BEHAVE WG
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2010

M. Bagnulo
UC3M
P. Matthews
Alcatel-Lucent
I. van Beijnum
IMDEA Networks
July 4, 2009

NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers
draft-ietf-behave-v6v4-xlate-stateful-00

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 an IPv6 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 IPv6 prefix to the IPv4 address to create an IPv6 address, where the IPv6 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 function in the devices

connecting an IPv6-only network to the IPv4-only network, along with the deployment of a few DNS64-enabled name servers in the IPv6-only network. However, some advanced features such as support for DNSSEC validating stub resolvers or support for some IPsec modes, 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 roughly 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 communication 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 [RFC4787], and ICMP [RFC5508].

- 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 an IPv6 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.

The IPv4 address pool is a set of IPv4 addresses, normally a small prefix assigned by the local administrator. 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.

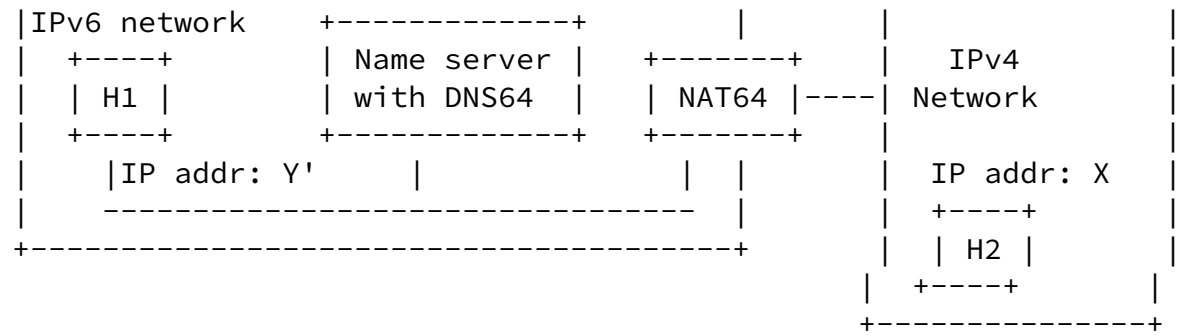
To allow an IPv6 initiator to do the standard DNS lookup to learn the address of the responder, DNS64 [I-D.bagnulo-behave-dns64] is used to synthesize an AAAA RR from the A RR (containing the real IPv4 address of the responder). DNS64 receives the DNS queries generated by the IPv6 initiator. If there 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. Walkthrough

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 IPv4 network.

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 network. This NAT64 has a /96 prefix (called Pref64::/96) that it uses to represent IPv4 addresses in the IPv6 address space and an IPv4 address T assigned to its IPv4 interface. the routing is configured in such a way, that the IPv6 packets addressed to a destination address containing Pref64::/96 are routed to the IPv6 interface of the NAT64 box.

Also shown is a local name server with DNS64 functionality. 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 set by H1.

3. The packet is routed to the IPv6 interface of the NAT64 (since the IPv6 routing is configured that way).

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.

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

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

7. 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.

8. 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 IPv6 representation of H2's IPv4 address (i.e. Pref64:X) 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 source transport address equal to the destination transport address of the incoming packet and destination IP address equal to the source IP address of the incoming packet.

NAT64 filtering is consistent with the recommendations of RFC 4787 [RFC4787], and the ones of RFC 5382 [RFC5382]

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:

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.

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

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

Endpoint-Independent Mapping: In NAT64, using the same mapping for all the sessions involving a given IPv6 transport address of an IPv6 host (irrespectively of the transport address of the IPv4 host involved in the communication). 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.

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.

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.

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

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.

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).

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 at least one IPv6 interface and at least one IPv4 interface. Each NAT64 device MUST have one unicast /96 IPv6 prefix assigned to it, denoted Pref64::/96. Each NAT64 box MUST have one or more unicast IPv4 addresses assigned to it.

A NAT64 uses the following dynamic data structures:

- UDP Binding Information Base
- UDP Session Table
- TCP Binding Information Base
- TCP Session Table

A NAT64 has two Binding Information Bases (BIBs): 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 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 is 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 error message, then the 5-tuple is computed by extracting the appropriate fields from the IP packet embedded inside the ICMP error 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 source IPv4 transport address equal to the destination IPv4 transport address in the incoming 5-tuple and destination IPv4 address (in the session table entry) 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 MAY 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 is 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.

3.3. 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.

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.

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.

TBD: Do we delete the session entry if we cannot create a BIB entry? [R/T] Yes, we think that the session entry should be deleted if we cannot create a BIB entry as it wouldn't make much sense to have a session entry without a BIB entry. Greg: I don't think you can make

a Session Table entry if you cannont create a BIB entry. You have to

look up BIB first, to determine the correct outgoing (T,t), or to make a new BIB entry. Only after that result can you create a completed Session Table entry.

If the rules specify that the packet is discarded, then the NAT64 MAY 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).

3.4. Translating the Packet

This step translates the packet from IPv6 to IPv4 or vice-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 outgoing 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 efficiently updating these checksums is described in [RFC3022].

- When the protocol following the IP header is ICMP (sections 3.4 and 4.4) and it is an ICMP error message, 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).

3.5. 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.

[R/T] The reference to step 2 here was a little confusing to us. Are you referring to Filtering and Updating Session Information (Section 3.2)? MB> I am not sure about this anymore. I mean what if the packet that is being hairpinned is an ICMP error msge, I mean don't we still need step 1?

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.

3.6. Path MTU discovery and fragmentation

It's the job of the network layer to adapt to different maximum packet sizes as packets move through the network. There are three mechanisms that handle this: transport layer negotiations such as the TCP MSS option, path MTU discovery and fragmentation. The difference between the IPv4 and IPv6 header sizes requires some handling in a NAT64 translator, and there are complications because of the differences between how IPv4 and IPv6 handle fragmentation, as well as the issue of how to demultiplex fragmented IPv4 packets.

The vast majority of both IPv4 and IPv6 hosts use path MTU discovery [RFC1191] [RFC1981]. With IPv4, PMTUD can be enabled on a per-packet basis by setting the DF bit to 1. With IPv6, there is no need for PMTUD for packets up to 1280 bytes because all IPv6 hosts are required to be able to receive 1280-byte packets without fragmentation. When sending larger packets, IPv6 hosts implicitly use PMTUD.

The fragmentation behavior specified in [RFC2765] is that upon the reception of an ICMPv6 "packet too big" message with an indicated packet size of less than 1280 octets, IPv6 hosts will transmit 1280-

octet packets, but include a fragment header in those packets. In a stateful translator, the identification value in this fragment header can't be used, so the fragment header itself serves no purpose. Additionally, the presence or absense of the fragment header isn't enough to determine whether to set the DF bit in packets translated to IPv4 to 0 (fragment header present) or 1 (no fragment header present). The reason for this is that operators may decide to forego path MTU discovery by configuring an MTU of 1280 and filtering incoming "too big" messages. The behavior specified below is meant to avoid PMTUD black holes in this situation

3.6.1. Translating whole packets and PMTUD

This section specifies the values in the fragmentation-related fields in the IPv4 header when no fragmentation occurs, and how path MTU

discovery is handled.

3.6.1.1. IPv6-to-IPv4 translation

If the NAT64 has the same MTUs on its IPv6 and IPv4 interfaces, it will never have to generate "packet too big" messages for incoming IPv6 packets because the translation from IPv6 to IPv4 reduces the packet size by 20 bytes, more if the IPv6 packet has extension headers that are removed during the translation, such as the fragment header. If the MTU on the IPv6 side is larger than 1280 bytes and more than 20 bytes smaller than the MTU on the IPv4 side, the NAT64 MUST generate the appropriate "packet too big" messages on the IPv6 side.

To support PMTUD, for translated packets that are larger than 1260 bytes on the IPv4 side (1280 bytes IPv6 packets with 20 byte size reduction through the translation), the DF bit is set to 1 in the resulting IPv4 packet.

IPv4 routers may generate "packet too big" messages indicating a supported MTU size smaller than 1280 bytes. In those cases, the IPv6 hosts will continue to send packets larger than what the IPv4 path MTU can support. To allow packets to be delivered successfully in this case, the DF bit is set to 0 in all translated packets smaller than or equal to 1260 bytes, to allow these packets to be fragmented in the IPv4 network.

Note: it is highly recommended for IPv4 hosts running services that may be used by IPv6 clients through a NAT64 translator to use an MTU size of at least 1260 bytes and to properly generate "packet too big" messages.

When a NAT64 translates "packet too big" messages from IPv6 to IPv4, it adjusts the advertised MTU to the minimum of the original advertised MTU + 20, the NAT64's MTU on the IPv6 side + 20 and the NAT64's MTU on the IPv4 side.

The identification field in the IPv4 header MUST be filled with a value generated by the NAT64 translator, similar to the way that identification values are created for locally generated packets. It is RECOMMENDED that a NAT64 translator keep an identification counter for every combination of remote IPv4 destination and protocol.

In theory, IPv4 packets with DF set to 1 don't need a unique identification value. However, it is not unheard of for operators to configure equipment to clear the DF bit, at which time an identification value with good uniqueness becomes necessary. As such, it is recommended that translators include a unique

identification value in all packets, including those with DF set to 1. However, since more packets will be sent with DF set to 1, this will use up identification values faster. Implementations may choose to segment the identification space and assign values from non-overlapping pools to packets with DF set to 0 and DF set to 1 to provide a longer period of uniqueness to fragmentable packets.

3.6.1.2. IPv4-to-IPv6

Because it may be necessary to include a fragmentation header or other extension header, the NAT64 MUST be prepared to generate "packet too big" messages for packets with the DF bit set to 1 received from the IPv4 side, regardless of the MTU sizes on the IPv4 and IPv6 interfaces. If the packet with DF = 1 is larger than can be transmitted on the IPv6 side after translation, the NAT64 returns a "packet too big" message indicating the maximum IPv4 packet size that would be supported using the same translation as the current packet. This can be calculated as IPv4-packet-size - 20.

When a NAT64 translates "packet too big" messages from IPv4 to IPv6, it adjusts the advertised MTU to the minimum of the original advertised MTU - 20, the NAT64's MTU on the IPv6 side and the NAT64's MTU on the IPv4 side - 20. However, if the advertised MTU in "packet too big" messages is smaller than 1260 bytes, the value put into the translated "packet too big" message is 1280. This makes sure that the IPv6 host will limit its packet sizes to 1280 bytes, so its packets are subsequently translated into IPv4 packets with DF set to 0. (This deviates from [RFC2765].)

3.6.2. Fragmentation

Because NAT deviates from normal router behavior, the limitation that IPv6 packets or IPv4 packets with DF set to 1 are not fragmented by routers doesn't apply to a NAT64 translator. Where appropriate, these packets are fragmented after translation as described below.

3.6.2.1. IPv4-to-IPv6

Because packets coming in on the IPv4 side may be larger than 1280 bytes after translation, a NAT64 MUST implement PMTUD on the IPv6 side. In other words, it must react to "packet too big" messages for any IPv6 destination that it communicates with by limiting the size of the packets that it sends to the advertised maximum.

In the case where, after translation from IPv4 to IPv6, a packet is larger than a destination's PMTU, the NAT64 returns a "packet too big" as outlined earlier in the case that the DF bit was set to 1 in the IPv4 packet. If the DF bit was set to 0, the translator first

translates the IPv4 packet, and then fragments the resulting IPv6 packets using normal IPv6 fragmentation rules. The lower 16 bits of the IPv6 identification field are copied from the IPv4 identification field. The upper 16 bits of the IPv6 identification field are set to 0.

Because NAT64 provides a stateful many-to-one (perhaps even many-to-many) translation, it is necessary to recognize which session a given packet belongs to. In the IPv4-to-IPv6 direction, the TCP or UDP port numbers must be known to accomplish this, but the port numbers only occur in the first fragment of a fragmented packet. There are two possible ways to deal with this:

1. Reassemble the packet before translating it.

2. Create translation state for the fragments belonging to the same packet so each packet can be translated.

Strategy 2 is attractive in large installations because it requires less storage and processing. However, it may still be necessary to buffer fragments for some time, as the fragment containing the first part of the packet (and with that, the port numbers) may not be the first one to arrive.

Note: based on the assumptions that hosts generate fragments in-order and that reordering must happen through parallel network links and that the path between these parallel links and a NAT64 supports speeds of at least 10 Mbps, there is a very high probability that two out-of-order fragments making up a packet will arrive at the NAT64 within 50 to 100 milliseconds. Further assuming that fragmented traffic makes up less than 10% of all traffic, this only requires a buffer of 6 to 12,500 fragments (50 ms at 10 Mbps to 100 ms at 10 Gbps).

In some cases, there may only be a single session matching the fragment's source and destination addresses and protocol number. In these cases, it would be possible to translate the fragments out-of-order. A NAT64 translator MAY do this for TCP, however, it MUST NOT translate UDP packets before the first fragment is available. The reason for this is that the fragment could be part of a packet setting up a new session. However, with TCP session establishment packets don't carry data, so it's extremely unlikely that they are fragmented. This is not the case with UDP, and in the IPv4-to-IPv6 direction, a UDP packet may have a zero checksum, which must be recalculated when translating to IPv6, for which the entire packet must be available.

3.6.2.2. IPv6-to-IPv4

For all IPv4 packets that the NAT64 creates through translation, the translator generates an ID value. This applies to all packets, regardless of their size or the value of the DF field. A NAT64

translator MAY employ strategies to avoid reusing an ID value for a certain source, destination, protocol tuple as long as possible. If the IPv4 packets are fragments of an IPv6 packet, then state is created that makes it possible for all the fragments to have the same ID value on the IPv4 side.

[RFC2765] specifies copying the lower bits from the IPv6 ID field in a fragment header (if present) to the IPv4 ID field, but this runs the risk of two IPv6 hosts talking to the same IPv4 destination through the NAT64 using the same ID value.

Otherwise, when translating IPv6 packets with a fragmentation header, the fragments are translated as per [RFC2765].

In the IPv6-to-IPv4 direction, there is no need to map a fragment to the session it belongs to in order to translate the fragment. However, it is necessary that all the fragments have the same identification value, so fragments may be translated individually, but state must be kept to be able to translate subsequent fragments of the same packet using the same identification value on the IPv4 side.

3.6.3. TCP MSS option

It is not recommended that NAT64 translators rewrite the TCP MSS option [RFC0793]. As such, assuming the common case of all 1500-octet MTUs, an IPv6 host will advertise a 1440-octet MSS, triggering the IPv4 host to generate 1480-octet packets that are translated to 1500-octet IPv6 packets. IPv4 hosts will advertise a 1460-octet MSS, which would be 1520-octet IPv6 packets. However, ethernet-connected IPv6 hosts can only send 1500-octet packets, so in the all-ethernet case, there is no dependency on path MTU discovery.

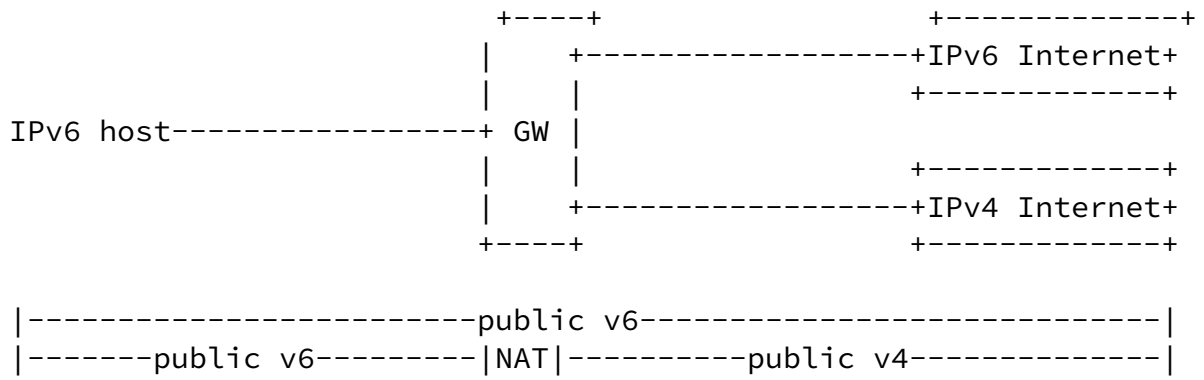
4. Application scenarios

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.

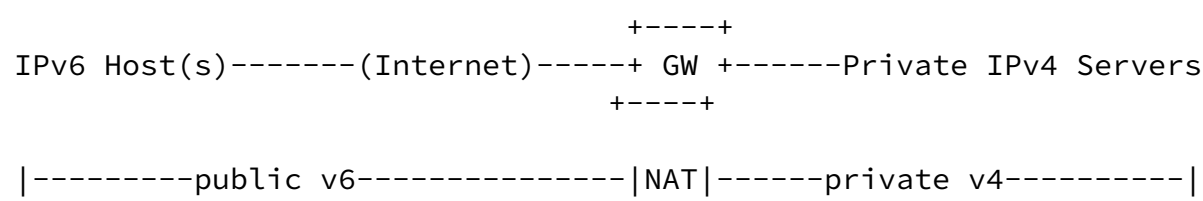


The proposed NAT64/DNS64 is perfectly suitable for this particular scenario. The deployment of the NAT64/DNS64 would be as follows: The NAT64 function should be located in the GW device that connects the IPv6 site to the IPv4 Internet. The DNS64 functionality can be placed either in the local recursive DNS server or in the local resolver in the hosts.

The proposed NAT64/DNS64 approach satisfies the requirements of this scenario, in particular because 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 IPv4 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 required in general since real (i.e. non

synthetic) AAAA RRs for the IPv4 servers containing the IPv6 representation of the IPv4 address of the servers can be created. See more discussion about this in [I-D.bagnulo-behave-dns64]

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. Security Considerations

Implications on end-to-end security.

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 [RFC3948].

Filtering.

NAT64 creates binding state using packets flowing from the IPv6 side to the IPv4 side. In accordance with the procedures defined in this document following the guidelines defined in RFC 4787 [RFC4787] a NAT64 must offer "enpoint independent filtering". This means:

> for any IPv6 side packet with source (S'1,s1) and destination (Pref64::D1,d1) that creates an external mapping to (S1,s1), (D1,d1),
>
> for any subsequent external connection to from S'1 to (D2,d2) within a given binding timer window,
>
> (S1,s1) = (S2,s2) for all values of D2,d2

Implementations may also provide support for "Address-Dependent Mapping" and "Address and Port-Dependent Mapping", as also defined in this document and following the guidelines defined in RFC 4787 [RFC4787].

The security properties however are determined by which packets the

NAT64 filter allows in and which it does not. The security properties are determined by the filtering behavior and filtering configuration in the filtering portions of the NAT64, not by the address mapping behavior. For example,

Without filtering - When "endpoint independent filtering" is used in NAT64, once a binding is created in the IPv6 ---> IPv4 direction, packets from any node on the IPv4 side destined to the IPv6 transport address will traverse the NAT64 gateway and be forwarded to the IPv6 transport address that created the binding. However,

With filtering - When "endpoint independent filtering" is used in NAT64, once a binding is created in the IPv6 ---> IPv4 direction, packets from any node on the IPv4 side destined to the IPv6 transport address will first be processed against the filtering rules. If the source IPv4 address is permitted, the packets will be forwarded to the IPv6 transport address. If the source IPv4 address is explicitly denied -- or the default policy is to deny all addresses not explicitly permitted -- then the packet will discarded. A dynamic filter may be employed where by the filter will only allow packets from the IPv4 address to which the original packet that created the binding was sent. This means that only the D IPv4 addresses to which the IPv6 host has initiated connections will be able to reach the IPv6 transport address, and no others. This essentially narrows the effective operation of the NAT64 device to a "Address Dependent" behavior, though not by its mapping behavior, but instead by its filtering behavior.

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 addresses that it 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 addresses. 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 resources available in the NAT64 such as memory or link capacity. For instance, 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 is 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.

6. IANA Considerations

7. 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.]]

8. Contributors

George Tsirtsis

Qualcomm

tsirtsis@googlemail.com

Greg Lebovitz

Juniper

gregory.ietf@gmail.com

9. 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.

10. References

10.1. 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.

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

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

[RFC5382] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, October 2008.

[RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, April 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.

10.2. Informative References

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[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
Avda. del Mar Mediterraneo, 22
Leganes, Madrid 28918
Spain

Email: iljitsch@muada.com