v6ops
Internet-Draft
Obsoletes: 6146 (if approved)
Intended status: Standards Track
Expires: 2 December 2026

M. Bagnulo
UC3M
P. Matthews
J. Palet Martinez, Ed.
The IPv6 Company
31 May 2026

Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers draft-ietf-v6ops-rfc6146-bis-05

Abstract

This document describes stateful NAT64 translation, which allows IPv6-only clients to contact IPv4-only servers/peers using unicast UDP, TCP, or ICMP. One or more public IPv4 addresses assigned to a NAT64 translator are shared among several IPv6-only clients. When stateful NAT64 is used in conjunction with DNS64, no changes are required in either the IPv6 client or the IPv4 server/peer.

At the time of writing this document, Stateful NAT64 has been widely and successfully deployed across many networks in the Internet.

This document obsoletes RFC6146.

Status of This Memo

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

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

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

This Internet-Draft will expire on 2 December 2026.

Copyright Notice

Table of Contents

1. Introduction

This document specifies stateful NAT64, a mechanism for IPv4-IPv6 transition and coexistence. Together with DNS64 [RFC6147], these two mechanisms allow an IPv6-only client to initiate communications to an IPv4-only server/peer. They also enable peer-to-peer communication between an IPv4 and an IPv6 node, where the communication can be initiated when either end uses existing, NAT-traversal, peer-to-peer communication techniques, such as Interactive Connectivity Establishment (ICE) [RFC8445] [RFC8839]. Stateful NAT64 also supports IPv4-initiated communications to a subset of the IPv6 hosts through statically configured bindings in the stateful NAT64, which is commonly implemented following the same approach as for Explicit Address Mappings for Stateless IP/ICMP Translation [RFC7757].

Stateful 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 the IP/ICMP Translation Algorithm defined in [RFC7915]. The IPv4 addresses of IPv4 hosts are algorithmically translated to and from IPv6 addresses by using the algorithm defined in [RFC6052] and an IPv6 prefix assigned to the stateful NAT64 for this specific purpose. The IPv6 addresses of IPv6 hosts are translated to and from IPv4 addresses by installing mappings in the normal Network Address Port Translation (NAPT) manner [RFC3022].

This specification only defines how stateful NAT64 translates unicast packets carrying TCP, UDP, and ICMP traffic. Multicast packets and other protocols, including the Stream Control Transmission Protocol (SCTP), the Datagram Congestion Control Protocol (DCCP), and IPsec, are out of the scope of this specification.

DNS64 [RFC6147] is a mechanism for synthesizing AAAA resource records (RRs) from A RRs. The IPv6 address contained in the synthetic AAAA RR is algorithmically generated from the IPv4 address and the IPv6 prefix assigned to a NAT64 device by using the same algorithm defined in [RFC6052]. This DNS64 synthesis can also be done in the IPv6 clients (self-synthesis).

Together, these two mechanisms allow any of the following:

* an IPv6-only-strict client (a host with a networking stack that only implements/uses IPv6)

* a dual-stack client connected to an IPv6-only segment/network

* a dual-stack client willing to use only IPv6 connectivity (IPv6-Mostly [RFC8925] case)

* a host running an IPv6-only application

to initiate communications to an IPv4-only server/peer. Note that in some cases, when using NAT64 together with other protocols (such as a 464XLAT [RFC6877] CLAT), this may be possible even just using NAT64, without the need of DNS64. However, as described in [RFC8683] it has the impact of forcing a double translation and has implications in extra delay for the connection establishment.

Across the rest of this document, for short, IPv6/IPv6-only client or node, unless explicitly stated, will apply for any of the cases enumerated in the preceding paragraph.

These mechanisms are playing a critical role in IPv4-IPv6 transition and coexistence. As a result of IPv4 address depletion and the limited size of [RFC1918] addressing space (in hyperscale data centers), more new clients are IPv6-only or IPv6-only connected and they may still need to connect to the existing IPv4-only servers/ peers. This is also combined with the additional complexity and management cost of dual-stack, which increases the desire for IPv6-Only and IPv6-Mostly solutions. The stateful NAT64 and DNS64 mechanisms are easily deployable, since they do not require changes to either the IPv6 client or the IPv4 server. For basic functionality, the approach only requires the deployment of the stateful NAT64 function somewhere in the path between the devices

connecting an IPv6-only network to the IPv4-only network, along with the deployment of a DNS64-enabled name server accessible to the IPv6-only hosts. If the host can be updated, then the DNS64 functionality can be built-in, as well as supporting new features which improves the functionality, such as the support, for example, of IPv6-Mostly. An analysis of the application scenarios can be found in [RFC6144].

For brevity, in the rest of the document, we will refer to the stateful NAT64 either as stateful NAT64 or simply as NAT64.

This document obsoletes [RFC6146]. Main changes are listed in Section 12.

It has to be remarked, that since the original NAT64 specification was published, it has been widely deployed across many networks, with great success.

1.1. Features of Stateful NAT64

The features of NAT64 are:

* NAT64 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 current NAT traversal techniques, such as ICE [RFC8445] [RFC8839], and with other NAT traversal techniques.

* In the absence of preexisting state in a NAT64, only IPv6 nodes can initiate sessions to IPv4 nodes. This works for roughly the same class of applications that work through IPv4-to-IPv4 NATs (NAT44).

* Depending on the filtering policy used (Endpoint-Independent, or Address-Dependent), IPv4-nodes might be able to initiate sessions to a given IPv6 node, if the NAT64 has an appropriate mapping (i.e., state) for an IPv6 node, via one of the following mechanisms:

  - The IPv6 node has recently initiated a session to the same or another IPv4 node. This is also the case if the IPv6 node has used a NAT-traversal technique (such as ICE).

  - A statically configured mapping exists for the IPv6 node, e.g. by means of [RFC7757].

* IPv4 address sharing: NAT64 allows multiple IPv6-only nodes to share a single IPv4 address to access the IPv4 Internet. This helps with the IPv4 exhaustion. The NAT64 can even be operated as a service by other parties, not necessarily the service provider providing the Internet access.

* As currently defined in this NAT64 specification, only TCP, UDP, and ICMP are supported. Support for other protocols (such as other transport protocols and IPsec without UDP encapsulation [RFC3948]) may be defined in separate documents.

1.2. Overview

This section provides a non-normative introduction to NAT64. This is achieved by describing the NAT64 behavior involving a simple setup that involves a single NAT64 device, a single DNS64, 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. A more complete set of possible deployment scenarios is described in [RFC8683] So, for the sake of clarity, only TCP and UDP are described in this overview; the details of ICMP, fragmentation, and other aspects of translation are purposefully avoided in this overview. The normative specification of NAT64 is provided in Section 3.

The NAT64 mechanism is implemented in a device that has (at least) two interfaces, an IPv4 interface connected to 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 device. The NAT64 will translate them and forward them as IPv4 packets through the IPv4 network to the IPv4 receiver. The reverse takes place for packets generated by hosts connected to 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. The state contains the binding of an IPv6 address and TCP/UDP port (hereafter called an IPv6 transport address) to an IPv4 address and TCP/UDP 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 both directions 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 an IPv4-only node to an IPv6-only node.

1.2.1. Stateful 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 an IPv4 packet header to an IPv6 packet header and vice versa is performed according to the IP/ICMP Translation Algorithm [RFC7915].

Address translation maps IPv6 transport addresses to IPv4 transport addresses and vice versa. In order to create these mappings, the NAT64 has two pools of addresses: 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 one or more IPv6 prefixes assigned to the translator itself. Hereafter, we will call the IPv6 address pool Pref64::/n; in the case there is more than one prefix assigned to the NAT64, the comments made about Pref64::/n apply to each of them. Pref64::/n will be used by the NAT64 to construct IPv4-Converted IPv6 addresses as defined in [RFC6052]. Due to the abundance of IPv6 address space, it is possible to assign one or more Pref64::/n, each of them being equal to or even bigger than the size of the whole IPv4 address space. This allows each IPv4 address to be mapped into a different IPv6 address by simply concatenating a Pref64::/n with the IPv4 address being mapped and a suffix. The provisioning of the Pref64::/n as well as the address format are defined in [RFC6052].

The IPv4 address pool is a set of IPv4 addresses, normally a 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 binding of IPv6 transport addresses into IPv4 transport addresses, instead of IPv6 addresses into IPv4 addresses directly, enabling a higher utilization of the limited IPv4 address pool. This implies that NAT64 performs both address and port translation.

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 far simpler to allow communications initiated from the IPv6 side toward an IPv4 node, whose address is algorithmically mapped into an IPv6 address, than communications initiated from IPv4-only nodes to an IPv6 node. In that case, an IPv4 address needs to be associated with the IPv6 node’s address dynamically.

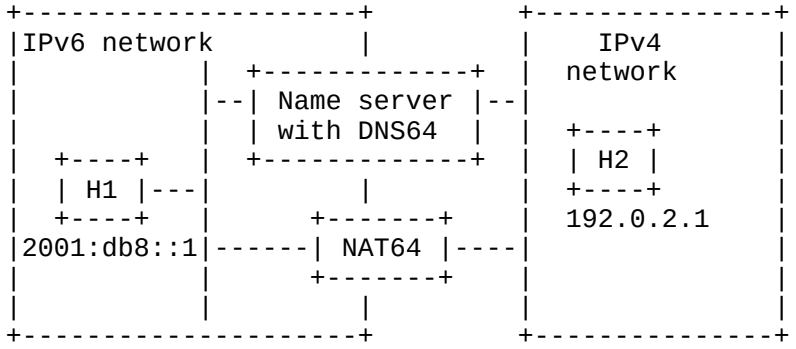
Using a mechanism such as DNS64, an IPv6 client obtains an IPv6 address that embeds the IPv4 address of the IPv4 server and sends a packet to that IPv6 address. The packets are intercepted by the NAT64 device, which associates an IPv4 transport address out of its IPv4 pool to the IPv6 transport address 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 a DNS lookup to learn the address of the responder, DNS64 [RFC6147] is used to synthesize AAAA RRs from the A RRs. The IPv6 addresses contained in the synthetic AAAA RRs contain a Pref64::/n assigned to the NAT64 and the IPv4 address of the responder. The synthetic AAAA RRs are passed back to the IPv6 initiator, which will initiate an IPv6 communication with an 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. Stateful NAT64 Behavior Walk-Through

In this section, we provide a simple example of the NAT64 behavior. We consider an IPv6 node that is located in an IPv6-only site and that initiates a TCP connection to an IPv4-only node located in the IPv4 network.

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



The figure above shows an IPv6 node H1 with an IPv6 address 2001:db8::1 and an IPv4 node H2 with IPv4 address 192.0.2.1. H2 has h2.example.com as its Fully Qualified Domain Name (FQDN).

A NAT64 connects the IPv6 network to the IPv4 network. This NAT64 uses the Well-Known Prefix 64:ff9b::/96 defined in [RFC6052] to represent IPv4 addresses in the IPv6 address space and a single IPv4 address 203.0.113.1 assigned to its IPv4 interface. The routing is configured in such a way that the IPv6 packets addressed to a destination address in 64:ff9b::/96 are routed to the IPv6 interface of the NAT64 device.

Also shown is a local name server with DNS64 functionality. The local name server uses the Well-Known Prefix 64:ff9b::/96 to create the IPv6 addresses in the synthetic RRs.

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 h2.example.com 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 Well-Known Prefix and the IPv4 address of H2 (i.e., 64:ff9b::192.0.2.1).

2. H1 sends a TCP SYN packet to H2. The packet is sent from a source transport address of (2001:db8::1,1500) to a destination transport address of (64:ff9b::192.0.2.1,80), where the ports are set by H1.

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

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

   * The NAT64 selects an unused port (e.g., 2000) on its IPv4 address 203.0.113.1 and creates the mapping entry (2001:db8::1,1500) <--> (203.0.113.1,2000)

   * The NAT64 translates the IPv6 header into an IPv4 header using the IP/ICMP Translation Algorithm [RFC7915].

   * The NAT64 includes (203.0.113.1,2000) as the source transport address in the packet and (192.0.2.1,80) as the destination transport address in the packet. Note that 192.0.2.1 is extracted directly from the destination IPv6 address of the received IPv6 packet that is being translated. The destination port 80 of the translated packet is the same as the destination port of the received IPv6 packet.

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

6. H2 node responds by sending a TCP SYN+ACK packet with the destination transport address (203.0.113.1,2000) and source transport address (192.0.2.1,80).

7. Since the IPv4 address 203.0.113.1 is assigned to the IPv4 interface of the NAT64 device, the packet is routed to the NAT64 device, which will look for an existing mapping containing (203.0.113.1,2000). Since the mapping (2001:db8::1,1500) <--> (203.0.113.1,2000) exists, the NAT64 performs the following operations:

   * The NAT64 translates the IPv4 header into an IPv6 header using the IP/ICMP Translation Algorithm [RFC7915].

   * The NAT64 includes (2001:db8::1,1500) as the destination transport address in the packet and (64:ff9b::192.0.2.1,80) as the source transport address in the packet. Note that 192.0.2.1 is extracted directly from the source IPv4 address of the received IPv4 packet that is being translated. The source port 80 of the translated packet is the same as the source port of the received IPv4 packet.

8. The translated packet is sent out of 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., 64:ff9b::192.0.2.1) through some scheme other than a DNS lookup. This is because the DNS64 processing does NOT result in any state being installed in the NAT64 and because the mapping of the IPv4 address into an IPv6 address is the result of concatenating the Well-Known Prefix to the original IPv4 address.

1.2.3. Filtering

NAT64 may do filtering, which means that it only allows a packet in through an interface under certain circumstances. The NAT64 can filter IPv6 packets based on the administrative rules to create entries in the binding and session tables. The filtering can be flexible and general, but the idea of the filtering is to provide the administrators necessary control to avoid denial-of-service (DoS) attacks that would result in exhaustion of the NAT64's IPv4 address, port, memory, and CPU resources. Filtering techniques of incoming IPv6 packets are not specific to the NAT64 and therefore are 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 [RFC4787] and [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 as well as filtering of ICMP packets.

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 this document.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC 2119 [RFC2119] RFC 8174 [RFC8174].

The following additional terms are used in this document:

3-Tuple: The tuple (source IP address, destination IP address, ICMP 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 bindings kept by a NAT64. Each NAT64 has a BIB for each translated protocol. An implementation compliant to this document would have a BIB for TCP, one for UDP, and one for ICMP Queries. Additional BIBs would be added to support other protocols, such as SCTP.

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 only filters incoming IPv4 packets destined to a transport address for which there is no 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 incoming IPv4

packets 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 a given IPv4 address X and destined for a transport address for which it has state 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 side as it arrived on. If the destination IPv6 address and its embedded IPv4 address are both assigned to the NAT64 itself, then the packet is being sent to another IPv6 host connected to the same NAT64. Such a packet is called a 'hairpin packet'. A NAT64 that forwards hairpin packets back to the IPv6 host is defined as supporting "hairpinning". 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. Hairpin packets can be either TCP or UDP. More detailed explanation of hairpinning and examples for the UDP case can be found in [RFC4787].

ICMP Query packet: ICMP packets that are not ICMP error messages. For ICMPv6, ICMPv6 Query Messages are the ICMPv6 Informational messages as defined in [RFC4443]. For ICMPv4, ICMPv4 Query messages are all ICMPv4 messages that are not ICMPv4 error messages.

Mapping or Binding: A mapping between an IPv6 transport address and a IPv4 transport address or a mapping between an (IPv6 address, ICMPv6 Identifier) pair and an (IPv4 address, ICMPv4 Identifier) pair. Used to translate the addresses and ports / ICMP Identifiers of packets flowing between the IPv6 host and the IPv4 host. In NAT64, the IPv4 address and port / ICMPv4 Identifier is always one assigned to the NAT64 itself, while the IPv6 address and port / ICMPv6 Identifier belongs to some IPv6 host.

Session: The flow of packets between two different hosts. This may be TCP, UDP, or ICMP Queries. In NAT64, typically one host is an IPv4 host, and the other one is an IPv6 host. However, due to hairpinning, both hosts might be IPv6 hosts.

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.

Stateful NAT64: A function that has per-flow state that translates IPv6 packets to IPv4 packets and vice versa, for TCP, UDP, and ICMP. The NAT64 uses binding state to perform the translation between IPv6 and IPv4 addresses. In this document, we also refer to stateful NAT64 simply as NAT64.

Stateful NAT64 device: The device where the NAT64 function is executed. In this document, we also refer to stateful NAT64 device simply as NAT64 device.

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 NAT terminology [RFC4787].

3. Stateful 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 at least one unicast /n IPv6 prefix assigned to it, denoted Pref64::/n. Additional considerations about the Pref64::/n are presented in Section 3.5.4. A NAT64 MUST have one or more unicast IPv4 addresses assigned to it.

A NAT64 uses the following conceptual 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 behavior. NAT64 implementations are free to use different data structures but they MUST store all the required information, and the externally visible outcome MUST be the same as the one described in this document.

The notation used is the following: uppercase letters are IPv4 addresses; uppercase letters with a prime(') are IPv6 addresses; lowercase letters are ports; IPv6 prefixes of length n are indicated by "P::/n"; mappings are indicated as "(X,x) <--> (Y',y)".

3.1. Binding Information Bases

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, 49832) 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. If the BIBs are implemented as specified in this document, it results in 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 specifies a mapping between an (IPv6 address, ICMPv6 Identifier) pair and an (IPv4 address, ICMPv4 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 Identifier. T will always be one of the IPv4 addresses assigned to the NAT64. A given (IPv6 or IPv4 address, ICMPv6 or ICMPv4 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 Section 3.5, but they can also be created manually by an 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 with 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.

3.2. Session Tables

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 addresses and a pair of IPv4 transport addresses:

(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 TCP or UDP Session Table Entry (STE), there are then five columns. The terminology used for the STE columns is from the perspective of an incoming IPv6 packet being translated into an outgoing IPv4 packet. The columns are:

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.

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

The STE lifetime.

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

(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 ICMP Query Session Table Entry (STE), there are then seven columns:

The STE source IPv6 address; X' in the example above.

The STE destination IPv6 address; Y' in the example above.

The STE ICMPv6 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.

The STE ICMPv4 Identifier; i2 in the example above.

The STE lifetime.

3.3. Packet Processing Overview

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 Identifier (in the IPv6 --> IPv4 direction) or the destination IPv4 address and the ICMPv4 Identifier (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 IP packet in general. In the case that the processing is specific to IPv6 packets, we will explicitly refer to the incoming packet as an incoming IPv6 packet; analogous terminology will apply in the case of processing that is specific to 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, but the externally visible outcome MUST be the same as the one described in this document.

3.4. Determining the Incoming Tuple

This step associates an 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 the source IP address, source port, destination IP address, destination port, and transport protocol. In case of ICMP Queries, the tuple is a 3-tuple consisting of the source IP address, destination IP address, and ICMP 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 received packet.

If the incoming packet is a complete (un-fragmented) 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 Identifier.

If the incoming IP packet contains a complete (un-fragmented) ICMP error message containing a UDP or a TCP packet, then the incoming 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 incoming 5-tuple, the embedded source port becomes the destination port in the incoming 5-tuple, etc. If it is not possible to determine the incoming 5-tuple (perhaps because not enough of the embedded packet is reproduced inside the ICMP message), then the incoming IP packet MUST be silently discarded.

If the incoming IP packet contains a complete (un-fragmented) ICMP error message containing an ICMP error message, then the packet is silently discarded.

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

The NAT64 MUST handle fragments. In particular, NAT64 MUST handle fragments arriving out of order, conditional on the following:

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

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

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

For incoming packets carrying TCP or UDP fragments with a non-zero checksum, NAT64 MAY elect to queue the fragments as they arrive and translate all fragments at the same time. In this case, the incoming tuple is determined as documented above to the un-fragmented packets. 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, and the rules (in the bulleted list above) about how the NAT64 handles (out-of-order) fragments apply.

For incoming IPv4 packets carrying UDP packets with a zero 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 MUST silently discard the packets. The handling of fragmented and un-fragmented UDP packets with a zero checksum as specified above deviates from that specified in [RFC7915].

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].

If the incoming packet is an IPv6 packet that contains a protocol other than TCP, UDP, or ICMPv6 in the last Next Header, then the packet SHOULD be discarded and, if the security policy permits, the NAT64 SHOULD send an ICMPv6 Destination Unreachable error message with Code 4 (Port Unreachable) to the source address of the received packet. NOTE: This behavior may be updated by future documents that define how other protocols such as SCTP or DCCP are processed by NAT64.

If the incoming packet is an IPv4 packet that contains a protocol other than TCP, UDP, or ICMPv4, then the packet SHOULD be discarded and, if the security policy permits, the NAT64 SHOULD send an ICMPv4 Destination Unreachable error message with Code 2 (Protocol Unreachable) to the source address of the received packet. NOTE: This behavior may be updated by future documents that define how other protocols such as SCTP or DCCP are processed by NAT64.

3.5. 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.

The details of this step depend on the protocol, i.e., UDP, TCP, or ICMP. The behaviors for UDP, TCP, and ICMP Queries are described in Section 3.5.1, Section 3.5.2, and Section 3.5.3, respectively. For the case of ICMP error messages, they do not affect in any way either the BIBs or the session tables, so there is no processing resulting from these messages in this section. ICMP error message processing continues in Section 3.6.

Irrespective 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 Section 8. In addition, the NAT64 MUST only process incoming IPv6 packets that contain a destination address that contains Pref64::/n. Likewise, the NAT64 MUST only process incoming IPv4 packets that contain a destination address that belongs to the IPv4 pool assigned to the NAT64.

3.5.1. UDP Session Handling

The following state information is stored for a UDP session:

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

Lifetime: 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 dynamically created UDP BIB entry are deleted, then the UDP BIB entry is also deleted.

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 the BIB IPv6 transport address that matches the IPv6 source transport address (X’,x). If such an entry does not exist, the NAT64 tries to create a new entry (if resources and policy permit). The source IPv6 transport address of the packet (X’,x) is used as the BIB IPv6 transport address, and the BIB IPv4 transport address is set to (T,t), which is allocated using the rules defined in Section 3.5.1.1. 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, the NAT64 tries to create a new entry (if resources and policy permit). 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 IPv6 transport address contained in the received IPv6 packet.

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

- The STE source IPv4 transport address is extracted from the corresponding UDP BIB entry, i.e., it 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 (see Section 3.5.4).

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 the maximum session lifetime. The maximum session lifetime MAY be configurable, and the default SHOULD be at least UDP_DEFAULT. The

maximum session lifetime MUST NOT be less than UDP_MIN. 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 (W,w) and destination IPv4 transport address (T,t) is processed as follows:

The NAT64 searches for a UDP BIB entry that contains the BIB IPv4 transport address matching (T,t), i.e., the IPv4 destination transport address in the incoming IPv4 packet. If such an entry does not exist, the packet MUST be dropped. An ICMP error message with Type 3 (Destination Unreachable) MAY be sent to the original sender of the packet.

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 an STE source IPv4 transport address equal to (T,t), i.e., the destination IPv4 transport address in the incoming packet, and STE destination IPv4 address equal to W, 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 error message MAY be sent to the original sender of the packet. The ICMP error message, if sent, has Type 3 (Destination Unreachable) and Code 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 containing the STE source IPv4 transport address equal to (T,t) and the STE destination IPv4 transport address equal to (W,w). If no such entry is found, the NAT64 tries to create a new entry (if resources and policy permit). 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 (Y'(W),w), w being the same port w as the source IPv4 transport address and Y'(W) being the IPv6 representation of W, generated using the algorithm described in Section 3.5.4.

- The STE source IPv4 transport address is set to (T,t), i.e., the destination IPv4 transport addresses contained in the received IPv4 packet.

- The STE destination IPv4 transport is set to (W,w), i.e., the source IPv4 transport addresses contained in the received IPv4 packet.

The NAT64 sets (or resets) the timer in the Session Table Entry to the maximum session lifetime. The maximum session lifetime MAY be configurable, and the default SHOULD be at least UDP_DEFAULT. The maximum session lifetime MUST NOT be less than UDP_MIN. The packet is translated and forwarded as described in the following sections.

3.5.1.1. Rules for Allocation of IPv4 Transport Addresses for UDP

When a new UDP BIB entry is created for a source transport address of (S’,s), 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 the NAT64 SHOULD 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 MAY assign a port t from another range where it has an available port. (This behavior is recommended in REQ 3-a of [RFC4787].)

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 MAY assign a port t from another range where it has an available port. (This behavior is recommended in REQ 3-a of [RFC4787].)

The NAT64 SHOULD preserve the port parity (odd/even), as per Section 4.2.2 of [RFC4787]).

In all cases, the allocated IPv4 transport address (T,t) MUST NOT be in use in another entry in the same BIB, but can be in use in other BIBs (e.g., the TCP and ICMP BIBs).

If it is not possible to allocate an appropriate IPv4 transport address or create a BIB entry, then the packet is discarded. The NAT64 SHOULD send an ICMPv6 Destination Unreachable error message with Code 3 (Address Unreachable).

3.5.2. TCP Session Handling

In this section, we describe how the TCP BIB and session table are populated. We do so by defining the state machine that the NAT64 uses for TCP. We first describe the states and the information contained in them, and then we describe the actual state machine and state transitions.

3.5.2.1. State Definition

The following state information is stored for a TCP session:

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

Lifetime: 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 dynamically created TCP BIB entry is also deleted.

Because the TCP session inactivity lifetime is set to at least 2 hours and 4 minutes (as per [RFC5382]), it is important that each TCP Session Table Entry corresponds to an existing TCP session. In order to do that, for each TCP session established, the TCP connection state is tracked using the following state machine.

The states are as follows:

CLOSED: Analogous to [RFC9293], 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 INIT: 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 IPv6 packet containing the TCP SYN in the opposite direction.

V6 INIT: An IPv6 packet containing a TCP SYN was received, translated, and forwarded 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: Represents an open connection, with data able to flow 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, and 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, and 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 packets in both directions for a short period of time to allow remaining packets (in particular, the ACKs) to be delivered.

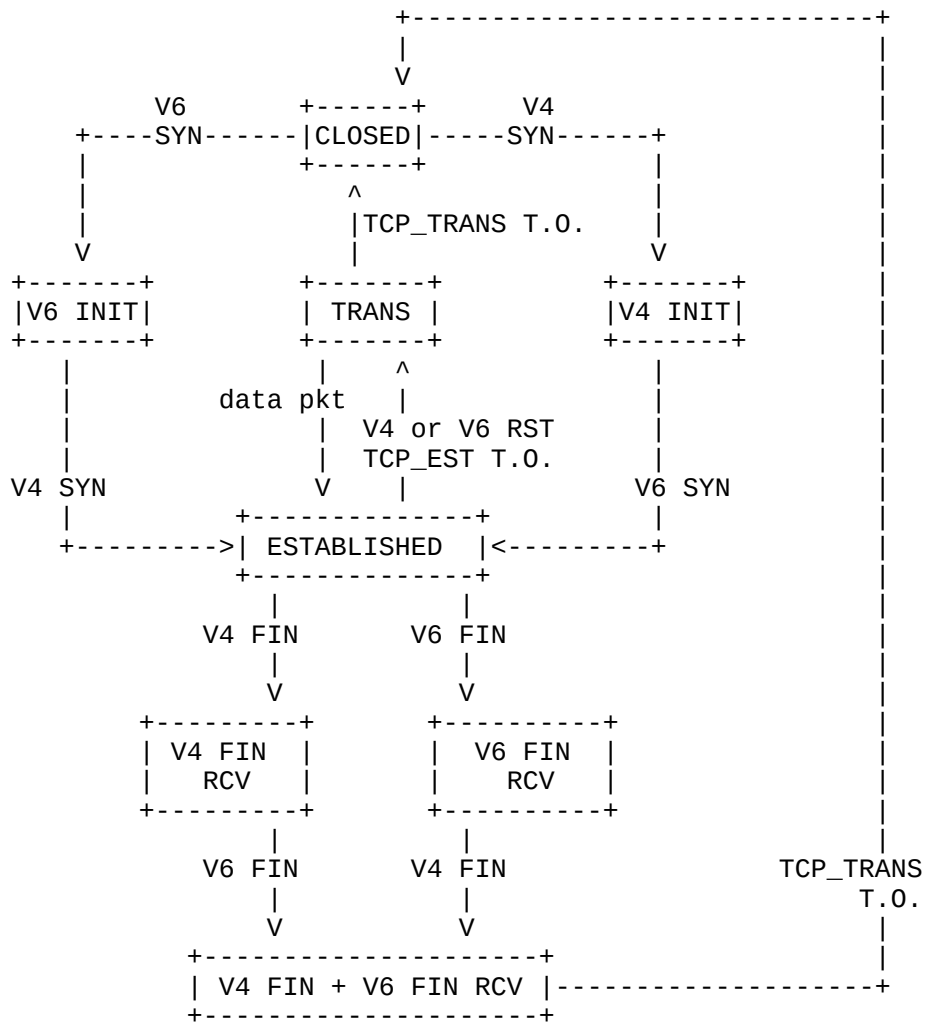
TRANS: The lifetime of the state for the connection is set to TCP_TRANS minutes either because a packet containing a TCP RST was received by the NAT64 for this connection or simply because the lifetime of the connection has decreased and there are only TCP_TRANS minutes left. The NAT64 will keep the state for the connection for TCP_TRANS minutes, and if no other data packets for that connection are received, the state for this connection is then terminated.

3.5.2.2. State Machine for TCP Processing in the NAT64

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 sessions are in CLOSED state. As we mention above, the CLOSED state is a fictional state when there 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.

The figure presents a simplified version of the state machine; refer to the text for the full specification of the state machine.



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 exist, the NAT64 tries to create a new BIB entry (if resources and policy permit). 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 3.5.2.3. The processing of the packet continues as described in bullet 2.

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

2. Then the NAT64 tries to create a new TCP session entry in the TCP session table (if resources and policy permit). 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 3.5.4.

   The lifetime of the TCP Session Table Entry is set to at least TCP_TRANS (the transitory connection idle timeout as defined in [RFC5382]).

3. The state of the session is moved to V6 INIT.

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 NAT64 tries to create a new Session Table Entry in the TCP session table (if resources and policy permit), 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 left unspecified and may be populated by other protocols that are out of the scope of this specification.

  - The STE destination IPv6 transport address contains the port y (i.e., the same port as the STE destination IPv4 transport address) and the IPv6 representation of Y (i.e., the IPv4 address of the STE destination IPv4 transport address), generated using the algorithm described in Section 3.5.4.

- The state is moved to V4 INIT.

- The lifetime of the STE entry is set to TCP_INCOMING_SYN as per [RFC5382], and the packet is stored. The result is that the NAT64 will not drop the packet based on the filtering, nor create a BIB entry. Instead, the NAT64 will only create the Session Table Entry and store the packet. The motivation for this is to support simultaneous open of TCP connections.

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

- The NAT64 tries to create a new Session Table Entry in the TCP session table (if resources and policy permit), 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 as the STE destination IPv4 transport address) and the IPv6 representation of Y (i.e., the IPv4 address of the STE destination IPv4 transport address), generated using the algorithm described in Section 3.5.4.

- The state is moved to V4 INIT.

- If the NAT64 is performing Address-Dependent Filtering, the lifetime of the STE entry is set to TCP_INCOMING_SYN as per [RFC5382], and the packet is stored. The motivation for creating the Session Table Entry and storing 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, the lifetime of the STE is set to at least TCP_TRANS (the transitory connection idle timeout as defined in [RFC5382]), and 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, the packet SHOULD be translated and forwarded if the security policy allows doing so. The state remains unchanged.

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

*** V4 INIT ***

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 TCP Session Table Entry is set to at least the maximum session lifetime. The value for the maximum session lifetime MAY be configurable, but it MUST NOT be less than TCP_EST (the established connection idle timeout as defined in [RFC5382]). The default value for the maximum session lifetime SHOULD be set to TCP_EST. The packet is translated and forwarded. The state is moved to ESTABLISHED.

If the lifetime expires, an ICMP Port Unreachable error (Type 3, Code 3) containing the IPv4 SYN packet stored is sent back to the source of the v4 SYN, the Session Table Entry is deleted, and the state is moved to CLOSED.

For any other packet, the packet SHOULD be translated and forwarded if the security policy allows doing so. The state remains unchanged.

*** V6 INIT ***

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 lifetime of the TCP Session Table Entry is set to at least the maximum session lifetime. The value for the maximum session lifetime MAY be configurable, but it MUST NOT be less than TCP_EST (the established connection idle timeout as defined in [RFC5382]). The default value for the maximum session lifetime SHOULD be set to TCP_EST. The packet is translated and forwarded.

If the lifetime expires, the Session Table Entry is deleted, and the state is moved to CLOSED.

If a V6 SYN packet is received, the packet is translated and forwarded. The lifetime of the TCP Session Table Entry is set to at least TCP_TRANS. The state remains unchanged.

For any other packet, the packet SHOULD be translated and forwarded if the security policy allows doing so. The state remains unchanged.

*** ESTABLISHED ***

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 TCP_TRANS and the state is moved to TRANS. (Since the NAT64 is uncertain whether the peer will accept the RST packet, instead of moving the state to CLOSED, it moves to TRANS, 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 packets keep flowing, the NAT64 assumes that the peer has not accepted the RST packet and moves back to the ESTABLISHED state. This is described below in the TRANS state processing description.)

If any other packet is received, the packet is translated and forwarded. The lifetime of the TCP Session Table Entry is set to at least the maximum session lifetime. The value for the maximum session lifetime MAY be configurable, but it MUST NOT be less than TCP_EST (the established connection idle timeout as defined in [RFC5382]). The default value for the maximum session lifetime SHOULD be set to TCP_EST. The state remains unchanged as ESTABLISHED.

If the lifetime expires, then the NAT64 SHOULD send a probe packet (as defined next) to at least one of the endpoints of the TCP connection. The probe packet is a TCP segment for the connection with no data. The sequence number and the acknowledgment number are set to zero. All flags but the ACK flag are set to zero. The state is moved to TRANS.

Upon the reception of this probe packet, the endpoint will reply with an ACK containing the expected sequence number for that connection. It should be noted that, for an active connection, each of these probe packets will generate one packet from each end involved in the connection, since the reply of the first point to the probe packet will generate a reply from the other endpoint.

*** V4 FIN RCV ***

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

If any packet other than the V6 FIN is received, the packet is translated and forwarded. The lifetime of the TCP Session Table Entry is set to at least the maximum session lifetime. The value for the maximum session lifetime MAY be configurable, but it MUST NOT be less than TCP_EST (the established connection idle timeout as defined in [RFC5382]). The default value for the maximum session lifetime SHOULD be set to TCP_EST. 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 TCP_TRANS. The state is moved to V6 FIN + V4 FIN RCV.

If any packet other than the V4 FIN is received, the packet is translated and forwarded. The lifetime of the TCP Session Table Entry is set to at least the maximum session lifetime. The value for the maximum session lifetime MAY be configurable, but it MUST NOT be less than TCP_EST (the established connection idle timeout as defined in [RFC5382]). The default value for the maximum session lifetime SHOULD be set to TCP_EST. 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.

*** TRANS ***

If a packet other than a RST packet is received, the lifetime of the TCP Session Table Entry is set to at least the maximum session lifetime. The value for the maximum session lifetime MAY be configurable, but it MUST NOT be less than TCP_EST (the established connection idle timeout as defined in [RFC5382]). The default value for the maximum session lifetime SHOULD be set to TCP_EST. The state is moved to ESTABLISHED.

If the lifetime expires, the Session Table Entry is deleted and the state is moved to CLOSED.

3.5.2.3. Rules for Allocation of IPv4 Transport Addresses for TCP

When a new TCP BIB entry is created for a source transport address of (S’,s), the NAT64 allocates an IPv4 transport address for this BIB entry as follows:

> If there exists some other BIB entry in any of the BIBs that contains S’ as the IPv6 address and maps it to some IPv4 address T, then T SHOULD be used 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 MAY assign a port t from another 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 MAY assign a port t from another 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 can be in use in other BIBs (e.g., the UDP and ICMP BIBs).

If it is not possible to allocate an appropriate IPv4 transport address or create a BIB entry, then the packet is discarded. The NAT64 SHOULD send an ICMPv6 Destination Unreachable error message with Code 3 (Address Unreachable).

3.5.3. ICMP Query Session Handling

The following state information is stored for an ICMP Query session in the ICMP Query session table:

Binding:(X’,Y’,i1) <--> (T,Z,i2)

Lifetime: a timer that tracks the remaining lifetime of the ICMP Query session. When the timer expires, the session is deleted. If all the ICMP Query sessions corresponding to a dynamically created ICMP Query BIB entry are deleted, then the ICMP Query BIB entry is also deleted.

An incoming ICMPv6 Informational packet with IPv6 source address X’, IPv6 destination address Y’, and ICMPv6 Identifier i1 is processed as follows:

If the local security policy determines that ICMPv6 Informational packets are to be filtered, the packet is silently discarded. Else, the NAT64 searches for an ICMP Query BIB entry that matches the (X’,i1) pair. If such an entry does not exist, the NAT64 tries to create a new entry (if resources and policy permit) 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 Identifier is set to i1 (i.e., the ICMPv6 Identifier).

- If there exists another BIB entry in any of the BIBs that contains the same IPv6 address X' and maps it to an 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.

- Any available value is used as the BIB ICMPv4 Identifier, i.e., any identifier value for which no other entry exists with the same (IPv4 address, ICMPv4 Identifier) 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, the NAT64 tries to create a new entry (if resources and policy permit). The information included in the new Session Table Entry is as follows:

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

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

- The STE ICMPv6 Identifier is set to 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 ICMPv4 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 3.5.4.

The NAT64 sets (or resets) the timer in the session table entry to the maximum session lifetime. By default, the maximum session lifetime is ICMP_DEFAULT. 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 ICMPv4 Identifier i2 is processed as follows:

The NAT64 searches for an ICMP Query BIB entry that contains X as the IPv4 address and i2 as the ICMPv4 Identifier. If such an entry does not exist, the packet is dropped. An ICMP error message MAY be sent to the original sender of the packet. The ICMP error message, if sent, has Type 3, Code 1 (Host Unreachable).

In contrast to the TCP and UDP specifications in previous sections, the definition of the ICMP Query session does not allow for a situation where multiple sessions might share T, t and Z but not z (as t and z do not exist), which means ICMP does not have an analogous Address-Dependent Filtering rule to apply at this stage.

Consequently, in case the packet is not discarded in the previous processing steps, the NAT64 then searches for a Session Table Entry with an STE source IPv4 address equal to X, an STE ICMPv4 Identifier equal to i2, and a STE destination IPv4 address equal to Y. If no such entry is found, the NAT64 tries to create a new entry (if resources and policy permit) with the following information:

- The STE source IPv4 address is set to X.
- The STE ICMPv4 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 ICMPv6 Identifier is set to the ICMPv6 Identifier of the corresponding BIB entry.
- The STE destination IPv6 address is set to the IPv6 representation of the IPv4 address of Y, generated using the algorithm described in Section 3.5.4.
- The NAT64 sets (or resets) the timer in the session table entry to the maximum session lifetime. By default, the maximum session lifetime is ICMP_DEFAULT. The maximum lifetime value SHOULD be configurable. The packet is translated and forwarded as described in the following sections.

3.5.4. Generation of the IPv6 Representations of IPv4 Addresses

NAT64 supports multiple algorithms for the generation of the IPv6 representation of an IPv4 address and vice versa. The constraints imposed on the generation algorithms are the following:

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

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

- If we note n the length of the prefix Pref64::/n, then n MUST be less than or equal to 96. If a Pref64::/n is configured through any means in the NAT64 (such as manually configured, or other automatic means not specified in this document), the default algorithm MUST use this prefix. If no prefix is available, the algorithm SHOULD use the Well-Known Prefix (64:ff9b::/96) defined in [RFC6052].

NAT64 MUST support the algorithm for generating IPv6 representations of IPv4 addresses defined in Section 2.3 of [RFC6052]. The aforementioned algorithm SHOULD be used as default algorithm.

3.6. Computing the Outgoing Tuple

This step computes the outgoing tuple by translating the IP addresses and port numbers or ICMP Identifier in the incoming tuple.

In the text below, a reference to a BIB means 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 by applying the algorithmic transformation described in Section 3.5.4. This distinction is important; without it, hairpinning doesn’t work correctly.

3.6.1. Computing the Outgoing 5-Tuple for TCP, UDP, and for ICMP Error Messages Containing a TCP or UDP Packets

The transport protocol in the outgoing 5-tuple is always the same as that in the incoming 5-tuple. When translating from IPv4 ICMP to IPv6 ICMP, the protocol number in the last next header field in the protocol chain is set to 58 (IPv6-ICMP). When translating from IPv6 ICMP to IPv4 ICMP, the protocol number in the protocol field of the IP header is set to 1 (ICMP).

When translating in the IPv6 --> IPv4 direction, let the source and destination transport addresses in the incoming 5-tuple be (S’,s) and (D’,d), respectively. The outgoing source transport address is computed as follows: if the BIB contains an 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 3.5.4.

When translating in the IPv4 --> IPv6 direction, let the source and destination transport addresses in the incoming 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 3.5.4.

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

3.6.2. Computing the Outgoing 3-Tuple for ICMP Query Messages and for ICMP Error Messages Containing an ICMP Query

When translating in the IPv6 --> IPv4 direction, let the source and destination addresses in the incoming 3-tuple be S’ and D’, respectively, and the ICMPv6 Identifier be i1. The outgoing source address is computed as follows: the BIB contains an entry (S’,i1) <--> (T,i2), then the outgoing source address is T and the ICMPv4 Identifier is i2.

The outgoing IPv4 destination address is computed algorithmically from D’ using the address transformation described in Section 3.5.4.

When translating in the IPv4 --> IPv6 direction, let the source and destination addresses in the incoming 3-tuple be S and D, respectively, and the ICMPv4 Identifier is i2. The outgoing source address is generated from S using the address transformation

algorithm described in Section 3.5.4. The BIB is searched for an entry containing (X',i1) <--> (D,i2), and, if found, the outgoing destination address is X' and the outgoing ICMPv6 Identifier is i1.

3.7. Translating the Packet

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

The translation of the packet is as specified in Sections 4 and 5 of the IP/ICMP Translation Algorithm [RFC7915], with the following modifications:

* When translating an IP header (Sections 4.1 and 5.1 of [RFC7915]), the source and destination IP address fields are set to the source and destination IP addresses from the outgoing tuple as determined in Section 3.6.

* 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 that 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 Identifier is set to the one from the outgoing 3-tuple as determined in Section 3.6.2.

* When the protocol following the IP header is ICMP 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 the IP/ICMP Translation Algorithm [RFC7915].

3.8. Handling Hairpinning

If the destination IP address of the translated packet is an IPv4 address assigned to the NAT64 itself, then the packet is a hairpin packet. Hairpin packets are processed as follows:

* The outgoing 5-tuple becomes the incoming 5-tuple.

* 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" (Section 3.5).

4. Protocol Constants

UDP_MIN: 2 minutes (as defined in [RFC4787])

UDP_DEFAULT: 5 minutes (as defined in [RFC4787])

TCP_TRANS: 4 minutes (as defined in [RFC5382])

TCP_EST: 2 hours (The minimum lifetime for an established TCP session defined in [RFC5382] is 2 hours and 4 minutes, which is achieved by adding the 2 hours with this timer and the 4 minutes with the TCP_TRANS timer.)

TCP_INCOMING_SYN: 6 seconds (as defined in [RFC5382])

FRAGMENT_MIN: 2 seconds

ICMP_DEFAULT: 60 seconds (as defined in [RFC5508])

5. Operational Considerations

Since the original [RFC6146] was published, there have been a notable number of new standards that, making use or working together NAT64, have allowed very relevant improvements towards the deployment of IPv6, greatly facilitating the transition.

This non-normative section briefly summarizes those standards and their relevance to the NAT64 updated specification, as well as relevant operational considerations.

5.1. NAT64 Prefix

[RFC6052] defined the WKP, which did not allowed the use of non-global IPv4 addresses. This has been discussed as inconvenient and is being reversed by NAT64 WKP [I-D.ietf-v6ops-nat64-wkp-1918]. Further to that, [RFC8215] specify a Local-Use IPv4/IPv6 Translation Prefix, adjacent to the WKP, facilitating the coexistence of multiple IPv4/IPv6 translation mechanisms in a single network domain.

5.2. NAT64 Prefix Discovery

[RFC7050] defined a best effort method for clients to discover the Pref64 being used by the NAT64. This was updated by [RFC8880] in order to confirm the need for the special use of the domain 'ipv4only.arpa', that must be configured in DNS64 recursive resolvers. However, in many deployments this zone is not being properly configured, which has implications on the load of IANA servers with unnecessary queries and lack of optimizations.

In a similar direction and to improve the discovery of the Pref64, instead of being a "best effort" method, [RFC8781], specified a ND option to be used in RAs. [RFC9872] further provides a recommendation for using [RFC8781] instead of [RFC7050]/[RFC8880].

One more alternative is the use [RFC7225] "Discovering NAT64 IPv6 Prefixes Using the Port Control Protocol (PCP)".

Section 3 of "Analysis of Solution Proposals for Hosts to Learn NAT64 Prefix" [RFC7051] already exposed the issues of the NAT64 prefix discovery, most of them resolved by [RFC8781], nonetheless is important for operators to review that in order to ensure a proper deployment.

5.3. NAT64 in other Protocols

464XLAT [RFC6877] resolved some of the issues of NAT64, such the reachability of IPv4-only destinations when DNS is not being used (literal IPv4 addresses, code-embedded IPv4 addresses, etc.). In fact it allows NAT64 to be used without DNS64 if a CLAT function is present in the host or other elements of the network. Note that not using DNS64 has some implications, as already described in [RFC8683]. 464XLAT has extensively deployed in the Internet, becoming the most succesful transition mechanism, generally using DNS64. [RFC8585] added information about the steps needed to configure CLAT in a CE in order to facilitate the deployment of stateful NAT64 in broadband networks.

Taking advantage of 464XLAT, and the new IPv6-Only Preferred Option for DHCPv4 [RFC8925], [I-D.ietf-v6ops-claton] further defines a detailed arquitecture in order to improve the implementatation and usage of CLAT in hosts and routers. This becomes a complete solution, specially for the case of enterprise networks with the definition of "IPv6-mostly" done by [I-D.ietf-v6ops-6mops].

Finally, [I-D.ietf-v6ops-icmpext-xlat-v6only-source] suggest that when a source IPv6 address of an ICMPv6 message can not be translated to an IPv4 address, the protocol translators use a special dummy address.

5.4. DNS64 Stub-Resolver Mode, Self-Synthesis or Local-Synthesis

DNS64 [RFC6147] already mentions that one of the options to locate the DNS64 function, in addition to authoritative and recursive name servers, is in the end host (stub resolver). In this document this is referenced as "DNS64 stub-resolver mode". Note that despite the information related to the "no need to update hosts" to deploy NAT64/DNS64 this has some possible issues, which were resolved by 464XLAT [RFC6877], as explained above. So, in most of the cases, actually the hosts Operating Systems are being updated not just becuase NAT64/DNS64, but other reasons, which means that modern hosts often don’t have an issue with that.

Happy Eyeballs version 2 [RFC8305] documents this using the terminology "self-synthesis", where hosts can synthetise the AAAA RRs by themselves, as an alternative for a DNS64-server synthetised response. This approach has many advantages in situations like DNSSEC validation being performed by the host and needs to be further explored in a new version of the DNS64 document.

Other documents use the terminology "local-synthesys".

5.5. Port Control Protocol

The Port Control Protocol (PCP) [RFC6887], has been designed in order to provide a mechanism to control how incoming packets are forwarded by upstream devices, such as in this case the stateful NAT64, avoiding the need for keepalive traffic.

5.6. Issues with IP Address Sharing

In the same line as when using IPv4-IPv4(-IPv4) NAT (NAT44/NAT444), NAT64 share many of the issues described in [RFC6269]. This needs to be carefully evaluated in any NAT64 deployment.

5.7. Previous Operational Experience

As many operators have been extensively deploying NAT64 in different enviroments, there are extensive recommendations based on that experience. Two complementary documents provide a good advice, from slighly different perspectives, "NAT64 Deployment Options and Experience" [RFC7269] and "Additional Deployment Guidelines for NAT64/464XLAT in Operator and Enterprise Networks" [RFC8683].

5.8. Benchmarking and Scalability

For the dimensioning of NAT64 deployments, "Benchmarking Methodology for Stateful NATxy Gateways" [RFC9693] provides useful considerations. In addition to that, actual benchmarking results for NAT64 implementations are provided by [Len2023] and [Len2024].

5.9. Port Allocation Schemes

NAT64 translators often by default are configured to maximize the use of ports per IPv4 public address, not pre-allocating a port-range per subscriber. This is actually one of the advantages of NAT64 compared with other transition mechanisms, as described "Pros and Cons of IPv6 Transition Technologies for IPv4-as-a-Service (IPv4aaS)" [RFC9313] (sections 3.4 and 4.7). This means that a much smaller IPv4 pool will be able to serve a larger number of subscribers. The trade-off is that logging requirements are bigger. However, this is tied to logging regulations requirements.

5.10. Logging, Alarms and Event Reporting

"Common Requirements for Carrier-Grade NATs (CGNs)" [RFC6888] analyzes common requirements for translators, applicable also to NAT64. Section 4 is devoted to logging requirements.

Operators also may need to configure alarms and events reporting, which can be done by using "IP Flow Information Export (IPFIX) Information Elements for Logging NAT Events" [RFC8158].

6. Implementation Status

Note to RFC Editor: If this document needs to be published, please remove this section before publication, as it is only intended for the IESG evaluation.

This section summarized the known status of existing and interoperable implementations of the protocol subject of this document, as well as closely related protocols. This is following ([RFC7942]) and intended to assist the relevant WGs, IESG and IETF as a whole, in the evaluation of the document for the document progress through the standardization process.

The description of the implementations does not imply any IETF endorsement and is solely based on publicly available information, which has not been formally confirmed by specific interoperability testing for this document publication; however, it is known to be confirmed by existing commercial working deployments worldwide and without known interoperability issues.

Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers ([RFC6146]) was originally published in April 2011.

([RFC6146]) is implemented together with other related protocols (just to name a few of the most relevant ones) such as:

* IPv6 Addressing of IPv4/IPv6 Translators ([RFC6052]).

* IP/ICMP Translation Algorithm ([RFC7915]).

Follows a list of known implementations by different products/ vendors, known to be mature and in production products/networks/ services worlwide:

* 6Wind. Implemented in multiple products. https://www.6wind.com/6wind-cg-nat-vrouter-with-nat64/.

* A10. Implemented in multiple products.

* AlliedTelesis. Implemented in multiple products. https://www.alliedtelesis.com/sites/default/files/documents/configuration-guides/transitioning_ipv4_to_ipv6_feature_overview_guide.pdf.

* Amazon. Virtual Private Cloud. https://docs.aws.amazon.com/vpc/latest/userguide/nat-gateway-nat64-dns64.html.

* Apple. Implemented since 2016. https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/UnderstandingandPreparingfortheIPv6Transition/UnderstandingandPreparingfortheIPv6Transition.html.

* Arista. Implemented in multiple products. https://www.arista.com/en/support/toi/eos-4-24-0f/14495-map-t-border-relay.

* Broadcom. Implemented in VMWare. https://techdocs.broadcom.com/us/en/vmware-cis/nsx/nsxt-dc/3-1/administration-guide/network-address-translation/configure-an-nsx-nat64.html.

* Cisco. Implemented in multiple series of products since 2010. https://www.cisco.com/c/en/us/td/docs/routers/ios/config/17-x/ip-addressing/b-ip-addressing/m_iadnat-stateless-nat64.html.

* Ecdysis. http://ecdysis.viagenie.ca/.

* F5. Implemented in multiple products. https://techdocs.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/cgn-implementations-11-6-0/2.html.

* Fortinet. Implemented in multiple products. https://docs.fortinet.com/document/fortigate/7.4.6/fortinet-carrier-grade-nat-field-reference-architecture-guide/891965/nat64.

* Huawei. Implemented in multiple series of products. https://support.huawei.com/enterprise/en/doc/EDOC1100278545/fe351de4/nat64-configuration.

* Infoblox. Implemented as part of the DNS64 support. https://www.a10networks.com/products/thunder-cgn/.

* Jool. Implemented since 2014. https://nicmx.github.io/Jool/en/index.html.

* Juniper. Implemented in multiple series of products. https://www.juniper.net/documentation/us/en/software/nce/nce-nat64-ipv6-ipv4-depletion/topics/concept/ipv6-nat64-ipv4-depletion-overview.html.

* Nokia. Implemented in multiple products. https://documentation.nokia.com/html/0_add-h-f/93-0262-HTML/7750_SR_OS_MSISA_Guide/Application-Assurance-NAT.pdf.

* OpenWrt. https://github.com/openwrt and https://openwrt.org.

* Palo Alto. Implemented in multiple products. https://docs.paloaltonetworks.com/ngfw/networking/nat64.

* Sophos. Implemented in multiple products. https://news.sophos.com/en-us/2025/04/08/sophos-firewall-v21-5-early-access-is-now-available/.

* Tayga. https://github.com/openthread/tayga and https://github.com/apalrd/tayga.

* VPP. https://docs.fd.io/vpp/17.07/nat64_doc.html.

* ZTE. Implemented in multiple products. https://www.zte.com.cn/global/product_index/ip_network_en/68e_e/zxr10-6800e/zxr10-6800e.html.

Note that even an effort has been done to compile an extensive list (including a relevant URL), there may be many more implementations not publicly known, so this list does not pretent to be exclusive, just an indication of a sufficient number of implementations, as required for the evaluation of the current implementation status.

7. IANA Considerations

This document requests IANA to replace references to [RFC6146] in the following registry groups with references to this document:

* Service Function Chaining Service Function Types. https://www.iana.org/assignments/service-function-chaining-service-function-types/service-function-chaining-service-function-types.xhtml.

* IP Flow Information Export (IPFIX) Entities. https://www.iana.org/assignments/ipfix/ipfix.xhtml.

8. Security Considerations

8.1. Implications on End-to-End Security

Any protocols that protect IP header information are essentially incompatible with NAT64. This implies that end-to-end IPsec verification will fail when the Authentication Header (AH) is used (both transport and tunnel mode) and when ESP is used in transport mode. This is inherent in 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.

8.2. 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 [RFC4787], a NAT64 MUST offer "Endpoint-Independent Mapping". This means:

> For any IPv6 packet with source (S’1,s1) and destination (Pref64::D1,d1) that creates an external mapping to (S1,s1v4), (D1,d1), for any subsequent packet from (S’1,s1) to (Pref64::D2,d2) that creates an external mapping to (S2,s2v4), (D2,d2), within a given binding timer window,
>
> (S1,s1v4) = (S2,s2v4) for all values of D2,d2

Implementations MAY also provide support for "Address-Dependent Mapping" as also defined in this document and following the guidelines defined in [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 Mapping" 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 Mapping" 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 be discarded. A dynamic filter may be employed whereby 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 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 an "Address-Dependent Mapping" behavior, though not by its mapping behavior, but instead by its filtering behavior.

As currently specified, the NAT64 only requires filtering traffic based on the 5-tuple. In some cases (e.g., statically configured mappings), this may make it easy for an attacker to guess. An attacker need not be able to guess other fields, e.g., the TCP sequence number, to get a packet through the NAT64. While such traffic might be dropped by the final destination, it does not provide additional mitigations against bandwidth/CPU attacks targeting the internal network. To avoid this type of abuse, a NAT64 MAY keep track of the sequence number of TCP packets in order to verify the proper sequencing of exchanged segments, in particular, those of the SYNs and the FINs.

8.3. Attacks on NAT64

The NAT64 device itself is a potential victim of different types of attacks. In particular, the NAT64 can be a victim of DoS attacks. The NAT64 device 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. 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 on 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. Similarly, a DoS attack against the NAT64 can be crafted by sending either V4 or V6 SYN packets that consume memory in the form of session and/or binding table entries. In the case of IPv4 SYNs the situation is aggravated by the requirement to also store the data packets for a given amount of time, requiring more memory from the NAT64 device. NAT64 devices MUST implement proper protection against such attacks, for instance, allocating a limited amount of memory for fragmented packet storage as specified in Section 3.4.

Another consideration related to NAT64 resource depletion refers to the preservation of binding state. Attackers may try to keep a binding state alive forever by sending periodic packets that refresh the state. In order to allow the NAT64 to defend against such attacks, the NAT64 MAY choose not to extend the session entry lifetime for a specific entry upon the reception of packets for that entry through the external interface. As described in the framework document [RFC6144], the NAT64 can be deployed in multiple scenarios, in some of which the Internet side is the IPv6 one, and in others of which the Internet side is the IPv4 one. It is then important to properly set which is the Internet side of the NAT64 in each specific configuration.

8.4. Avoiding Hairpinning Loops

If an IPv6-only client can guess the IPv4 binding address that will be created, it can use the IPv6 representation of that address as the source address for creating this binding. Then, any packet sent to the binding’s IPv4 address could loop in the NAT64. This is prevented in the current specification by filtering incoming packets containing Pref64::/n in the source address, as described below.

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: any

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

It is not hard to guess the IPv4 address that will be allocated. First, the attacker creates a binding and uses (for example) Simple Traversal of the UDP Protocol through NAT (STUN) [RFC8489] to learn its external IPv4 address. New bindings will always have this address. Then, it uses a source port in the range 1-1023. This will increase the chances to 1/512 (since range and parity are preserved by NAT64 in UDP).

In order to address this vulnerability, the NAT64 MUST drop IPv6 packets whose source address is in Pref64::/n, as defined in Section 3.5.

9. Contributors (original authors of RFC6146 (2011)

Marcelo Bagnulo
UC3M
Av. Universidad 30
28911 Leganes Madrid
Spain
Phone: +34-91-6249500
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
Email: philip_matthews@magma.ca

Iljitsch van Beijnum
IMDEA Networks
Avda. del Mar Mediterraneo, 22
28918 Leganes Madrid
Spain
Email: iljitsch@muada.com

10. Contributors

George Tsirtsis

Qualcomm
tsirtsis@googlemail.com

Greg Lebovitz

Juniper
gregory.ietf@gmail.com

Simon Perreault

Viagenie
simon.perreault@viagenie.ca

11. Acknowledgements

Thanks to Mohamed Boucadair, Michael Richardson, Tom Petch, Ted Lemon, Daryll Sweer, Brian E. Carpenter, Goetz Goerisch, Gabor Lencse and XiPeng Xiao for the inputs provided.

Special thanks to Alberto Leiva Popper, who not only reported errata 4756, but also engaged in private discussions and provided a very detailed explanation, as experienced NAT64 implementor (Jool), easing the job to draft the text for resolving this errata. Also to Marc Lepage, who reported errata 8416.

Lorenzo Colitti, Dave Thaler, Dan Wing, Alberto Garcia-Martinez, Reinaldo Penno, Ranjana Rao, Lars Eggert, Senthil Sivakumar, Zhen Cao, Xiangsong Cui, Mohamed Boucadair, Dong Zhang, Bryan Ford, Kentaro Ebisawa, Charles Perkins, Magnus Westerlund, Ed Jankiewicz, David Harrington, Peter McCann, Julien Laganier, Pekka Savola, and Joao Damas reviewed the document and provided useful comments to improve it.

The content of the document 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.

12. ANNEX: Changes since RFC6146

* This version basically resolved 2 errata (4756 and 8416). Note of the errata have any implications in the protocol itself.

* Updated references.

* Added implementation status section.

* Added IANA Considerations section.

* A few grammar corrections and shortened sentences.

* Improved/updated text in intro, related to EAM, usage of NAT64 for 464XLAT and IPv6-Mostly, Pref64 discovery and self-synthesys.

* Added new section with Operational Considerations.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <https://www.rfc-editor.org/info/rfc4787>.

[RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <https://www.rfc-editor.org/info/rfc5382>.

[RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, DOI 10.17487/RFC5508, April 2009, <https://www.rfc-editor.org/info/rfc5508>.

[RFC6052] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", RFC 6052, DOI 10.17487/RFC6052, October 2010, <https://www.rfc-editor.org/info/rfc6052>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <https://www.rfc-editor.org/info/rfc6146>.

[RFC7915] Bao, C., Li, X., Baker, F., Anderson, T., and F. Gont, "IP/ICMP Translation Algorithm", RFC 7915, DOI 10.17487/RFC7915, June 2016, <https://www.rfc-editor.org/info/rfc7915>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

13.2. Informative References

[I-D.ietf-v6ops-6mops]
Buraglio, N., Caletka, O., and J. Linkova, "IPv6-mostly Networks: Deployment and Operations Considerations", Work in Progress, Internet-Draft, draft-ietf-v6ops-6mops-07, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-6mops-07>.

[I-D.ietf-v6ops-claton]
Colitti, L., Linkova, J., and T. Jensen, "464XLAT Customer-side Translator (CLAT): Node Behavior and Recommendations", Work in Progress, Internet-Draft, draft-ietf-v6ops-claton-16, 5 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-claton-16>.

[I-D.ietf-v6ops-icmpext-xlat-v6only-source]
Lamparter, D. E. and J. Linkova, "Using Dummy IPv4 Address and Node Identification Extensions for IP/ICMP translators (XLATs)", Work in Progress, Internet-Draft, draft-ietf-v6ops-icmpext-xlat-v6only-source-01, 6 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-icmpext-xlat-v6only-source-01>.

[I-D.ietf-v6ops-nat64-wkp-1918]
Kumari, W. and J. Linkova, "NAT64 WKP", Work in Progress, Internet-Draft, draft-ietf-v6ops-nat64-wkp-1918-02, 16 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-nat64-wkp-1918-02>.

[Len2023] Lencse, G., Shima, K., and K. Cho, "Benchmarking methodology for stateful NAT64 gateways", Computer Communications, vol. 210, no. 1, pp. 256-272, DOI 10.1016/j.comcom.2023.08.009, 1 October 2023, <https://www.sciencedirect.com/science/article/pii/S0140366423002931>.

[Len2024] Lencse, G., "Making Stateless and Stateful Network Performance Measurements Unbiased", Computer Communications, vol. 225, no. 1, pp. 141-155, DOI 10.1016/j.comcom.2024.05.018, 1 September 2024, <https://www.sciencedirect.com/science/article/abs/pii/S0140366424001993>.

[RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, DOI 10.17487/RFC1858, October 1995, <https://www.rfc-editor.org/info/rfc1858>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <https://www.rfc-editor.org/info/rfc3022>.

[RFC3128] Miller, I., "Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)", RFC 3128, DOI 10.17487/RFC3128, June 2001, <https://www.rfc-editor.org/info/rfc3128>.

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, DOI 10.17487/RFC3948, January 2005, <https://www.rfc-editor.org/info/rfc3948>.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <https://www.rfc-editor.org/info/rfc4963>.

[RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, DOI 10.17487/RFC6144, April 2011, <https://www.rfc-editor.org/info/rfc6144>.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <https://www.rfc-editor.org/info/rfc6147>.

[RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <https://www.rfc-editor.org/info/rfc6269>.

[RFC6877] Mawatari, M., Kawashima, M., and C. Byrne, "464XLAT: Combination of Stateful and Stateless Translation", RFC 6877, DOI 10.17487/RFC6877, April 2013, <https://www.rfc-editor.org/info/rfc6877>.

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/info/rfc6887>.

[RFC6888] Perreault, S., Ed., Yamagata, I., Miyakawa, S., Nakagawa, A., and H. Ashida, "Common Requirements for Carrier-Grade NATs (CGNs)", BCP 127, RFC 6888, DOI 10.17487/RFC6888, April 2013, <https://www.rfc-editor.org/info/rfc6888>.

[RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <https://www.rfc-editor.org/info/rfc7050>.

[RFC7051] Korhonen, J., Ed. and T. Savolainen, Ed., "Analysis of Solution Proposals for Hosts to Learn NAT64 Prefix", RFC 7051, DOI 10.17487/RFC7051, November 2013, <https://www.rfc-editor.org/info/rfc7051>.

[RFC7225] Boucadair, M., "Discovering NAT64 IPv6 Prefixes Using the Port Control Protocol (PCP)", RFC 7225, DOI 10.17487/RFC7225, May 2014, <https://www.rfc-editor.org/info/rfc7225>.

[RFC7269] Chen, G., Cao, Z., Xie, C., and D. Binet, "NAT64 Deployment Options and Experience", RFC 7269, DOI 10.17487/RFC7269, June 2014, <https://www.rfc-editor.org/info/rfc7269>.

[RFC7757] Anderson, T. and A. Leiva Popper, "Explicit Address Mappings for Stateless IP/ICMP Translation", STD 103, RFC 7757, DOI 10.17487/RFC7757, February 2016, <https://www.rfc-editor.org/info/rfc7757>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC8158] Sivakumar, S. and R. Penno, "IP Flow Information Export (IPFIX) Information Elements for Logging NAT Events", RFC 8158, DOI 10.17487/RFC8158, December 2017, <https://www.rfc-editor.org/info/rfc8158>.

[RFC8215] Anderson, T., "Local-Use IPv4/IPv6 Translation Prefix", RFC 8215, DOI 10.17487/RFC8215, August 2017, <https://www.rfc-editor.org/info/rfc8215>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

[RFC8489] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", RFC 8489, DOI 10.17487/RFC8489, February 2020, <https://www.rfc-editor.org/info/rfc8489>.

[RFC8585] Palet Martinez, J., Liu, H. M.-H., and M. Kawashima, "Requirements for IPv6 Customer Edge Routers to Support IPv4-as-a-Service", RFC 8585, DOI 10.17487/RFC8585, May 2019, <https://www.rfc-editor.org/info/rfc8585>.

[RFC8683] Palet Martinez, J., "Additional Deployment Guidelines for NAT64/464XLAT in Operator and Enterprise Networks", RFC 8683, DOI 10.17487/RFC8683, November 2019, <https://www.rfc-editor.org/info/rfc8683>.

[RFC8781] Colitti, L. and J. Linkova, "Discovering PREF64 in Router Advertisements", RFC 8781, DOI 10.17487/RFC8781, April 2020, <https://www.rfc-editor.org/info/rfc8781>.

[RFC8839] Petit-Huguenin, M., Nandakumar, S., Holmberg, C., Ker辰nen, A., and R. Shpount, "Session Description Protocol (SDP) Offer/Answer Procedures for Interactive Connectivity Establishment (ICE)", RFC 8839, DOI 10.17487/RFC8839, January 2021, <https://www.rfc-editor.org/info/rfc8839>.

[RFC8880] Cheshire, S. and D. Schinazi, "Special Use Domain Name 'ipv4only.arpa'", RFC 8880, DOI 10.17487/RFC8880, August 2020, <https://www.rfc-editor.org/info/rfc8880>.

[RFC8925] Colitti, L., Linkova, J., Richardson, M., and T. Mrugalski, "IPv6-Only Preferred Option for DHCPv4", RFC 8925, DOI 10.17487/RFC8925, October 2020, <https://www.rfc-editor.org/info/rfc8925>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

[RFC9313] Lencse, G., Palet Martinez, J., Howard, L., Patterson, R., and I. Farrer, "Pros and Cons of IPv6 Transition Technologies for IPv4-as-a-Service (IPv4aaS)", RFC 9313, DOI 10.17487/RFC9313, October 2022, <https://www.rfc-editor.org/info/rfc9313>.

[RFC9693] Lencse, G. and K. Shima, "Benchmarking Methodology for Stateful NATxy Gateways", RFC 9693, DOI 10.17487/RFC9693, January 2025, <https://www.rfc-editor.org/info/rfc9693>.

[RFC9872] Buraglio, N., Jensen, T., and J. Linkova, "Recommendations for Discovering IPv6 Prefix Used for IPv6 Address Synthesis", RFC 9872, DOI 10.17487/RFC9872, September 2025, <https://www.rfc-editor.org/info/rfc9872>.

Authors' Addresses

Marcelo Bagnulo
UC3M
Av. Universidad 30
28911 Leganes Madrid
Spain
Phone: +34-91-6249500
Email: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es/marcelo

Philip Matthews
Email: philip_matthews@magma.ca

Jordi Palet Martinez (editor)
The IPv6 Company
Molino de la Navata, 75
28420 La Navata - Galapagar (Madrid)
Spain
Email: jordi.palet@theipv6company.com
URI: http://www.theipv6company.com/