BEHAVE WG
Internet-Draft
Intended status: Standards Track
Expires: June 20, 2010

M. Bagnulo
UC3M
P. Matthews
Alcatel-Lucent
I. van Beijnum
IMDEA Networks

December 17, 2009

# NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers
## draft-ietf-behave-v6v4-xlate-stateful-07

## 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. NAT64 also support IPv4 initiated communications to a subset of the IPv6 hosts through statically configured bindings in the NAT64. This document specifies NAT64, and gives suggestions on how it should be deployed.

http://www.ietf.org/shadow.html.

This Internet-Draft will expire on June 20, 2010.

Copyright Notice

Table of Contents

## 1. Introduction

This document specifies NAT64, a mechanism for IPv6-IPv4 transition and co-existence. Together with DNS64 [I-D.ietf-behave-dns64], these two mechanisms allow a IPv6-only client to initiate communications to an IPv4-only server, 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. NAT64 also support IPv4 initiated communications to a subset of the IPv6 hosts through statically configured bindings in the NAT64.

NAT64 is a mechanism for translating IPv6 packets to IPv4 packets and vice-versa. The translation is done by translating the packet headers according to IP/ICMP Translation Algorithm [I-D.ietf-behave-v6v4-xlate], 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 the general case of 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 are:

- NAT64 as specified in this document is compliant with the recommendations for how NATs should handle UDP [RFC4787], TCP [RFC5382], and ICMP [RFC5508]. As such, NAT64 only supports Endpoint-Independent mappings and supports both Endpoint-Independent and Address dependent filtering. Because of the compliance with the aforementioned requirements, NAT64 is compatible with ICE [I-D.ietf-mmusic-ice].

- In the absence of any state in NAT64 regarding a given IPv6 node, only said IPv6 node can initiate sessions to IPv4 nodes. This works for roughly the same class of applications that work through IPv4-to-IPv4 NATs.

- Depending on the filtering policy used (Endpoint-Independent, or Address-Dependent), IPv4-nodes MAY be able to initiate sessions to

a given IPv6 node, if the NAT64 somehow has an appropriate mapping (i.e.,state) for said IPv6 node, via one of the following mechanism.

* The IPv6 node has recently initiated a session to the same or other external-IPv4 node.

* The IPv6 node has used a NAT-traversal technique (such as ICE) which essentially results in the previous bullet point.

* If static configuration (i.e. mapping) exists regarding said IPv6 node

## 1.2. Overview

This section provides a non-normative introduction to the mechanisms of NAT64. This is achieved by describing the NAT64 behavior involving a simple setup, that involves a single NAT64 box, a single DNS64 box and a simple network topology. The goal of this description is to provide the reader with a general view of NAT64. It is not the goal of this section to describe all possible configurations nor to provide a normative specification of the NAT64 behavior. The normative specification of NAT64 is provided in Section 3.

NAT64 mechanism is implemented in an NAT64 box which has (at least)





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 either statically configured in the NAT64 or it 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, in the general case, NAT64 only supports communications initiated by the IPv6-only node towards an IPv4-only node. Some additional mechanisms (like ICE) or static binding configuration, can be used to provide support for communications initiated by the IPv4-only node to the IPv6-only node.

### 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 IP/ICMP Translation Algorithm [I-D.ietf-behave-v6v4-xlate].

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).

The IPv6 address pool is an IPv6 prefix assigned to the translator itself (hereafter called Pref64::/n). Due to the abundance of IPv6 address space, it is possible to assign an Pref64::/n that is equal or even bigger than the whole IPv4 address space. This allows each IPv4 address to be mapped into a different IPv6 address by simply concatenating the Pref64::/n with the IPv4 address being mapped and a suffix (i.e. an IPv4 address X is mapped into the IPv6 address





Pref64:X:SUFFIX). The provisioning of the Pref64::/n is discussed at length in [I-D.ietf-behave-address-format]

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, except for the static/manually created ones, 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 algorithmically 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.

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.ietf-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 returned, DNS64 creates a synthetic AAAA RR that includes the IPv6 representations of the IPv4 address created by concatenating the Pref64::/n of a NAT64 to the responder's IPv4 address and a suffix (i.e. if the IPv4 node has IPv4 address X, then the synthetic AAAA RR will contain the IPv6 address formed as Pref64:X:SUFFIX). 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

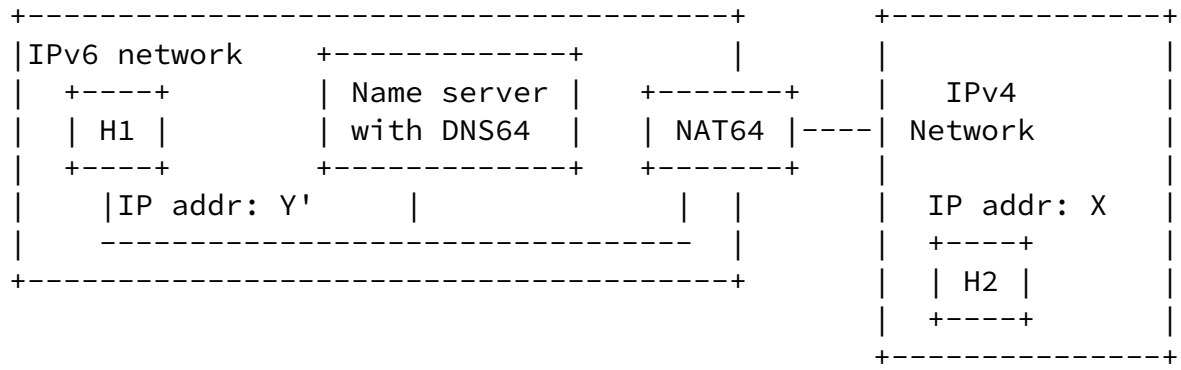
before.

### 1.2.2. NAT64 Behaviour 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 of length n are indicated by "P::/n", an IPv6 address built from an IPv4 address X by adding the prefix P and a suffix SUFFIX is indicated as "P:X:SUFFIX"", 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 /n prefix (called Pref64::/n) that it uses to represent IPv4 addresses in the IPv6 address space and a single 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::/n 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 /n prefix assigned to the local NAT64 (Pref64::/n). For the purpose of this example, we assume it learns this through manual configuration.

For this example, assume the typical DNS situation where IPv6 hosts have only stub resolvers and 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 Pref64::/n associated to the NAT64 box and the IPv4 address of H2 and a suffix (i.e. Pref64:X:SUFFIX).

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:SUFFIX,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 IP/ICMP Translation Algorithm [I-D.ietf-behave-v6v4-xlate].

   * 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 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 IP/ICMP Translation Algorithm [I-D.ietf-behave-v6v4-xlate].

   * The NAT64 includes (Y',y) as destination transport address in the packet and (Pref64:X:SUFFIX,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:SUFFIX) 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::/n in this document) to the original IPv4 address and a suffix.

### 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. The NAT64 can do filtering of IPv6 packets based on the administrative rules to create BIB and session entries. The filtering can be flexible enough and broad enough but the idea of the filtering is to provide the operators necessary control to avoid DoS attacks that would result in exhaustion of NAT64 address, port, memory and CPU resources. Filtering techniques of incoming IPv6 packets is not specific to the NAT64 and therefore is not described in this specification.

Filtering of IPv4 packets on the other hand is tightly coupled to the NAT64 state and therefore is described in this specification. In this document, we consider that the NAT64 may do no filtering, or it may filter incoming IPv4 packets.

NAT64 filtering of incoming IPv4 packets is consistent with the recommendations of RFC 4787 [RFC4787], and the ones of RFC 5382 [RFC5382]. Because of that, the NAT64 as specified in this document, supports both Endpoint-Independent filtering and Address-Dependent filtering, both for TCP and UDP.

If a NAT64 performs Endpoint-Independent filtering of incoming IPv4 packets, then an incoming IPv4 packet is dropped unless the NAT64 has state for the destination transport address of the incoming IPv4 packet.

If a NAT64 performs Address-Dependent filtering of incoming IPv4 packets, then an incoming IPv4 packet is dropped unless the NAT64 has state involving the destination transport address of the IPv4





incoming packet and the particular source IP address of the incoming IPv4 packet.

## 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:

3-Tuple: The tuple (source IP address, destination IP address, Query Identifier). A 3-tuple uniquely identifies an ICMP Query session. When an ICMP Query session flows through a NAT64, each session has two different 3-tuples: one with IPv4 addresses and one with IPv6 addresses.

5-Tuple: The tuple (source IP address, source port, destination IP address, destination port, transport protocol). A 5-tuple uniquely identifies a UDP/TCP session. When a UDP/TCP 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 three BIBs, one for TCP, one for UDP and one for ICMP Queries.

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.

Filtering, Endpoint-Independent: The NAT64 filters out only incoming IPv4 packets not destined to a transport address for which there is not state in the NAT64, regardless of the source IPv4 transport address. The NAT forwards any packets destined to any transport address for which it has state. In other words, having state for





a given transport address is sufficient to allow any packets back to the internal endpoint. See [RFC4787] for the definition of the different types of filtering in IPv4-to-IPv4 NATs.

Filtering, Address-Dependent: The NAT64 filters out incoming IPv4 packets not destined to a transport address for which there is no state (similar to the Endpoint-Independent filtering). Additionally, the NAT64 will filter out incoming IPv4 packets coming from IPv4 address X and destined for a transport address that it has state for if the NAT64 has not sent packets to X previously (independently of the port used by X). In other words, for receiving packets from a specific IPv4 endpoint, it is necessary for the IPv6 endpoint to send packets first to that specific IPv4 endpoint's IP address.

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 through the NAT.

Mapping or Binding: 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. The NAT64 uses mapping state to perform the translation between IPv6 and IPv4 addresses. The translation involves not only the IP header, but also the transport header (TCP or UDP).

Session: A TCP, UDP or ICMP Query session. In other words, the bi-directional flow of packets between 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 three session tables, one for TCP, one for UDP and one for ICMP Queries.

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).

Tuple: Refers to either a 3-Tuple or a 5-tuple as defined above.

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 /n IPv6 prefix assigned to it, denoted Pref64::/n (Additional consideration about the Pref64::/n are presented in Section 3.2.4). 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
- ICMP Query Binding Information Base
- ICMP Query Session Table

These tables contain information needed for the NAT64 processing. The actual division of the information into six tables is done in order to ease the description of the NAT64 behaviour. NAT64 implementations MAY use different data structures as long as they store all the required information and the externally visible outcome is the same as the one described in this document.

A NAT64 has three Binding Information Bases (BIBs): one for TCP, one for UDP and one for ICMP Queries. In the case of UDP and TCP BIBs, 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. The BIB has then two columns, the BIB IPv6 transport address and the BIB IPv4 transport address. 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. This implementation of the BIBs ensures Endpoint-Independent mappings in the NAT64. The information in the BIBs is also used to implement Endpoint-Independent filtering. (Address-Dependent filtering is implemented using the Session tables described below.)

In the case of the ICMP Query BIB, each ICMP Query BIB entry specify a mapping between an (IPv6 address, IPv6 Identifier) pair and an (IPv4 address, IPv4 Identifier) pair.

```
(X',I1) <--> (T,I2)
```

where X' is some IPv6 address, T is an IPv4 address, I1 is an ICMPv6 Identifier and I2 is an ICMPv4 Identifiers. T will always be one of the IPv4 addresses assigned to the NAT64. A given (IPv6 or IPv4 address, IPv6 or IPv4 Identifier) pair can appear in at most one entry in the ICMP Query BIB.

Entries in any of the three BIBs can be created dynamically as the result of the flow of packets as described in the Section 3.2 but the can also be created manually by the system administrator. NAT64 implementations SHOULD support manually configured BIB entries for any of the three BIBs. Dynamically-created entries are deleted from the corresponding BIB when the last session associated to the BIB entry is removed from the session table. Manually-configured BIB entries are not deleted when there is no corresponding session table entry and can only be deleted by the administrator.

A NAT64 also has three session tables: one for TCP sessions, one for UDP sessions and one for ICMP Query sessions. Each entry keeps information on the state of the corresponding session. In the TCP and UDP session tables, each entry specifies a mapping between a pair of IPv6 transport address and a pair of IPv4 transport address:

```
(X',x),(Y',y) <--> (T,t),(Z,z)
```

where X' and Y' are IPv6 addresses, T and Z are IPv4 addresses, and x, y, z and t are ports. T will always be one of the IPv4 addresses assigned to the NAT64. Y' is always the IPv6 representation of the IPv4 address Z, so Y' is obtained from Z using the algorithm applied by the NAT64 to create IPv6 representations of IPv4 addresses. y will always be equal to z.





For each Session Table Entry (STE), there are then five columns:

The STE source IPv6 transport address, (X',x) in the example above,

The STE destination IPv6 transport address, (Y',y) in the example above,

The STE source IPv4 transport address, (T,t) in the example above, and,

The STE destination IPv4 transport address, (Z,z) in the example above.

The STE lifetime.

The terminology used for the session table entry columns is from the perspective of an incoming IPv6 packet being translated into an outgoing IPv4 packet.

In the ICMP query session table, each entry specifies a mapping between a 3-tuple of IPv6 source address, IPv6 destination address and ICMPv6 Query Id and a 3-tuple of IPv4 source address, IPv4 destination address and ICMPv4 Query Id:

(X',Y',I1) <--> (T,Z,I2)

where X' and Y' are IPv6 addresses, T and Z are IPv4 addresses, I1 is an ICMPv6 Identifier and I2 is an ICMPv4 Identifier. T will always be one of the IPv4 addresses assigned to the NAT64. Y' is always the IPv6 representation of the IPv4 address Z, so Y' is obtained from Z using the algorithm applied by the NAT64 to create IPv6 representations of IPv4 addresses.

For each Session Table Entry (STE), there are then six columns:

The STE source IPv6 address, X' in the example above,

The STE destination IPv6 address, Y' in the example above,

The STE IPv6 Identifier, I1 in the example above,

The STE source IPv4 address, T in the example above,

The STE destination IPv4 address, Z in the example above, and,

The STE IPv4 Identifier, I2 in the example above.

The STE lifetime.

The NAT64 uses the session state information to determine when the session is completed, and also uses session information for Address-Dependent filtering. A session can be uniquely identified by either an incoming tuple or an outgoing tuple.

For each TCP or UDP 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). For each ICMP Query session, there is a corresponding BIB entry, uniquely specified by either the source IPv6 address and ICMPv6 Query Id (in the IPv6 --> IPv4 direction) or the destination IPv4 address and the ICMPv4 Query Id (in the IPv4 --> IPv6 direction). However, for all the BIBs, a single BIB entry can have multiple corresponding sessions. When the last corresponding session is deleted, if the BIB entry was dynamically created, the BIB entry is deleted.

The NAT64 will receive packets through its interfaces. These packets can be either IPv6 packets or IPv4 packets and they may carry TCP traffic, UDP traffic or ICMP traffic. The processing of the packets will be described next. In the case that the processing is common to all the aforementioned types of packets, we will refer to the packet as the incoming packet in general. In case that the processing is specific to IPv6 packets, we will refer to the incoming IPv6 packet and similarly to the IPv4 packets.

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

1. Determining the incoming tuple

2. Filtering and updating binding and session information

3. Computing the outgoing 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.

### 3.1. Determining the Incoming tuple

This step associates a incoming tuple with every incoming IP packet for use in subsequent steps. In the case of TCP, UDP and ICMP error packets, the tuple is a 5-tuple consisting of source IP address, source port, destination IP address, destination port, transport protocol. In case of ICMP Queries, the tuple is a 3-tuple consisting of the source IP address, destination IP address and Query Identifier.

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 packet is an ICMP query message (i.e. an ICMPv4 Query message or an ICMPv6 Informational message), the 3-tuple is the source IP address, the destination IP address and the ICMP Query Identifier.

If the incoming IP packet contains a complete (un-fragmented) ICMP error message containing a UDP or a TCP segment, 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.

If the incoming IP packet contains a complete (un-fragmented) ICMP error message containing an ICMP Query message, then the 3-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 3-tuple, the embedded destination IP address becomes the source address in the 3-tuple and the embedded Identifier is used as the Identifier of the 3-tuple If it is not possible to determine the 3-tuple (perhaps because not enough of the embedded packet is reproduced inside the

ICMP message), then the incoming IP packet is silently discarded.

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 the external behavior of the NAT64 is compliant with the following conditions:





The NAT64 MUST handle fragments, even if the arrive out-of-order, conditioned to the following:

The NAT64 MUST limit the amount of resources devoted to the storage of fragmented packets in order to protect from DoS attack.

As long as the NAT64 has available resources, the NAT64 MUST allow the fragments to arrive over a time interval. The time interval MUST be configurable and the default value MUST be of at least 10 seconds.

The NAT64 MAY require that the UDP, TCP, or ICMP header be completely contained within the fragment that contains OFFSET equal to zero.

For incoming packets carrying TCP or UDP fragments with non-null checksum, 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, subsequent fragments may arrive before the first.

For incoming IPv4 packets carrying UDP segments with null checksum, if the NAT64 has enough resources, the NAT64 MUST reassemble the packets and MUST calculate the checksum. If the NAT64 does not have enough resources, then it will silently discard the packets.

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

with reassembling packets at high rates, described in [RFC4963].

## 3.2. Filtering and Updating Binding and Session Information

This step updates binding and session information stored in the appropriate tables. This step may also filter incoming packets, if desired.

Irrespectively of the transport protocol used, the NAT64 must silently discard all incoming IPv6 packets containing a source address that contains the Pref64::/n. This is required in order to prevent hairpinning loops as described in the Security Considerations section. In addition, the NAT64 function will only process incoming IPv6 packets that contain a destination address that contains Pref64::/n. Likewise, the NAT64 function will only process incoming





IPv4 packets that contain a destination address that belong to the IPv4 pool assigned to the NAT64.

The details of this step depend on the protocol (UDP, TCP or ICMP Query).

### 3.2.1. UDP Session Handling

The state information stored for a UDP session in the UDP session table includes a timer that tracks the remaining lifetime of the UDP session. When the timer expires, the UDP session is deleted. If all the UDP sessions corresponding to a UDP BIB entry are deleted, then the UDP BIB entry is also deleted (only applies to the case of dynamically created entries).

An IPv6 incoming packet with an incoming tuple with source transport address (X',x) and destination transport address (Y',y) is processed as follows:

The NAT64 searches for a UDP BIB entry that contains an BIB IPv6 transport address that matches the IPv6 source transport address (X',x). If such an entry does not exists, a new entry is created. As BIB IPv6 transport address, the source IPv6 transport address of the packet (X',x) is included and the BIB IPv4 transport address is set to (T,t) which is allocated using the rules defined in Section 3.2.1.1. The result is a BIB entry as follows: (X',x)

<--> (T,t).

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. The information included in the session table is as follows:

The STE source IPv6 transport address is set to (X',x), the source IPv6 transport addresses contained in the received IPv6 packet,

The STE destination IPv6 transport address is set to (Y',y), the destination IPv6 transport addresses contained in the received IPv6 packet,

The STE source IPv4 transport address is extracted from the corresponding UDP BIB entry i.e. is set to (T,t),

The STE destination IPv4 transport is set to (Z(Y'),y), y being the same port as the STE destination IPv6 transport address and Z(Y') being algorithmically generated from the IPv6 destination address (i.e. Y') using the reverse algorithm as specified in





Section 3.2.4 .

The result is a Session table entry as follows: (X',x),(Y',y) <--> (T,t),(Z(Y'),y)

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. The packet is translated and forwarded as described in the following sections.

An IPv4 incoming packet, with an incoming tuple with source IPv4 transport address (Y,y) and destination IPv4 transport address (X,x) is processed as follows:

The NAT64 searches for a UDP BIB entry that contains an BIB IPv4 transport address matches (Y,y) i.e. the IPv4 destination transport address in the incoming IPv4 packet. If such an entry does not exists, the packet is dropped. 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).

If the NAT64 applies Address-Dependent 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 STE source IPv4 transport address equal to (X,x) (i.e. the destination IPv4 transport address in the incoming packet) and STE destination IPv4 address equal to Y (i.e. the source IPv4 address in the incoming packet). 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).

In case the packet is not discarded in the previous processing (either because the NAT64 is not filtering or because the packet is compliant with the Address-dependent filtering rule), then the NAT64 searches for the session table entry corresponding containing the STE source IPv4 transport address equal to (X,x) and the STE destination IPv4 transport address equal to (Y,y). If no such entry is found, a new entry is created. In case a new UDP session table entry is created, it contains the following information:





The STE source IPv6 transport address is extracted from the corresponding UDP BIB entry

The STE destination IPv6 transport address is set to (Z'(Y),y), y being the same port y than the destination IPv4 transport address and Z'(Y) being the IPv6 representation of Y, generated using the algorithm described in Section 3.2.4

The STE source IPv4 transport address is set to (X,x) the destination IPv4 transport addresses contained in the received IPv4 packet,

The STE destination IPv4 transport is set to (Y,y), the source IPv4 transport addresses contained in the received IPv4 packet.

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.

#### 3.2.1.1. Rules for allocation of IPv4 transport addresses for UDP

If the rules specify that a new UDP 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 of the IPv4 pool assigned to the NAT64 to be used for translation.

If the port s is in the Well-Known port range 0..1023, then the NAT64 SHOULD allocate a port t from this same range. Otherwise, if the port s is in the range 1024..65535, then the NAT64 SHOULD allocate a port t from this range. Furthermore, if port s is even, then t SHOULD be even, and if port s is odd, then t SHOULD be odd. (this behavior is recommended in Section 7.1 of [RFC5382])

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 (referring to the UDP and TCP BIBs).

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. The NAT64 MAY send an ICMPv6 Destination Unreachable/ Address unreachable (Code 3) message.





### 3.2.2. TCP Session Handling

The state information stored for a TCP session:

Binding:(X',x),(Y',y) <--> (T,t),(Z,z)

Lifetime: is a timer that tracks the remaining lifetime of the TCP session. When the timer expires, the TCP session is deleted. If all the TCP sessions corresponding to a TCP BIB entry are deleted, then the TCP BIB entry is also deleted (only applies to the case of dynamically created entries).

TCP sessions are expensive, because their inactivity lifetime is set to at least 2 hours and 4 min (as per [RFC5382]), so it is important that each TCP session table entry corresponds to an existent TCP session. In order to do that, for each TCP session established through it, it tracks the corresponding state machine as follows.

The states are the following ones:

CLOSED: Analogous to [RFC0793], CLOSED is a fictional state because it represents the state when there is no state for this particular 5-tuple, and therefore, no connection.

V4 SYN RCV: An IPv4 packet containing a TCP SYN was received by the NAT64, implying that a TCP connection is being initiated from the IPv4 side. The NAT64 is now waiting for a matching IPv4 packet containing the TCP SYN in the opposite direction.

V6 SYN RCV: An IPv6 packet containing a TCP SYN was received by the NAT64, implying that a TCP connection is being initiated from the IPv6 side. The NAT64 is now waiting for a matching IPv4 packet containing the TCP SYN in the opposite direction.

ESTABLISHED: Represent an open connection, with data flowing in both directions.

V4 FIN RCV: An IPv4 packet containing a TCP FIN was received by the NAT64, data can still flow in the connection, the NAT64 is waiting for a matching TCP FIN in the opposite direction.

V6 FIN RCV: An IPv6 packet containing a TCP FIN was received by the NAT64, data can still flow in the connection, the NAT64 is waiting for a matching TCP FIN in the opposite direction.

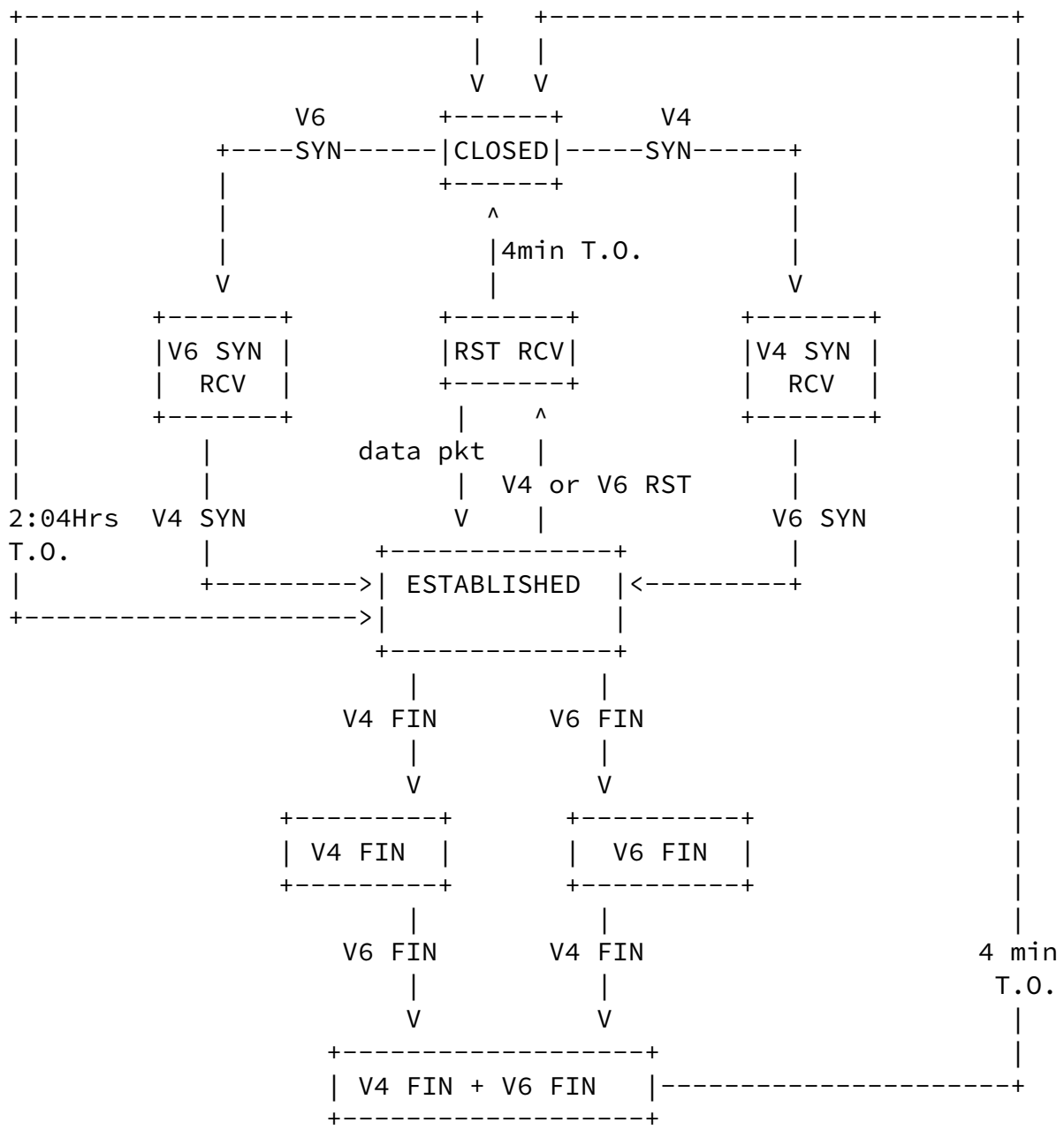
V6 FIN + V4 FIN RCV: Both an IPv4 packet containing a TCP FIN and an IPv6 packet containing an TCP FIN for this connection were received by the NAT64. The NAT64 keeps the connection state alive

and forwards packet in both directions for a short period of time to allow remaining packets (in particular the ACKs) to be delivered.

RST RCV: A packet containing a TCP RST was received by the NAT64 for this connection. The NAT64 will keep the state for the connection for a short time and if no other data packets for that connection are received, the assumption is that the node has accepted the RST packet and the state for this connection is then terminated.

The state machine used by the NAT64 for the TCP session processing is depicted next. The described state machine handles all TCP segments received through the IPv6 and IPv4 interface. There is one state machine per TCP connection that is potentially established through the NAT64. After bootstrapping of the NAT64 device, all TCP session are in CLOSED state. As we mention above, the CLOSED state is a fictional state when is no state for that particular connection in the NAT64. It should be noted that there is one state machine per connection, so only packets belonging to a given connection are inputs to the state machine associated to that connection. In other words, when in the state machine below we state that a packet is received, it is implicit that the incoming 5-tuple of the data packet matches to the one of the state machine.

A TCP segment with the SYN flag set that is received through the IPv6 interface is called a V6 SYN, similarly, V4 SYN, V4 FIN, V6 FIN, V6 FIN + V4 FIN, V6 RST and V4 RST.




We next describe the state information and the transitions.

*** CLOSED ***

If a V6 SYN is received with an incoming tuple with source transport

address (X',x) and destination transport address (Y',y) (This is the case of a TCP connection initiated from the IPv6 side), the processing is as follows:

1. The NAT64 searches for a TCP BIB entry that matches the IPv6 source transport address (X',x).





If such an entry does not exists, a new BIB entry is created. The BIB IPv6 transport address is set to (X',x) (i.e. the source IPv6 transport address of the packet). The BIB IPv4 transport address is set to an IPv4 transport address allocated using the rules defined in Section 3.2.2.1 The processing of the packet continues as described in bullet 2.

If the entry already exists, then the processing continues as described bullet 2.

2. Then a new TCP session entry is created in the TCP session table. The information included in the session table is as follows:

   The STE source IPv6 transport address is set to (X',x) (i.e. the source transport address contained in the received V6 SYN packet,

   The STE destination IPv6 transport address is set to (Y',y) (i.e. the destination transport address contained in the received V6 SYN packet,

   The STE source IPv4 transport address is set to the BIB IPv4 transport address of the corresponding TCP BIB entry.

   The STE destination IPv4 transport address contains the port y (i.e. the same port as the IPv6 destination transport address) and the IPv4 address that is algorithmically generated from the IPv6 destination address (i.e. Y') using the reverse algorithm as specified in Section 3.2.4.

   The lifetime of the TCP session table entry is set to at least to 4 min (the transitory connection idle timeout as defined in [RFC5382]).

3. The state of the session is moved to V6 SYN RCV.

4. The NAT64 translates and forwards the packet as described in the following sections

If a V4 SYN packet is received with an incoming tuple with source IPv4 transport address (Y,y) and destination IPv4 transport address (X,x) (This is the case of a TCP connection initiated from the IPv4 side), the processing is as follows:

If the security policy requires silently dropping externally initiated TCP connections, then the packet is silently discarded, else,





If the destination transport address contained in the incoming V4 SYN (i.e. X,x) is not in use in the TCP BIB, then the packet is discarded and an ICMP Port Unreachable error (Type 3, Code 3) is sent back to the source of the v4 SYN. The state remains unchanged in CLOSED

If the destination transport address contained in the incoming V4 SYN (i.e. X,x) is in use in the TCP BIB, then

A new session table entry is created in the TCP session table, containing the following information:

The STE source IPv4 transport address is set to (X,x) (i.e. the destination transport address contained in the V4 SYN)

The STE destination IPv4 transport address is set to (Y,y) (i.e. the source transport address contained in the V4 SYN)

The STE transport IPv6 source address is set to the IPv6 transport address contained in the corresponding TCP BIB entry.

The STE destination IPv6 transport address contains the port y (i.e. the same port than the destination IPv4 transport address) and the IPv6 representation of Y (i.e. the IPv4 address of the destination IPv4 transport address), generated using the algorithm described in Section 3.2.4.

The lifetime of the entry is set to 6 seconds as per [RFC5382].

The state is moved to V4 SYN RCV.

If the NAT64 is performing Address-Dependent filtering, the packet is discarded (The motivation for creating the session table entry and discarding the packet (instead of simply dropping the packet based on the filtering) is to support simultaneous open of TCP connections).

If the NAT64 is not performing Address-Dependent filtering, it translates and forwards the packet as described in the following sections.

For any other packet belonging to this connection,

If there is a corresponding entry in the TCP BIB other packets SHOULD be forwarded if the security policy allows to do so. The state remains unchanged.





If there is no corresponding entry in the TCP BIB the packet is silently discarded.

*** V4 SYN RCV ***

If a V6 SYN is received with incoming tuple with source transport address (X',x) and destination transport address (Y',y), then the lifetime of the corresponding TCP session table entry is set to at least 2 hours 4 min (the established connection idle timeout as defined in [RFC5382]). The packet is translated and forwarded. The state is moved to ESTABLISHED.

If the lifetime expires, the session table entry is deleted and, the state is moved to CLOSED.

For any other packet, other packets SHOULD be forwarded if the security policy allows to do so. The state remains unchanged.

*** V6 SYN RCV ***

If a V4 SYN is received (with or without the ACK flag set), with an

incoming tuple with source IPv4 transport address (Y,y) and destination IPv4 transport address (X,x), then the state is moved to ESTABLISHED. The timer is set to at least 2 hours 4 min (the established connection idle timeout as defined in [RFC5382]). The packet is translated and forwarded.

If the lifetime expires, the session table entry is deleted and the state is moved to CLOSED.

For any other packet, other packets SHOULD be forwarded if the security policy allows to do so. The state remains unchanged.

*** ESTABLISHED ***

If the lifetime expires, the session table entry is deleted and the state is moved to CLOSED.

If a V4 FIN packet is received, the packet is translated and forwarded. The state is moved to V4 FIN RCV.

If a V6 FIN packet is received, the packet is translated and forwarded. The state is moved to V6 FIN RCV.

If a V4 RST or a V6 RST packet is received, the packet is translated and forwarded. The lifetime is set to 4 min and state is moved to RST RCV. (Since the NAT64 is uncertain whether the peer will accept the RST packet, instead of moving the state to CLOSED, it moves to





the RST RCV, which has a shorter lifetime. If no other packets are received for this connection during the short timer, the NAT64 assumes that the peer has accepted the RST packet and moves to CLOSED. If packet keep flowing, the NAT64 assumes that the peer has not accepted the RST packet and moves back to ESTABLISHED state.)

If any other packet is received, the packet is translated and forwarded. The lifetime is set to at least 2 hours and 4 min. The state remains unchanged as ESTABLISHED.

*** V4 FIN RCV ***

If a V6 FIN packet is received, the packet is translated and forwarded. The lifetime is set to 4 min. The state is moved to V6

FIN + V4 FIN RCV.

If any other packet is received, the packet is translated and forwarded. The lifetime is set to at least 2 hours and 4 min. The state remains unchanged as V4 FIN RCV.

If the lifetime expires, the session table entry is deleted and the state is moved to CLOSED.

*** V6 FIN RCV ***

If a V4 FIN packet is received, the packet is translated and forwarded. The lifetime is set to 4 min. The state is moved to V6 FIN + V4 FIN RCV.

If any other packet is received, the packet is translated and forwarded. The lifetime is set to at least 2 hours and 4 min. The state remains unchanged as V6 FIN RCV.

If the lifetime expires, the session table entry is deleted and the state is moved to CLOSED.

*** V6 FIN + V4 FIN RCV ***

All packets are translated and forwarded.

If the lifetime expires, the session table entry is deleted and the state is moved to CLOSED.

*** RST RCV ***

If a packet other than a RST packet is received, the lifetime is set to at least 2 hours and 4 min and the state is moved to ESTABLISHED.





If the lifetime expires, the session table entry is deleted and the state is moved to CLOSED.

### 3.2.2.1. Rules for allocation of IPv4 transport addresses for TCP

If the rules specify that a new TCP 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 of the IPv4 pool assigned to the NAT64 to be used for translation.

If the port s is in the Well-Known port range 0..1023, and the NAT64 has an available port t in the same port range, then the NAT64 SHOULD allocate the port t. If the NAT64 does not have a port available in the same range, the NAT64 SHOULD assign a port t from other range where it has an available port.

If the port s is in the range 1024..65535, and the NAT64 has an available port t in the same port range, then the NAT64 SHOULD allocate the port t. If the NAT64 does not have a port available in the same range, the NAT64 SHOULD assign a port t from other range where it has an available port.

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 (referring to the UDP and TCP BIBs).

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. The NAT64 MAY send an ICMPv6 Destination Unreachable/ Address unreachable (Code 3) message.

### 3.2.3. ICMP Query Session Handling

The state information stored for an ICMP Query session in the ICMP Query session table includes a timer that tracks the remaining lifetime of the session. When the timer expires, the session is deleted. If all the sessions corresponding to a ICMP Query BIB entry are deleted, then the ICMP Query BIB entry is also deleted in the case of dynamically created entries.

An incoming ICMPv6 Informational packet with IPv6 source address X', IPv6 destination address Y' and Identifier I1, is processed as follows:

If the local security policy determines that ICMPv6 Informative packets are to be filtered, the packet is silently discarded. Else, the NAT64 searches for a ICMP Query BIB entry that matches the (X',I1) pair. If such entry does not exist, a new entry is created with the following data:

The BIB IPv6 address is set to X' i.e. the source IPv6 address of the IPv6 packet.

The BIB ICMPv6 Query Id is set to I1 i.e. the ICMPv6 Query Identifier.

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

As the BIB ICMPv4 Identifier use any available value i.e. any identifier value for which no other entry exists with the same (IPv4 address, ICMPv4 Query Id) pair.

The NAT64 searches for an ICMP query session table entry corresponding to the incoming 3-tuple (X',Y',I1). If no such entry is found, a new entry is created. The information included in the new session table entry is as follows:

The STE IPv6 source address is set to the X' i.e. the address contained in the received IPv6 packet,

The STE IPv6 destination address is set to the Y' i.e. the address contained in the received IPv6 packet,

The STE IPv6 identifier is set to the I1 I.e. the identifier contained in the received IPv6 packet,

The STE IPv4 source address is set to the IPv4 address contained in the corresponding BIB entry,

The STE IPv4 identifier is set to the IPv4 identifier contained in the corresponding BIB entry,

The STE IPv4 destination address is algorithmically generated from Y' using the reverse algorithm as specified in Section 3.2.4.

The NAT64 sets or resets the timer in the session table entry to maximum session lifetime. By default, the maximum session lifetime is 60 seconds. The maximum lifetime value SHOULD be

configurable. The packet is translated and forwarded as described in the following sections.

An incoming ICMPv4 Query packet with source IPv4 address Y, destination IPv4 address X and Identifier I2 is processed as follows:

The NAT64 searches for a ICMP Query BIB entry that contains X as IPv4 address matches and I2 as the IPv4 Identifier. If such an entry does not exists, the packet is dropped. 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).

If 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 STE source IPv4 address equal to X, an STE IPv4 Identifier equal to I2 and a STE destination IPv4 address equal to Y. 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).

In case the packet is not discarded in the previous processing (either because the NAT64 is not filtering or because the packet is compliant with the Address-dependent filtering rule), then the NAT64 searches for a session table entry with a STE source IPv4 address equal to X, an STE IPv4 Identifier equal to I2 and a STE destination IPv4 address equal to Y. If no such entry is found, a new entry is created with the following information:

The STE source IPv4 address is set to X,

The STE IPv4 Identifier is set to I2,

The STE destination IPv4 address is set to Y,

The STE source IPv6 address is set to the IPv6 address of the corresponding BIB entry,

The STE IPv6 Identifier is set to the IPv6 Identifier of the corresponding BIB entry, and,

The STE destination IPv6 address is set to the IPv6 representation of the IPv4 address of Y, generated using the





algorithm described in Section 3.2.4.

The NAT64 sets or resets the timer in the session table entry to maximum session lifetime. By default, the maximum session lifetime is 60 seconds. The maximum lifetime value SHOULD be configurable. The packet is translated and forwarded as described in the following sections.

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

NAT64 support multiple algorithms for the generation of the IPv6 representation of an IPv4 address. The constraints imposed to the generation algorithms are the following:

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

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

The NAT64 to create the IPv6 address to be included in the destination address field of the outgoing IPv6 packets from the IPv4 address included in the destination address field of the incoming IPv4 packet.

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

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

If we note n the length of the prefix Pref64::/n, then n MUST

the less or equal than 96. If a Pref64::/n is configured through any means in the DNS64 (such as manually configured, or other automatic mean not specified in this document), the default algorithm MUST use this prefix. If no prefix is available, the algorithm SHOULD use the Well-Known prefix (64:FF9B::/96) defined in [I-D.ietf-behave-address-format]

NAT64 MUST support the algorithm for generating IPv6 representations of IPv4 addresses defined in section 2.1 of [I-D.ietf-behave-address-format]. The aforementioned algorithm SHOULD be used as default algorithm.





## 3.3. Computing the Outgoing Tuple

This step computes the outgoing tuple by translating the addresses and ports or ICMP Query Id in the incoming tuple.

In the text below, a reference to the the "BIB" means either the TCP BIB the UDP BIB or the ICMP Query BIB as appropriate.

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 applying the algorithmic transformation described in Section 3.2.4. This distinction is important; without it, hairpinning doesn't work correctly.

### 3.3.1. Computing the outgoing 5-tuple for TCP and UDP.

The transport protocol in the outgoing 5-tuple is always the same as that in the incoming 5-tuple.

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: the BIB contains a entry (S',s) <--> (T,t), then the outgoing source transport address is (T,t).

The outgoing destination address is computed algorithmically from D' using the address transformation described in Section 3.2.4.

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 generated from S using the address transformation algorithm described in Section 3.2.4.

The BIB table is searched for an entry (X',x) <--> (D,d), and the outgoing destination transport address is set to (X',x).





### 3.3.2. Computing the outgoing 3-tuple for ICMP Query messages

When translating in the IPv6 --> IPv4 direction, let the incoming source and destination addresses in the 3-tuple be S' and D' respectively and the ICMPv6 Query Identifier be I1. The outgoing source address is computed as follows: the BIB contains a entry (S',I1) <--> (T,I2), then the outgoing source address is T and the ICMPv4 Query Id is I2.

The outgoing IPv4 destination address is computed algorithmically from D' using the address transformation described in Section 3.2.4.

When translating in the IPv4 --> IPv6 direction, let the incoming source and destination addresses in the 3-tuple be S and D respectively and the ICMPv4 query Id is I2. The outgoing source address is generated from S using the address transformation algorithm described in Section 3.2.4. The BIB is searched for an entry containing (X',I1) <--> (D,I2) and the outgoing destination address is X' and the outgoing ICMPv6 Query Id is I1.

## 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 IP/ICMP Translation Algorithm [I-D.ietf-behave-v6v4-xlate], 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 tuple as determined in Section 3.3.

- 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 and it is an ICMP Query message, the ICMP query Identifier is set to the one of the outgoing 3-tuple as determined in Section 3.3.2.

- 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).

The size of outgoing packets as well and the potential need for fragmentation is done according to the behavior defined in IP/ICMP Translation Algorithm [I-D.ietf-behave-v6v4-xlate]

## 3.5. Handling Hairpinning

This step handles hairpinning if necessary. A NAT64 that forwards packets originating from an IPv6 address, destined for an IPv4 address that matches the active mapping for another IPv6 address, back to that IPv6 address are defined as as supporting "hairpinning".

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 Pref64::/n 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. Filtering and updating binding and session information described in Section 3.2

# 4. 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]. The actual extensions to support IPsec are out of the scope of this document.

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.

Avoiding hairpinning loops

If the IPv6-only client can guess the IPv4 binding address that will be created, it can use the IPv6 representation of it as source address for creating this binding. Then any packet sent to the binding's IPv4 address will loop in the NAT64.

Consider the following example:

Suppose that the IPv4 pool is 192.0.2.0/24

Then the IPv6-only client sends this to NAT64:

Source: [Pref64::192.0.2.1]:500

Destination: whatever

The NAT64 allocates 192.0.2.1:500 as IPv4 binding address. Now anything sent to 192.0.2.1:500, be it a hairpinned IPv6 packet or an IPv4 packet, will loop.

It should be noted that it is not hard to guess the IPv4 address that will be allocated. First the attacker creates a binding and use e.g. STUN to know your external IPv4. New bindings will always have this address. Then it uses a source port in the range 1-1023. This will increase your chances to 1/512 (since range and parity must be preserved).

In order to address this vulnerability, the NAT64 drops IPv6 packets whose source address is in Pref64::/n.

## 5. IANA Considerations

This document contains no IANA considerations.

## 6. Contributors

George Tsirtsis

Qualcomm

tsirtsis@googlemail.com

Greg Lebovitz

Juniper

gregory.ietf@gmail.com

Simon Parreault

Viagenie

simon.perreault@viagenie.ca

## 7. Acknowledgements

Dave Thaler, Dan Wing, Alberto Garcia-Martinez, Reinaldo Penno, Ranjana Rao, Lars Eggert, Senthil Sivakumar, Zhen Cao and Joao Damas reviewed the document and provided useful comments to improve it.

The content of the draft was improved thanks to discussions with Christian Huitema, 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.

## 8. References

### 8.1. 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.

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

[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.ietf-behave-v6v4-xlate]
Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm", draft-ietf-behave-v6v4-xlate-04 (work in progress), November 2009.

[I-D.ietf-behave-address-format]
Huitema, C., Bao, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", draft-ietf-behave-address-format-02 (work in progress), December 2009.

## 8.2. Informative References

[I-D.ietf-behave-dns64]
Bagnulo, M., Sullivan, A., Matthews, P., and I. Beijnum, "DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", draft-ietf-behave-dns64-03 (work in progress), December 2009.

[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.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, July 2007.

[I-D.ietf-behave-v6v4-framework]
Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", draft-ietf-behave-v6v4-framework-03 (work in progress), October 2009.

## Appendix A. Application scenarios

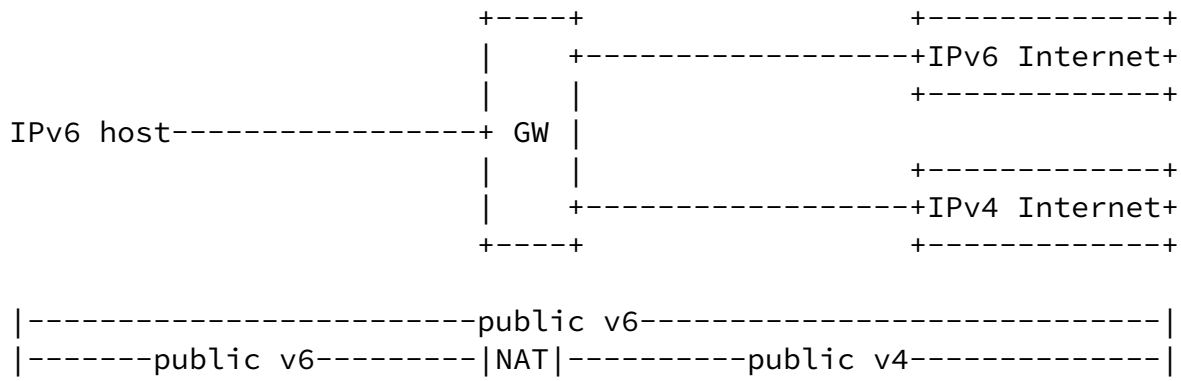
In this section, we describe how to apply NAT64/DNS64 to the suitable scenarios described in [I-D.ietf-behave-v6v4-framework] .

## A.1. Scenario 1: an IPv6 network to the IPv4 Internet

An 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.





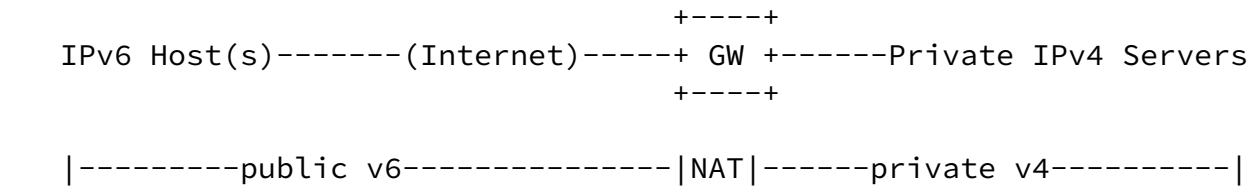



The proposed NAT64/DNS64 is perfectly suitable for this particular scenario. The deployment of the NAT64/DNS64 would be as follows: The NAT64 function should be located in the GW device that connects the IPv6 site to the IPv4 Internet. The DNS64 functionality can be placed either in the local recursive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

## A.2. Scenario 3: the IPv6 Internet to an IPv4 network

The scenario of servers using 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 even may not 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.ietf-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.

## 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
Alcatel-Lucent
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