to pass through NAT64.

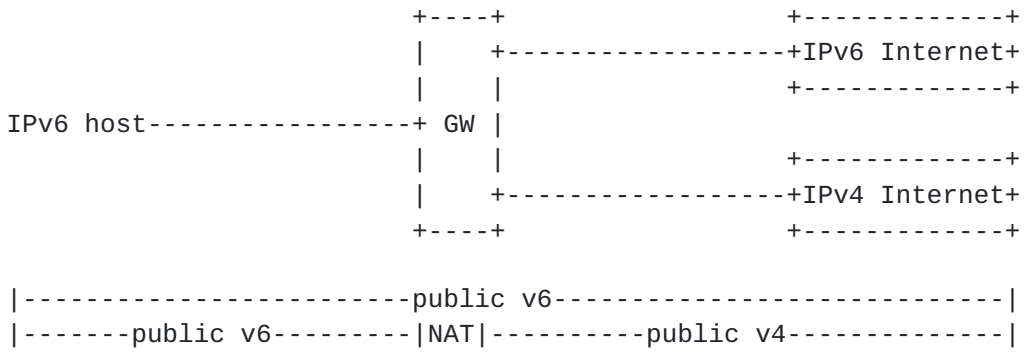
4. Application scenarios

In this section, we describe how to apply NAT64/DNS64 to the suitable

scenarios described in draft-arkko-townsley-coexistence.

4.1. Enterprise IPv6 only network

The Enterprise IPv6 only network basically has IPv6 hosts (those that are currently available) and because of different reasons including operational simplicity, wants to run those hosts in IPv6 only mode, while still providing access to the IPv4 Internet. The scenario is depicted in the picture below.

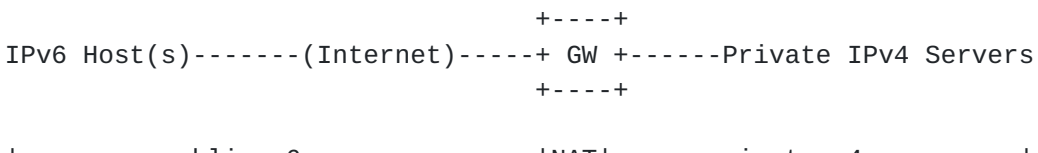


The proposed NAT64/DNS64 is perfectly suitable for this particular scenario. The deployment of the NAT64/DNS64 would be as follows: The NAT64 function should be located in the GW device that connects the IPv6 site to the IPv4 Internet. The DNS64 functionality can be placed in different places. Probably the best trade-off between architectural cleanness deployment simplicity would be to place it in the local recursive DNS server of the enterprise site. The option that is easier to deploy would be to co-locate it with the NAT64 box. The cleanest option would be included in the local resolver of the IPv6 hosts, but this option seems the harder to deploy cause it implies changes to the hosts.

The proposed NAT64/DNS64 approach satisfies the requirements of this scenario, in particular cause it doesn't require any changes to current IPv6 hosts in the site to obtain basic functionality.

4.2. Reaching servers in private IPv4 space

The scenario of servers using IPv4 private addresses and being reached from the IPv6 Internet basically includes the cases that for whatever reason the servers cannot be upgraded to IPv6 and they don't have public VIPv4 addresses and it would be useful to allow IPv6 nodes in the IPv6 Internet to reach those servers. This scenario is depicted in the figure below.



This scenario can again be perfectly served by the NAT64 approach. In this case the NAT64 functionality is placed in the GW device connecting the IPv6 Internet to the server's site. In this case, the DNS64 functionality is not needed. Since the server's site is running the NAT64 and the servers, it can publish in its own DNS server the AAAA RR corresponding to the servers i.e. AAAA RR associating the FQDN of the server and the Pref64:ServerIPv4Addr. In this case, there is no need to synthesize AAAA RR cause the site can configure them in the DNS itself.

Again, this scenario is satisfied by the NAT64 since it supports the required functionality without requiring changes in the IPv4 servers nor in the IPv6 clients.

5. Discussion

5.1. About the Prefix used to map the IPv4 address space into IPv6

In the NAT64 approach, we need to represent the IPv4 addresses in the IPv6 Internet. Since there is enough address space in IPv6, we can easily embed the IPv4 address into an IPv6 address, so that the IPv4 address information can be extracted from the IPv6 address without requiring additional state. One way to that is to use an IPv6 prefix Pref64::/96 and juxtapose the IPv4 address at the end (there are other ways of doing it, but we are not discussing the different formats here). In this document the Pref64::/96 prefix is extracted from the address block assigned to the site running the NAT64 box. However, one could envision the usage of other prefixes for that function. In particular, it would be possible to define a well-known prefix that can be used by the NAT64 devices to map IPv4 (public) addresses into IPv6 addresses, irrespectively of the address space of the site where the NAT64 is located. In this section, we discuss the pro and cons of the different options.

the different options for Pref64::/96 are the following

Local: A locally assigned prefix out of the address block of the site running the NAT64 box

Well-known: A well know prefix that is reserved for this purpose. We have the following different options:

IPv4 mapped prefix

IPv4 compatible prefix

A new prefix assigned by IANA for this purpose

The reasons why using a well-known prefix is attractive are the following: Having a global well-know prefix would allow to identify which addresses are "real" IPv6 addresses with native connectivity and which addresses are IPv6 addresses that represent an IPv4 address. From an architectural perspective, it seems the right thing to do to make this visible since hosts an applications could react accordingly and avoid or prefer such type of connectivity if needed. From the DNS64 perspective, using the well-know prefix would imply that the same synthetic AAAA RR will be created throughout the IPv6 Internet, which would result in consistent view of the RR irrespectively of the location in the topology. From a more practical perspective, having a well-know prefix would allow to completely decouple the DNS64 from the NAT64, since the DNS64 would always use the well-know prefix to create the synthetic AAAA RR and there is no need to configure the same Pref64::/96 both in the DNS64 and the NAT64 that work together.

Among the different options available for the well-know prefix, the option of using a pre-existing prefix such as the IPv4-mapped or IPv4-compatible prefix has the advantage that would potentially allow the default selection of native connectivity over translated connectivity for legacy hosts in communications involving dual-stack hosts. This is because current RFC3484 default policy table include entries for the IPv4-mapped prefix and the IPv4-compatible prefix, implying that native IPv6 prefixes will be preferred over these. However, current implementations do not use the IPv4-mapped prefix on the wire, beating the purpose of support unmodified hosts. The IPv4-compatible prefix is used by hosts on the wire, but has a higher priority than the IPv4-mapped prefix, which implies that current hosts would prefer translated connectivity over native IPv4 connectivity (represented by the IPv4-mapped prefix in the default policy table). So neither of the prefixes that are present in the default policy table would result in the legacy hosts preferring native connectivity over translated connectivity, so it doesn't seem to be a compelling reason to re-use neither the IPv4-mapped not the IPv4-compatible prefix for this. So, we conclude that among the the well know prefix options, the preferred option would be to ask for a new prefix from IANA to be allocated for this.

However, there are several issues when considering using the well-know prefix option, namely:

The well-know prefix is suitable only for mapping IPv4 public addresses into IPv6. IPv4 public addresses can be mapped using the same prefix cause they are globally unique. However, the well-known prefix is not suitable for mapping IPv4 private addresses. This is so because we cannot leverage on the uniqueness of the IPv4 address to achieve uniqueness of the IPv6 address, so we need to use a different IPv6 prefix to disambiguate the different private IPv4 address realms. As we describe above, there is a clear use case for mapping IPv4 private addresses, so there is a pressing need to map IPv4 private addresses. In order to do so we will need to use at least for IPv4 private addresses, IPv6 local prefixes. In that case, the architectural goal of distinguishing the "real" IPv6 addresses from the IPv6 addresses that represent IPv4 addresses can no longer be achieved in a general manner, making this option less attractive.

The usage of a single well-known prefix to map IPv4 addresses irrespectively of the NAT64 used, may results in failure modes in sites that have more than one NAT64 device. The main problem is that intra-site routing fluctuations that result in packets of an ongoing communication flow through a different NAT64 box that the one they were initially using (e.g. a change in an ECMP load balancer), would break ongoing communications. This is so because the different NAT64 boxes will use a different IPv4 address, so the IPv4 peer of the communications will receive packets coming from a different IPv4 address. This is avoided using a local address, since each NAT64 box can have a different Pref64::/06 associated, to routing fluctuations would not result in using a different NAT64 box.

The usage of a well-known prefix is also problematic in the case that different routing domains want to exchange routing information involving these routes. Consider the case of an IPv6 site that has multiple providers and that each of these providers provides access to the IPv4 Internet using the well know prefix. Consider the hypothetical case that different parts of the IPv4 Internet are reachable through different IPv6 ISPs (yes, this means that in a futuristic scenario, the IPv4 Internet is partitioned). In order to reach the different parts through the different ISPs, more specific routes representing the different IPv4 destinations reachable need to be injected in the IPv6 sites. This basically means that such configuration would imply to import the IPv4 routing entropy into the IPv6 routing system. If different local prefixes are used, then each ISP only announces its own local prefix, and then the burden of defining which IPv4 destination is reachable through which ISP is placed somewhere else (e.g. in the DNS64).

6. Security Considerations

Implications on end-to-end security, IPSec and TLS.

Any protocol that protect IP header information are essentially incompatible with NAT64. So, this implies that end to end IPSec verification will fail when AH is used (both transport and tunnel mode) and when ESP is used in transport mode. This is inherent to any network layer translation mechanism. End-to-end IPsec protection can be restored, using UDP encapsulation as described in [RFC2765].

TBD: TLS implications

Filtering.

NAT64 creates binding state using packets flowing from the IPv6 side to the IPv4 side. So, NAT64 implements by definition, at least, endpoint independent filtering, meaning that in order to enable any packet to flow from the IPv4 side to the IPv6 side, there must have been a packet flowing from the IPv6 side to the IPv4 side the created the binding information to be used for packets in the other direction. Endpoint independent filtering allows that once a binding is created, it can be used by any node on the IPv4 side to send packets to the IPv6 transport address that created the binding. This basically means that as long a the IPv6 node does not open a hole in the NAT64, incoming communications are blocked and that once that the IPv6 node has sent the first packet, this packet opens the door for any node on the IPv4 side to send packets to that IPv6 transport address. It is possible to configure the NAT64 to implement more stringent security policy, if endpoint independent filtering is considered not secure enough. In particular, if the security policy of the NAT64 requires it, is it possible to configure the NAT64 to perform address dependent filtering. This basically means that the binding state created can only be used by to send packets from the IPv4 address to which the original packet that created the binding was sent to. This basically means that the door is open only for that IPv4 address to send packet to the IPv6 transport address.

Attacks to NAT64.

The NAT64 device itself is a potential victim of different type of attacks. In particular, the NAT64 can be a victim of DoS attacks. The NAT64 box has a limited number of resources that can be consumed by attackers creating a DoS attack. The NAT64 has a limited number of IPv4 address that is uses to create the bindings. Even though the NAT64 performs address and port translation, it is possible for an attacker to consume all the IPv4 transport addresses by sending IPv6 packets with different source IPv6 transport address. It should be

noted that this attack can only be launched from the IPv6 side, since IPv4 packets are not used to create binding state. DoS attacks can also affect other limited resource available in the NAT64 such as memory or link capacity. For instance, if the NAT64 implements reassembly of fragmented packets, it is possible for an attacker to launch a DoS attack to the memory of the NAT64 device by sending fragments that the NAT64 will store for a given period. If the number of fragments if high enough, the memory of the NAT64 could be exhausted. NAT64 devices should implement proper protection against such attacks, for instance allocating a limited amount of memory for fragmented packet storage.

7. IANA Considerations

The IANA is requested to assign an EDNS Option Code value for the SAS option.

TBD: Set up an IANA registry for SAS flags??

8. Changes from Previous Draft Versions

Note to RFC Editor: Please remove this section prior to publication of this document as an RFC.

[[This section lists the changes between the various versions of this draft.]]

9. Contributors

George Tsirtsis

Qualcomm

tsirtsis@googlemail.com

10. Acknowledgements

Dave Thaler, Dan Wing, Alberto Garcia-Martinez, Reinaldo Penno and Joao Damas reviewed the document and provided useful comments to improve it.

The content of the draft was improved thanks to discussions with Fred Baker and Jari Arkko.

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

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

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

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

[RFC2765] Nordmark, E., "Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, February 2000.

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

[I-D.ietf-behave-tcp] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", draft-ietf-behave-tcp-08 (work in progress), September 2008.

[I-D.ietf-behave-nat-icmp] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP protocol", draft-ietf-behave-nat-icmp-12 (work in progress), January 2009.

[I-D.bagnulo-behave-dns64] Bagnulo, M., Sullivan, A., Matthews, P., Beijnum, I., and M. Endo, "DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", draft-bagnulo-behave-dns64-02 (work in progress), March 2009.

11.2. Informative References

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

February 2000.

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

[RFC3128] Miller, I., "Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)", RFC 3128, June 2001.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, January 2001.

[RFC4966] Aoun, C. and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status", RFC 4966, July 2007.

[I-D.ietf-mmusic-ice] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", draft-ietf-mmusic-ice-19 (work in progress), October 2007.

[RFC3498] Kuhfeld, J., Johnson, J., and M. Thatcher, "Definitions of Managed Objects for Synchronous Optical Network (SONET) Linear Automatic Protection Switching (APS) Architectures", RFC 3498, March 2003.

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

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