Internet Engineering Task Force (IETF)
Request for Comments: 6296
Category: Experimental
ISSN: 2070-1721

M. Wasserman
Painless Security
F. Baker
Cisco Systems

June 2011

IPv6-to-IPv6 Network Prefix Translation

Abstract

This document describes a stateless, transport-agnostic IPv6-to-IPv6 Network Prefix Translation (NPTv6) function that provides the address-independence benefit associated with IPv4-to-IPv4 NAT (NAPT44) and provides a 1:1 relationship between addresses in the "inside" and "outside" prefixes, preserving end-to-end reachability at the network layer.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6296.

Table of Contents

1. Introduction 3
1.1. What is Address Independence? 4
1.2. NPTv6 Applicability 5
1.3. Requirements Terminology 7
2. NPTv6 Overview 7
2.1. NPTv6: The Simplest Case 7
2.2. NPTv6 between Peer Networks 8
2.3. NPTv6 Redundancy and Load Sharing 9
2.4. NPTv6 Multihoming 9
2.5. Mapping with No Per-Flow State 10
2.6. Checksum-Neutral Mapping 10
3. NPTv6 Algorithmic Specification 11
3.1. NPTv6 Configuration Calculations 11
3.2. NPTv6 Translation, Internal Network to External Network . 12
3.3. NPTv6 Translation, External Network to Internal Network . 12
3.4. NPTv6 with a /48 or Shorter Prefix 12
3.5. NPTv6 with a /49 or Longer Prefix 13
3.6. /48 Prefix Mapping Example 13
3.7. Address Mapping for Longer Prefixes 14
4. Implications of Network Address Translator Behavioral Requirements 15
4.1. Prefix Configuration and Generation 15
4.2. Subnet Numbering 15
4.3. NAT Behavioral Requirements 15
5. Implications for Applications 16
5.1. Recommendation for Network Planners Considering Use of NPTv6 Translation 17
5.2. Recommendations for Application Writers 18
5.3. Recommendation for Future Work 18
6. A Note on Port Mapping 18
7. Security Considerations 19
8. Acknowledgements 19
9. References 20
9.1. Normative References 20
9.2. Informative References 20
Appendix A. Why GSE? 23
Appendix B. Verification Code 25

1. Introduction

This document describes a stateless IPv6-to-IPv6 Network Prefix Translation (NPTv6) function, designed to provide address independence to the edge network. It is transport-agnostic with respect to transports that do not checksum the IP header, such as SCTP, and to transports that use the TCP/UDP/DCCP (Datagram Congestion Control Protocol) pseudo-header and checksum [RFC1071].

For reasons discussed in [RFC2993] and Section 5, the IETF does not recommend the use of Network Address Translation technology for IPv6. Where translation is implemented, however, this specification provides a mechanism that has fewer architectural problems than merely implementing a traditional stateful Network Address Translator in an IPv6 environment. It also provides a useful alternative to the complexities and costs imposed by multihoming using provider-independent addressing and the routing and network management issues of overlaid ISP address space. Some problems remain, however. The reader should consider the alternatives suggested in [RFC4864] and the considerations of [RFC5902] for improved approaches.

The stateless approach described in this document has several ramifications:

- Any security benefit that NAPT44 might offer is not present in NPTv6, necessitating the use of a firewall to obtain those benefits if desired. An example of such a firewall is described in [RFC6092].

- End-to-end reachability is preserved, although the address used "inside" the edge network differs from the address used "outside" the edge network. This has implications for application referrals and other uses of Internet layer addresses.

- If there are multiple identically configured prefix translators between two networks, there is no need for them to exchange dynamic state, as there is no dynamic state -- the algorithmic translation will be identical across each of them. The network can therefore asymmetrically route, load share, and fail-over among them without issue.

- Since translation is 1:1 at the network layer, there is no need to modify port numbers or other transport parameters.

- TCP sessions that authenticate peers using the TCP Authentication Option [RFC5925] cannot have their addresses translated, as the addresses are used in the calculation of the Message Authentication Code. This consideration applies in general to any

UNilateral Self-Address Fixing (UNSAF) [RFC3424] Protocol, which the IAB recommends against the deployment of in an environment that changes Internet addresses.

- o Applications using the Internet Key Exchange Protocol Version 2 (IKEv2) [RFC5996] should, at least in theory, detect the presence of the translator; while no NAT traversal solution is required, [RFC5996] would require such sessions to use UDP.

1.1. What is Address Independence?

For the purposes of this document, IPv6 address independence consists of the following set of properties:

From the perspective of the edge network:

* The IPv6 addresses used inside the local network (for interfaces, access lists, and logs) do not need to be renumbered if the global prefix(es) assigned for use by the edge network are changed.

* The IPv6 addresses used inside the edge network (for interfaces, access lists, and logs) or within other upstream networks (such as when multihoming) do not need to be renumbered when a site adds, drops, or changes upstream networks.

* It is not necessary for an administration to convince an upstream network to route its internal IPv6 prefixes or for it to advertise prefixes derived from other upstream networks into it.

* Unless it wants to optimize routing between multiple upstream networks in the process of multihoming, there is no need for a BGP exchange with the upstream network.

From the perspective of the upstream network:

* IPv6 addresses used by the edge network are guaranteed to have a provider-allocated prefix, eliminating the need and concern for BCP 38 [RFC2827] ingress filtering and the advertisement of customer-specific prefixes.

Thus, address independence has ramifications for the edge network, networks it directly connects with (especially its upstream networks), and the Internet as a whole. The desire for address independence has been a primary driver for IPv4 NAT deployment in medium- to large-sized enterprise networks, including NAT deployments

in enterprises that have plenty of IPv4 provider-independent address space (from IPv4 "swamp space"). It has also been a driver for edge networks to become members of Regional Internet Registry (RIR) communities, seeking to obtain BGP Autonomous System Numbers and provider-independent prefixes, and as a result has been one of the drivers of the explosion of the IPv4 route table. Service providers have stated that the lack of address independence from their customers has been a negative incentive to deployment, due to the impact of customer routing expected in their networks.

The Local Network Protection [RFC4864] document discusses a related concept called "Address Autonomy" as a benefit of NAPT44. [RFC4864] indicates that address autonomy can be achieved by the simultaneous use of global addresses on all nodes within a site that need external connectivity and Unique Local Addresses (ULAs) [RFC4193] for all internal communication. However, this solution fails to meet the requirement for address independence, because if an ISP renumbering event occurs, all of the hosts, routers, DHCP servers, Access Control Lists (ACLs), firewalls, and other internal systems that are configured with global addresses from the ISP will need to be renumbered before global connectivity is fully restored.

The use of IPv6 provider-independent (PI) addresses has also been suggested as a means to fulfill the address-independence requirement. However, this solution requires that an enterprise qualify to receive a PI assignment and persuade its ISP to install specific routes for the enterprise’s PI addresses. There are a number of practical issues with this approach, especially if there is a desire to route to a number of geographically and topologically diverse sites, which can sometimes involve coordinating with several ISPs to route portions of a single PI prefix. These problems have caused numerous enterprises with plenty of IPv4 swamp space to choose to use IPv4 NAT for part, or substantially all, of their internal network instead of using their provider-independent address space.

1.2. NPTv6 Applicability

NPTv6 provides a simple and compelling solution to meet the address-independence requirement in IPv6. The address-independence benefit stems directly from the translation function of the network prefix translator. To avoid as many of the issues associated with NAPT44 as possible, NPTv6 is defined to include a two-way, checksum-neutral, algorithmic translation function, and nothing else.

The fact that NPTv6 does not map ports and is checksum-neutral avoids the need for an NPTv6 Translator to rewrite transport layer headers. This makes it feasible to deploy new or improved transport layer

protocols without upgrading NPTv6 Translators. Similarly, since NPTv6 does not rewrite transport layer headers, NPTv6 will not interfere with encryption of the full IP payload in many cases.

The default NPTv6 address-mapping mechanism is purely algorithmic, so NPTv6 translators do not need to maintain per-node or per-connection state, allowing deployment of more robust and adaptive networks than can be deployed using NAPT44. Since the default NPTv6 mapping can be performed in either direction, it does not interfere with inbound connection establishment, thus allowing internal nodes to participate in direct Peer-to-Peer applications without the application layer overhead one finds in many IPv4 Peer-to-Peer applications.

Although NPTv6 compares favorably to NAPT44 in several ways, it does not eliminate all of the architectural problems associated with IPv4 NAT, as described in [RFC2993]. NPTv6 involves modifying IP headers in transit, so it is not compatible with security mechanisms, such as the IPsec Authentication Header, that provide integrity protection for the IP header. NPTv6 may interfere with the use of application protocols that transmit IP addresses in the application-specific portion of the IP datagram. These applications currently require Application Layer Gateways (ALGs) to work correctly through NAPT44 devices, and similar ALGs may be required for these applications to work through NPTv6 Translators. The use of separate internal and external prefixes creates complexity for DNS deployment, due to the desire for internal nodes to communicate with other internal nodes using internal addresses, while external nodes need to obtain external addresses to communicate with the same nodes. This frequently results in the deployment of "split DNS", which may add complexity to network configuration.

The choice of address within the edge network bears consideration. One could use a ULA, which maximizes address independence. That could also be considered a misuse of the ULA; if the expectation is that a ULA prevents access to a system from outside the range of the ULA, NPTv6 overrides that. On the other hand, the administration is aware that it has made that choice and could deploy a second ULA for the purpose of privacy if it desired; the only prefix that will be translated is one that has an NPTv6 Translator configured to translate to or from it. Also, using any other global-scope address format makes one either obtain a PI prefix or be at the mercy of the agency from which it was allocated.

There are significant technical impacts associated with the deployment of any prefix translation mechanism, including NPTv6, and we strongly encourage anyone who is considering the implementation or

deployment of NPTv6 to read [RFC4864] and [RFC5902], and to carefully consider the alternatives described in that document, some of which may cause fewer problems than NPTv6.

1.3. Requirements Terminology

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

2. NPTv6 Overview

NPTv6 may be implemented in an IPv6 router to map one IPv6 address prefix to another IPv6 prefix as each IPv6 datagram transits the router. A router that implements an NPTv6 prefix translation function is referred to as an NPTv6 Translator.

2.1. NPTv6: The Simplest Case

In its simplest form, an NPTv6 Translator interconnects two network links, one of which is an "internal" network link attached to a leaf network within a single administrative domain and the other of which is an "external" network with connectivity to the global Internet. All of the hosts on the internal network will use addresses from a single, locally routed prefix, and those addresses will be translated to/from addresses in a globally routable prefix as IP datagrams transit the NPTv6 Translator. The lengths of these two prefixes will be functionally the same; if they differ, the longer of the two will limit the ability to use subnets in the shorter.

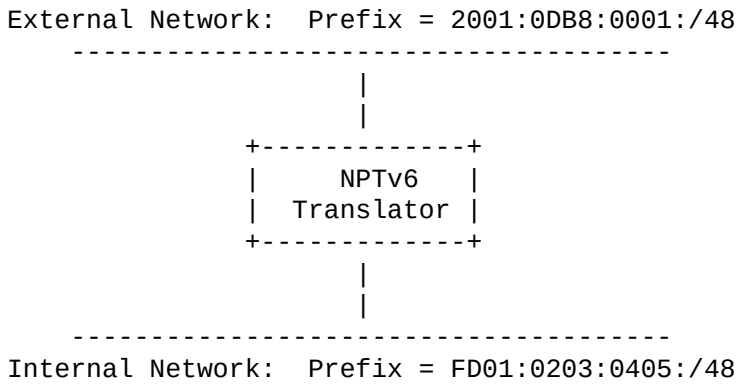

Figure 1: A Simple Translator

Figure 1 shows an NPTv6 Translator attached to two networks. In this example, the internal network uses IPv6 Unique Local Addresses (ULAs) [RFC4193] to represent the internal IPv6 nodes, and the external network uses globally routable IPv6 addresses to represent the same nodes.

When an NPTv6 Translator forwards datagrams in the "outbound" direction, from the internal network to the external network, NPTv6 overwrites the IPv6 source prefix (in the IPv6 header) with a corresponding external prefix. When datagrams are forwarded in the "inbound" direction, from the external network to the internal network, the IPv6 destination prefix is overwritten with a corresponding internal prefix. Using the prefixes shown in the diagram above, as an IP datagram passes through the NPTv6 Translator in the outbound direction, the source prefix (FD01:0203:0405:/48) will be overwritten with the external prefix (2001:0DB8:0001:/48). In an inbound datagram, the destination prefix (2001:0DB8:0001:/48) will be overwritten with the internal prefix (FD01:0203:0405:/48). In both cases, it is the local IPv6 prefix that is overwritten; the remote IPv6 prefix remains unchanged. Nodes on the internal network are said to be "behind" the NPTv6 Translator.

2.2. NPTv6 between Peer Networks

NPTv6 can also be used between two private networks. In these cases, both networks may use ULA prefixes, with each subnet in one network mapped into a corresponding subnet in the other network, and vice versa. Or, each network may use ULA prefixes for internal addressing and global unicast addresses on the other network.

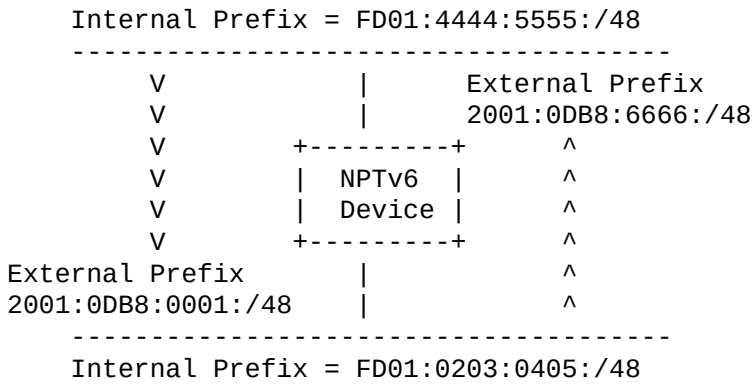

Figure 2: Flow of Information in Translation

2.3. NPTv6 Redundancy and Load Sharing

In some cases, more than one NPTv6 Translator may be attached to a network, as shown in Figure 3. In such cases, NPTv6 Translators are configured with the same internal and external prefixes. Since there is only one translation, even though there are multiple translators, they map only one external address (prefix and Interface Identifier (IID)) to the internal address.

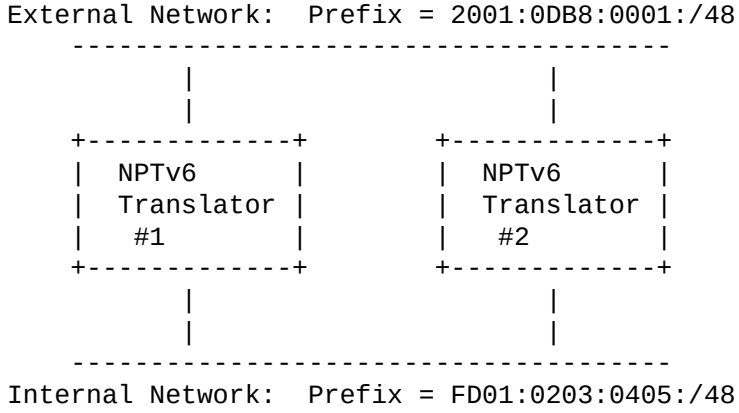

Figure 3: Parallel Translators

2.4. NPTv6 Multihoming

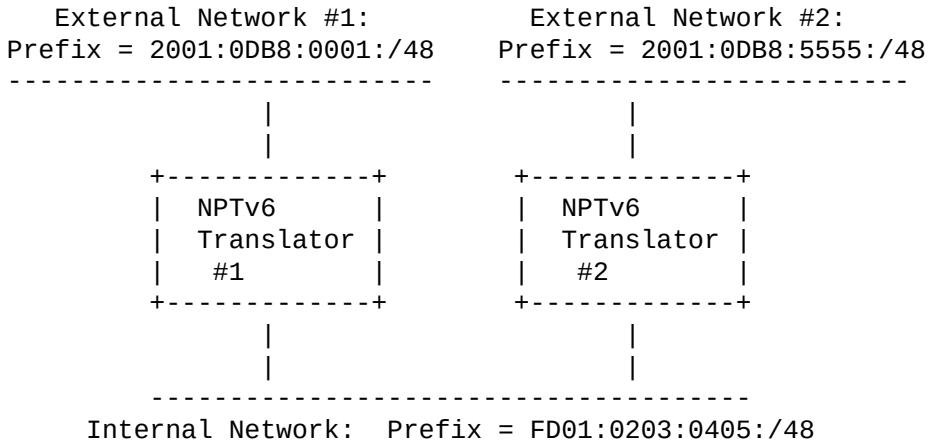

Figure 4: Parallel Translators with Different Upstream Networks

When multihoming, NPTv6 Translators are attached to an internal network, as shown in Figure 4, but are connected to different external networks. In such cases, NPTv6 Translators are configured with the same internal prefix but different external prefixes. Since

there are multiple translations, they map multiple external addresses (prefix and IID) to the common internal address. A system within the edge network is unable to determine which external address it is using apart from services such as Session Traversal Utilities for NAT (STUN) [RFC5389].

Multihoming in this sense has one negative feature as compared with multihoming with a provider-independent address: when routes change between NPTv6 Translators, the translated prefix can change since the upstream network changes. This causes sessions and referrals dependent on it to fail as well. This is not expected to be a major issue, however, in networks where routing is generally stable.

2.5. Mapping with No Per-Flow State

When NPTv6 is used as described in this document, no per-node or per-flow state is maintained in the NPTv6 Translator. Both inbound and outbound datagrams are translated algorithmically, using only information found in the IPv6 header. Due to this property, NPTv6's two-way, algorithmic address mapping can support both outbound and inbound connection establishment without the need for maintenance of mapping state or for state-priming or rendezvous mechanisms. This is a significant improvement over NAPT44 devices, but it also has significant security implications, which are described in Section 7.

2.6. Checksum-Neutral Mapping

When a change is made to one of the IP header fields in the IPv6 pseudo-header checksum (such as one of the IP addresses), the checksum field in the transport layer header may become invalid. Fortunately, an incremental change in the area covered by the Internet standard checksum [RFC1071] will result in a well-defined change to the checksum value [RFC1624]. So, a checksum change caused by modifying part of the area covered by the checksum can be corrected by making a complementary change to a different 16-bit field covered by the same checksum.

The NPTv6 mapping mechanisms described in this document are checksum-neutral, which means that they result in IP headers that will generate the same IPv6 pseudo-header checksum when the checksum is calculated using the standard Internet checksum algorithm [RFC1071]. Any changes that are made during translation of the IPv6 prefix are offset by changes to other parts of the IPv6 address. This results in transport layers that use the Internet checksum (such as TCP and UDP) calculating the same IPv6 pseudo-header checksum for both the internal and external forms of the same datagram, which avoids the need for the NPTv6 Translator to modify those transport layer headers to correct the checksum value.

The outgoing checksum correction is achieved by making a change to a 16-bit section of the source address that is not used for routing in the external network. Due to the nature of checksum arithmetic, when the corresponding correction is applied to the same bits of destination address of the inbound packet, the Destination Address (DA) is returned to the correct internal value.

As noted in Section 4.2, this mapping results in an edge network using a /48 external prefix to be unable to use subnet 0xFFFF.

3. NPTv6 Algorithmic Specification

The [RFC4291] IPv6 Address is reproduced for clarity in Figure 5.

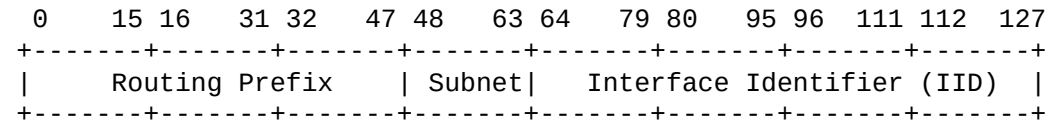

Figure 5: Enumeration of the IPv6 Address [RFC4291]

3.1. NPTv6 Configuration Calculations

When an NPTv6 Translation function is configured, it is configured with

- one or more "internal" interfaces with their "internal" routing domain prefixes, and
- one or more "external" interfaces with their "external" routing domain prefixes.

In the simple case, there is one of each. If a single router provides NPTv6 translation services between a multiplicity of domains (as might be true when multihoming), each internal/external pair must be thought of as a separate NPTv6 Translator from the perspective of this specification.

When an NPTv6 Translator is configured, the translation function first ensures that the internal and external prefixes are the same length, extending the shorter of the two with zeroes if necessary. These two prefixes will be used in the prefix translation function described in Sections 3.2 and 3.3.

They are then zero-extended to /64 for the purposes of a calculation. The translation function calculates the one’s complement sum of the 16-bit words of the /64 external prefix and the /64 internal prefix. It then calculates the difference between these values: internal

minus external. This value, called the "adjustment", is effectively constant for the lifetime of the NPTv6 Translator configuration and is used in per-datagram processing.

3.2. NPTv6 Translation, Internal Network to External Network

When a datagram passes through the NPTv6 Translator from an internal to an external network, its IPv6 Source Address is either changed in two ways or results in the datagram being discarded:

- If the internal subnet number has no mapping, such as being 0xFFFF or simply not mapped, discard the datagram. This SHOULD result in an ICMP Destination Unreachable.

- The internal prefix is overwritten with the external prefix, in effect subtracting the difference between the two checksums (the adjustment) from the pseudo-header's checksum, and

- A 16-bit word of the address has the adjustment added to it using one's complement arithmetic. If the result is 0xFFFF, it is overwritten as zero. The choice of word is as specified in Sections 3.4 or 3.5 as appropriate.

3.3. NPTv6 Translation, External Network to Internal Network

When a datagram passes through the NPTv6 Translator from an external to an internal network, its IPv6 Destination Address is changed in two ways:

- The external prefix is overwritten with the internal prefix, in effect adding the difference between the two checksums (the adjustment) to the pseudo-header's checksum, and

- A 16-bit word of the address has the adjustment subtracted from it (bitwise inverted and added to it) using one's complement arithmetic. If the result is 0xFFFF, it is overwritten as zero. The choice of word is as specified in Section 3.4 or Section 3.5 as appropriate.

3.4. NPTv6 with a /48 or Shorter Prefix

When an NPTv6 Translator is configured with internal and external prefixes that are 48 bits in length (a /48) or shorter, the adjustment MUST be added to or subtracted from bits 48..63 of the address.

This mapping results in no modification of the Interface Identifier (IID), which is held in the lower half of the IPv6 address, so it will not interfere with future protocols that may use unique IIDs for node identification.

NPTv6 Translator implementations MUST implement the /48 mapping.

3.5. NPTv6 with a /49 or Longer Prefix

When an NPTv6 Translator is configured with internal and external prefixes that are longer than 48 bits in length (such as a /52, /56, or /60), the adjustment must be added to or subtracted from one of the words in bits 64..79, 80..95, 96..111, or 112..127 of the address. While the choice of word is immaterial as long as it is consistent, these words MUST be inspected in that sequence and the first that is not initially 0xFFFF chosen, for consistency's sake.

NPTv6 Translator implementations SHOULD implement the mapping for longer prefixes.

3.6. /48 Prefix Mapping Example

For the network shown in Figure 1, the Internal Prefix is FD01:0203:0405:/48, and the External Prefix is 2001:0DB8:0001:/48.

If a node with internal address FD01:0203:0405:0001::1234 sends an outbound datagram through the NPTv6 Translator, the resulting external address will be 2001:0DB8:0001:D550::1234. The resulting address is obtained by calculating the checksum of both the internal and external 48-bit prefixes, subtracting the internal prefix from the external prefix using one's complement arithmetic to calculate the "adjustment", and adding the adjustment to the 16-bit subnet field (in this case, 0x0001).

To show the work:

The one's complement checksum of FD01:0203:0405 is 0xFCF5. The one's complement checksum of 2001:0DB8:0001 is 0xD245. Using one's complement arithmetic, 0xD245 - 0xFCF5 = 0xD54F. The subnet in the original datagram is 0x0001. Using one's complement arithmetic, 0x0001 + 0xD54F = 0xD550. Since 0xD550 != 0xFFFF, it is not changed to 0x0000.

So, the value 0xD550 is written in the 16-bit subnet area, resulting in a mapped external address of 2001:0DB8:0001:D550::1234.

When a response datagram is received, it will contain the destination address 2001:0DB8:0001:D550::0001, which will be mapped back to FD01:0203:0405:0001::1234 using the inverse mapping algorithm.

In this case, the difference between the two prefixes will be calculated as follows:

Using one's complement arithmetic, 0xFCF5 - 0xD245 = 0x2AB0. The subnet in the original datagram = 0xD550. Using one's complement arithmetic, 0xD550 + 0x2AB0 = 0x0001. Since 0x0001 != 0xFFFF, it is not changed to 0x0000.

So the value 0x0001 is written into the subnet field, and the internal value of the subnet field is properly restored.

3.7. Address Mapping for Longer Prefixes

If the prefix being mapped is longer than 48 bits, the algorithm is slightly more complex. A common case will be that the internal and external prefixes are of different lengths. In such a case, the shorter prefix is zero-extended to the length of the longer as described in Section 3.1 for the purposes of overwriting the prefix. Then, they are both zero-extended to 64 bits to facilitate one's complement arithmetic. The "adjustment" is calculated using those 64-bit prefixes.

For example, if the internal prefix is a /48 ULA and the external prefix is a /56 provider-allocated prefix, the ULA becomes a /56 with zeros in bits 48..55. For purposes of one's complement arithmetic, they are then both zero-extended to 64 bits. A side effect of this is that a subset of the subnets possible in the shorter prefix is untranslatable. While the security value of this is debatable, the administration may choose to use them for subnets that it knows need no external accessibility.

We then find the first word in the IID that does not have the value 0xFFFF, trying bits 64..79, and then 80..95, 96..111, and finally 112..127. We perform the same calculation (with the same proof of correctness) as in Section 3.6 but apply it to that word.

Although any 16-bit portion of an IPv6 IID could contain 0xFFFF, an IID of all-ones is a reserved anycast identifier that should not be used on the network [RFC2526]. If an NPTv6 Translator discovers a datagram with an IID of all-zeros while performing address mapping, that datagram MUST be dropped, and an ICMPv6 Parameter Problem error SHOULD be generated [RFC4443].

Note: This mechanism does involve modification of the IID; it may not be compatible with future mechanisms that use unique IIDs for node identification.

4. Implications of Network Address Translator Behavioral Requirements

4.1. Prefix Configuration and Generation

NPTv6 Translators MUST support manual configuration of internal and external prefixes and MUST NOT place any restrictions on those prefixes except that they be valid IPv6 unicast prefixes as described in [RFC4291]. They MAY also support random generation of ULA addresses on command. Since the most common place anticipated for the implementation of an NPTv6 Translator is a Customer Premises Equipment (CPE) router, the reader is urged to consider the requirements of [RFC6204].

4.2. Subnet Numbering

For reasons detailed in Appendix B, a network using NPTv6 Translation and a /48 external prefix MUST NOT use the value 0xFFFF to designate a subnet that it expects to be translated.

4.3. NAT Behavioral Requirements

NPTv6 Translators MUST support hairpinning behavior, as defined in the NAT Behavioral Requirements for UDP document [RFC4787]. This means that when an NPTv6 Translator receives a datagram on the internal interface that has a destination address that matches the site's external prefix, it will translate the datagram and forward it internally. This allows internal nodes to reach other internal nodes using their external, global addresses when necessary.

Conceptually, the datagram leaves the domain (is translated as described in Section 3.2) and returns (is again translated as described in Section 3.3). As a result, the datagram exchange will be through the NPTv6 Translator in both directions for the lifetime of the session. The alternative would be to require the NPTv6 Translator to drop the datagram, forcing the sender to use the correct internal prefix for its peer. Performing only the external-to-internal translation results in the datagram being sent from the untranslated internal address of the source to the translated and therefore internal address of its peer, which would enable the session to bypass the NPTv6 Translator for future datagrams. It would also mean that the original sender would be unlikely to recognize the response when it arrived.

Because NPTv6 does not perform port mapping and uses a one-to-one, reversible-mapping algorithm, none of the other NAT behavioral requirements apply to NPTv6.

5. Implications for Applications

NPTv6 Translation does not create several of the problems known to exist with other kinds of NATs as discussed in [RFC2993]. In particular, NPTv6 Translation is stateless, so a "reset" or brief outage of an NPTv6 Translator does not break connections that traverse the translation function, and if multiple NPTv6 Translators exist between the same two networks, the load can shift or be dynamically load shared among them. Also, an NPTv6 Translator does not aggregate traffic for several hosts/interfaces behind a fewer number of external addresses, so there is no inherent expectation for an NPTv6 Translator to block new inbound flows from external hosts and no issue with a filter or blacklist associated with one prefix within the domain affecting another. A firewall can, of course, be used in conjunction with an NPTv6 Translator; this would allow the network administrator more flexibility to specify security policy than would be possible with a traditional NAT.

However, NPTv6 Translation does create difficulties for some kinds of applications. Some examples include:

- An application instance "behind" an NPTv6 Translator will see a different address for its connections than its peers "outside" the NPTv6 Translator.

- An application instance "outside" an NPTv6 Translator will see a different address for its connections than any peer "inside" an NPTv6 Translator.

- An application instance wishing to establish communication with a peer "behind" an NPTv6 Translator may need to use a different address to reach that peer depending on whether the instance is behind the same NPTv6 Translator or external to it. Since an NPTv6 Translator implements hairpinning (Section 4.3), it suffices for applications to always use their external addresses. However, this creates inefficiencies in the local network and may also complicate implementation of the NPTv6 Translator. [RFC3484] also would prefer the private address in such a case in order to reduce those inefficiencies.

- An application instance that moves from a realm "behind" an NPTv6 Translator to a realm that is "outside" the network, or vice versa, may find that it is no longer able to reach its peers at the same addresses it was previously able to use.

- o An application instance that is intermittently communicating with a peer that moves from behind an NPTv6 Translator to "outside" of it, or vice versa, may find that it is no longer able to reach that peer at the same address that it had previously used.

Many, but not all, of the applications that are adversely affected by NPTv6 Translation are those that do "referrals" -- where an application instance passes its own addresses, and/or addresses of its peers, to other peers. (Some believe referrals are inherently undesirable; others believe that they are necessary in some circumstances. A discussion of the merits of referrals, or lack thereof, is beyond the scope of this document.)

To some extent, the incidence of these difficulties can be reduced by DNS hacks that attempt to expose addresses "behind" an NPTv6 Translator only to hosts that are also behind the same NPTv6 Translator and perhaps to also expose only the "internal" addresses of hosts behind the NPTv6 Translator to other hosts behind the same NPTv6 Translator. However, this cannot be a complete solution. A full discussion of these issues is out of scope for this document, but briefly: (a) reliance on DNS to solve this problem depends on hosts always making queries from DNS servers in the same realm as they are (or on DNS interception proxies, which create their own problems) and on mobile hosts/applications not caching those results; (b) reliance on DNS to solve this problem depends on network administrators on all networks using such applications to reliably and accurately maintain current DNS entries for every host using those applications; and (c) reliance on DNS to solve this problem depends on applications always using DNS names, even though they often must run in environments where DNS names are not reliably maintained for every host. Other issues are that there is often no single distinguished name for a host and no reliable way for a host to determine what DNS names are associated with it and which names are appropriate to use in which contexts.

5.1. Recommendation for Network Planners Considering Use of NPTv6 Translation

In light of the above, network planners considering the use of NPTv6 translation should carefully consider the kinds of applications that they will need to run in the future and determine whether the address-stability and provider-independence benefits are consistent with their application requirements.

5.2. Recommendations for Application Writers

Several mechanisms (e.g., STUN [RFC5389], Traversal Using Relays around NAT (TURN) [RFC5766], and Interactive Connectivity Establishment (ICE) [RFC5245]) have been used with traditional IPv4 NAT to circumvent some of the limitations of such devices. Similar mechanisms could also be applied to circumvent some of the issues with an NPTv6 Translator. However, all of these require the assistance of an external server or a function co-located with the translator that can tell an "internal" host what its "external" addresses are.

5.3. Recommendation for Future Work

It might be desirable to define a general mechanism that would allow hosts within a translation domain to determine their external addresses and/or request that inbound traffic be permitted. If such a mechanism were to be defined, it would ideally be general enough to also accommodate other types of NAT likely to be encountered by IPV6 applications, in particular IPv4/IPv6 Translation [RFC6144] [RFC6147] [RFC6145] [RFC6146] [RFC6052]. For this and other reasons, such a mechanism is beyond the scope of this document.

6. A Note on Port Mapping

In addition to overwriting IP addresses when datagrams are forwarded, NAPT44 devices overwrite the source port number in outbound traffic and the destination port number in inbound traffic. This mechanism is called "port mapping".

The major benefit of port mapping is that it allows multiple computers to share a single IPv4 address. A large number of internal IPv4 addresses (typically from one of the [RFC1918] private address spaces) can be mapped into a single external, globally routable IPv4 address, with the local port number used to identify which internal node should receive each inbound datagram. This address-amplification feature is not generally foreseen as a necessity at this time.

Since port mapping requires rewriting a portion of the transport layer header, it requires NAPT44 devices to be aware of all of the transport protocols that they forward, thus stifling the development of new and improved transport protocols and preventing the use of IPsec encryption. Modifying the transport layer header is incompatible with security mechanisms that encrypt the full IP payload and restricts the NAPT44 to forwarding transport layers that use weak checksum algorithms that are easily recalculated in routers.

Since there is significant detriment caused by modifying transport layer headers and very little, if any, benefit to the use of port mapping in IPv6, NPTv6 Translators that comply with this specification MUST NOT perform port mapping.

7. Security Considerations

When NPTv6 is deployed using either of the two-way, algorithmic mappings defined in this document, it allows direct inbound connections to internal nodes. While this can be viewed as a benefit of NPTv6 versus NAPT44, it does open internal nodes to attacks that would be more difficult in a NAPT44 network. From a security standpoint, although this situation is not substantially worse than running IPv6 with no NAT, some enterprises may assume that an NPTv6 Translator will offer similar protection to a NAPT44 device.

The port mapping mechanism in NAPT44 implementations requires that state be created in both directions. This has lead to an industry-wide perception that NAT functionality is the same as a stateful firewall. It is not. The translation function of the NAT only creates dynamic state in one direction and has no policy. For this reason, it is RECOMMENDED that NPTv6 Translators also implement firewall functionality such as described in [RFC6092], with appropriate configuration options including turning it on or off.

When [RFC4864] talks about randomizing the subnet identifier, the idea is to make it harder for worms to guess a valid subnet identifier at an advertised network prefix. This should not be interpreted as endorsing concealment of the subnet identifier behind the obfuscating function of a translator such as NPTv6. [RFC4864] specifically talks about how to obtain the desired properties of concealment without using a translator. Topology hiding when using NAT is often ineffective in environments where the topology is visible in application layer messaging protocols such as DNS, SIP, SMTP, etc. If the information were not available through the application layer, [RFC2993] would not be valid.

Due to the potential interactions with IKEv2/IPsec NAT traversal, it would be valuable to test interactions of NPTv6 with various aspects of current-day IKEv2/IPsec NAT traversal.

8. Acknowledgements

The checksum-neutral algorithmic address mapping described in this document is based on email written by Iljtsch van Beijnum.

The following people provided advice or review comments that substantially improved this document: Allison Mankin, Christian Huitema, Dave Thaler, Ed Jankiewicz, Eric Kline, Iljtsch van Beijnum, Jari Arkko, Keith Moore, Mark Townsley, Merike Kaeo, Ralph Droms, Remi Despres, Steve Blake, and Tony Hain.

9. References

9.1. Normative References

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

[RFC2526] Johnson, D. and S. Deering, "Reserved IPv6 Subnet Anycast Addresses", RFC 2526, March 1999.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

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

9.2. Informative References

[GSE] O'Dell, M., "GSE - An Alternate Addressing Architecture for IPv6", Work in Progress, February 1997.

[NIST] NIST, "Draft NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 1.0", September 2009.

[RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.

[RFC1624] Rijsinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, May 2000.

[RFC2993] Hain, T., "Architectural Implications of NAT", RFC 2993, November 2000.

[RFC3424] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, November 2002.

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

[RFC4864] Van de Velde, G., Hain, T., Droms, R., Carpenter, B., and E. Klein, "Local Network Protection for IPv6", RFC 4864, May 2007.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC5902] Thaler, D., Zhang, L., and G. Lebovitz, "IAB Thoughts on IPv6 Network Address Translation", RFC 5902, July 2010.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

[RFC6052] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", RFC 6052, October 2010.

[RFC6092] Woodyatt, J., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, January 2011.

[RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, April 2011.

[RFC6145] Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm", RFC 6145, April 2011.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, April 2011.

[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, April 2011.

[RFC6204] Singh, H., Beebee, W., Donley, C., Stark, B., and O. Troan, "Basic Requirements for IPv6 Customer Edge Routers", RFC 6204, April 2011.

Appendix A. Why GSE?

For the purpose of this discussion, let us oversimplify the Internet's structure by distinguishing between two broad classes of networks: transit and edge. A "transit network", in this context, is a network that provides connectivity services to other networks. Its Autonomous System (AS) number may show up in a non-final position in BGP AS paths, or in the case of mobile and residential broadband networks, it may offer network services to smaller networks that cannot justify RIR membership. An "edge network", in contrast, is any network that is not a transit network; it is the ultimate customer, and while it provides internal connectivity for its own use, it is a consumer of transit services in other respects. In terms of routing, a network in the transit domain generally needs some way to make choices about how it routes to other networks; an edge network is generally quite satisfied with a simple default route.

The [GSE] proposal, and as a result this proposal (which is similar to GSE in most respects and inspired by it), responds directly to current concerns in the RIR communities. Edge networks are used to an environment in IPv4 in which their addressing is disjoint from that of their upstream transit networks; it is either provider independent, or a network prefix translator makes their external address distinct from their internal address, and they like the distinction. In IPv6, there is a mantra that edge network addresses should be derived from their upstream, and if they have multiple upstreams, edge networks are expected to design their networks to use all of those prefixes equivalently. They see this as unnecessary and unwanted operational complexity and, as a result, are pushing very hard in the RIR communities for provider-independent addressing.

Widespread use of provider-independent addressing has a natural and perhaps unavoidable side effect that is likely to be very expensive in the long term. With widespread PI addressing, the routing table will enumerate the networks at the edge of the transit domain, the edge networks, rather than enumerate the transit domain. Per the BGP Update Report of 17 December 2010, there are currently over 36,000 Autonomous Systems being advertised in BGP, of which over 15,000 advertise only one prefix. There are in the neighborhood of 5000 ASs that show up in a non-final position in AS paths, and perhaps another 5000 networks whose AS numbers are terminal in more than one AS path. In other words, we have prefixes for some 36,000 transit and edge networks in the route table now, many of which arguably need an Autonomous System number only for multihoming. The vast majority of networks (2/3) having the tools necessary to multihome are not

visibly doing so and would be well served by any solution that gives them address independence without the overhead of RIR membership and BGP routing.

Current growth estimates suggest that we could easily see that be on the order of 10,000,000 within fifteen years. Tens of thousands of entries in the route table are very survivable; while our protocols and computers will likely do quite well with tens of millions of routes, the heat produced and power consumed by those routers, and the inevitable impact on the cost of those routers, is not a good outcome. To avoid having a massive and unscalable route table, we need to find a way that is politically acceptable and returns us to enumerating the transit domain, not the edge.

There have been a number of proposals. As described, Shim6 moves the complexity to the edge, and the edge is rebelling. Geographic addressing in essence forces ISPs to "own" geographic territory from a routing perspective, as otherwise there is no clue in the address as to what network a datagram should be delivered to in order to reach it. Metropolitan Addressing can imply regulatory authority and, even if it is implemented using internet exchange consortia, visits a great deal of complexity on the transit networks that directly serve the edge. The one that is likely to be most acceptable is any proposal that enables an edge network to be operationally independent of its upstreams, with no obligation to renumber when it adds, drops, or changes ISPs, and with no additional burden placed either on the ISP or the edge network as a result. From an application perspective, an additional operational requirement in the words of the Roadmap for the Smart Grid [NIST] is that

> "...the network should provide the capability to enable an application in a particular domain to communicate with an application in any other domain over the information network, with proper management control as to who and where applications can be inter-connected."

In other words, the structure of the network should allow for and enable appropriate access control, but the structure of the network should not inherently limit access.

The GSE model, by statelessly translating the prefix between an edge network and its upstream transit network, accomplishes that with a minimum of fuss and bother. Stated in the simplest terms, it enables the edge network to behave as if it has a provider-independent prefix from a multihoming and renumbering perspective without the overhead of RIR membership or maintenance of BGP connectivity, and it enables

the transit networks to aggressively aggregate what are from their perspective provider-allocated customer prefixes, to maintain a rational-sized routing table.

Appendix B. Verification Code

This non-normative appendix is presented as a proof of concept; it is in no sense optimized. For example, one's complement arithmetic is implemented in portable subroutines, where operational implementations might use one's complement arithmetic instructions through a pragma; such implementations probably need to explicitly force 0xFFFF to 0x0000, as the instruction will not. The original purpose of the code was to verify whether or not it was necessary to suppress 0xFFFF by overwriting with zero and whether predicted issues with subnet numbering were real.

The point is to

- demonstrate that if one or the other representation of zero is not used in the word in which the checksum is updated, the program maps inner and outer addresses in a manner that is, mathematically, 1:1 and onto (each inner address maps to a unique outer address, and that outer address maps back to exactly the same inner address), and

- give guidance on the suppression of 0xFFFF checksums.

In short, in one's complement arithmetic, x-x=0 but will take the negative representation of zero. If 0xFFFF results are forced to the value 0x0000, as is recommended in [RFC1071], the word the checksum is adjusted in cannot be initially 0xFFFF, as on the return it will be forced to 0. If 0xFFFF results are not forced to the value 0x0000 as is recommended in [RFC1071], the word the checksum is adjusted in cannot be initially 0, as on the return it will be calculated as 0+(~0) = 0xFFFF. We chose to follow [RFC1071]'s recommendations, which implies a requirement to not use 0xFFFF as a subnet number in networks with a /48 external prefix.

```
/*
 * Copyright (c) 2011 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
```

```
 *
 * - Redistributions in binary form must reproduce the above
 *   copyright notice, this list of conditions and the following
 *   disclaimer in the documentation and/or other materials provided
 *   with the distribution.
 *
 * - Neither the name of Internet Society, IETF or IETF Trust, nor
 *   the names of specific contributors, may be used to endorse or
 *   promote products derived from this software without specific
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
 * CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
#include "stdio.h"
#include "assert.h"
/*
 * program to verify the NPTv6 algorithm
 *
 * argument:
 *    Perform negative zero suppression: boolean
 *
 * method:
 *    We specify an internal and an external prefix.  The prefix
 *    length is presumed to be the common length of both and, for
 *    this, is a /48.  We perform the three algorithms specified.
 *    The "datagram" address is in effect the source address
 *    internal->external and the destination address
 *    external->internal.
 */
unsigned short  inner_init[] = {
    0xFD01, 0x0203, 0x0405, 1, 2, 3, 4, 5};
unsigned short  outer_init[] = {
    0x2001, 0x0db8, 0x0001, 1, 2, 3, 4, 5};
unsigned short  inner[8];
unsigned short  datagram[8];
unsigned char   checksum[65536] = {0};
```

```
  unsigned short  outer[8];
  unsigned short  adjustment;
  unsigned short  suppress;
  /*
   * One's complement sum.
   * return number1 + number2
   */
  unsigned short
  add1(number1, number2)
      unsigned short  number1;
      unsigned short  number2;
  {
      unsigned int    result;

      result = number1;
      result += number2;
      if (suppress) {
          while (0xFFFF <= result) {
              result = result + 1 - 0x10000;
          }
      } else {
          while (0xFFFF < result) {
              result = result + 1 - 0x10000;
          }
      }
      return result;
  }

  /*
   * One's complement difference
   * return number1 - number2
   */
  unsigned short
  sub1(number1, number2)
      unsigned short  number1;
      unsigned short  number2;
  {
      return add1(number1, ~number2);
  }

  /*
   * return one's complement sum of an array of numbers
   */
  unsigned short
  sum1(numbers, count)
      unsigned short *numbers;
      int             count;
  {
```

```
    unsigned int    result;

    result = *numbers++;
    while (--count > 0) {
        result += *numbers++;
    }

    if (suppress) {
        while (0xFFFF <= result) {
            result = result + 1 - 0x10000;
        }
    } else {
        while (0xFFFF < result) {
            result = result + 1 - 0x10000;
        }
    }
    return result;
}

/*
 * NPTv6 initialization: Section 3.1 assuming Section 3.4
 *
 * Create the /48, a source address in internal format, and a
 * source address in external format.  Calculate the adjustment
 * if one /48 is overwritten with the other.
 */
void
nptv6_initialization(subnet)
    unsigned short  subnet;
{
    int             i;
    unsigned short  inner48;
    unsigned short  outer48;

    /* Initialize the internal and external prefixes. */
    for (i = 0; i < 8; i++) {
        inner[i] = inner_init[i];
        outer[i] = outer_init[i];
    }
    inner[3] = subnet;
    outer[3] = subnet;
    /* Calculate the checksum adjustment. */
    inner48 = sum1(inner, 3);
    outer48 = sum1(outer, 3);
    adjustment = sub1(inner48, outer48);
}

/*
```

```
 * NPTv6 datagram from edge to transit: Section 3.2 assuming
 * Section 3.4
 *
 * Overwrite the prefix in the source address with the outer
 * prefix and adjust the checksum.
 */
void
nptv6_inner_to_outer()
{
    int             i;

    /* Let's get the source address into the datagram. */
    for (i = 0; i < 8; i++) {
        datagram[i] = inner[i];
    }

    /* Overwrite the prefix with the outer prefix. */
    for (i = 0; i < 3; i++) {
        datagram[i] = outer[i];
    }

    /* Adjust the checksum. */
    datagram[3] = add1(datagram[3], adjustment);
}

/*
 * NPTv6 datagram from transit to edge: Section 3.3 assuming
 * Section 3.4
 *
 * Overwrite the prefix in the destination address with the
 * inner prefix and adjust the checksum.
 */
void
nptv6_outer_to_inner()
{
    int             i;

    /* Overwrite the prefix with the outer prefix. */
    for (i = 0; i < 3; i++) {
        datagram[i] = inner[i];
    }

    /* Adjust the checksum. */
    datagram[3] = sub1(datagram[3], adjustment);
}

/*
 * Main program
```

```
 */
main(argc, argv)
    int             argc;
    char          **argv;
{
    unsigned        subnet;
    int             i;

    if (argc < 2) {
           fprintf(stderr, "usage: nptv6 supression\n");
           assert(0);
       }
       suppress = atoi(argv[1]);
       assert(suppress <= 1);

       for (subnet = 0; subnet < 0x10000; subnet++) {
           /* Section 3.1: initialize the system */
           nptv6_initialization(subnet);

           /* Section 3.2: take a datagram from inside to outside */
           nptv6_inner_to_outer();

           /* The resulting checksum value should be unique. */
           if (checksum[subnet]) {
                printf("inner->outer duplicated checksum: "
                       "inner: %x:%x:%x:%x:%x:%x:%x:%x(%x) "
                       "calculated: %x:%x:%x:%x:%x:%x:%x:%x(%x)\n",
                       inner[0], inner[1], inner[2], inner[3],
                       inner[4], inner[5], inner[6], inner[7],
                       sum1(inner, 8), datagram[0], datagram[1],
                       datagram[2], datagram[3], datagram[4],
                       datagram[5], datagram[6], datagram[7],
                       sum1(datagram, 8));
        }

        checksum[subnet] = 1;

        /*
         * The resulting checksum should be the same as the inner
         * address's checksum.
         */
        if (sum1(datagram, 8) != sum1(inner, 8)) {
            printf("inner->outer incorrect: "
                   "inner: %x:%x:%x:%x:%x:%x:%x:%x(%x) "
                   "calculated: %x:%x:%x:%x:%x:%x:%x:%x(%x)\n",
                   inner[0], inner[1], inner[2], inner[3],
                   inner[4], inner[5], inner[6], inner[7],
                   sum1(inner, 8),
```

```
                    datagram[0], datagram[1], datagram[2], datagram[3],
                    datagram[4], datagram[5], datagram[6], datagram[7],
                    sum1(datagram, 8));
            }

            /* Section 3.3: take a datagram from outside to inside */
            nptv6_outer_to_inner();

            /*
             * The returning datagram should have the same checksum it
             * left with.
             */
            if (sum1(datagram, 8) != sum1(inner, 8)) {
                printf("outer->inner checksum incorrect: "
                    "calculated: %x:%x:%x:%x:%x:%x:%x:%x(%x) "
                    "inner: %x:%x:%x:%x:%x:%x:%x:%x(%x)\n",
                    datagram[0], datagram[1], datagram[2], datagram[3],
                    datagram[4], datagram[5], datagram[6], datagram[7],
                    sum1(datagram, 8), inner[0], inner[1], inner[2],
                    inner[3], inner[4], inner[5], inner[6], inner[7],
                    sum1(inner, 8));
            }

            /*
             * And every octet should calculate back to the same inner
             * value.
             */
            for (i = 0; i < 8; i++) {
                if (inner[i] != datagram[i]) {
                    printf("outer->inner different: "
                        "calculated: %x:%x:%x:%x:%x:%x:%x:%x "
                        "inner: %x:%x:%x:%x:%x:%x:%x:%x\n",
                        datagram[0], datagram[1], datagram[2],
                        datagram[3], datagram[4], datagram[5],
                        datagram[6], datagram[7], inner[0], inner[1],
                        inner[2], inner[3], inner[4], inner[5],
                        inner[6], inner[7]);
                    break;
                }
            }
        }
    }
```

Authors' Addresses

Margaret Wasserman
Painless Security
North Andover, MA 01845
USA

Phone: +1 781 405 7464
EMail: mrw@painless-security.com
URI: http://www.painless-security.com

Fred Baker
Cisco Systems
Santa Barbara, California 93117
USA

Phone: +1-408-526-4257
EMail: fred@cisco.com