Network Working Group
Request for Comments: 2765
Category: Standards Track

E. Nordmark
Sun Microsystems

February 2000

Stateless IP/ICMP Translation Algorithm (SIIT)

Status of this Memo

Copyright Notice

Abstract

This document specifies a transition mechanism algorithm in addition to the mechanisms already specified in [TRANS-MECH]. The algorithm translates between IPv4 and IPv6 packet headers (including ICMP headers) in separate translator "boxes" in the network without requiring any per-connection state in those "boxes". This new algorithm can be used as part of a solution that allows IPv6 hosts, which do not have a permanently assigned IPv4 addresses, to communicate with IPv4-only hosts. The document neither specifies address assignment nor routing to and from the IPv6 hosts when they communicate with the IPv4-only hosts.

Acknowledgements

This document is a product of the NGTRANS working group. Some text has been extracted from an old Internet Draft titled "IPAE: The SIPP Interoperability and Transition Mechanism" authored by R. Gilligan, E. Nordmark, and B. Hinden. George Tsirtsis provides the figures for Section 1. Keith Moore provided a careful review of the document.

Table of Contents

1. Introduction and Motivation

The transition mechanisms specified in [TRANS-MECH] handle the case of dual IPv4/IPv6 hosts interoperating with both dual hosts and IPv4-only hosts, which is needed early in the transition to IPv6. The dual hosts are assigned both an IPv4 and one or more IPv6 addresses. As the number of available globally unique IPv4 addresses becomes smaller and smaller as the Internet grows there will be a desire to take advantage of the large IPv6 address and not require that every new Internet node have a permanently assigned IPv4 address.

There are several different scenarios where there might be IPv6-only hosts that need to communicate with IPv4-only hosts. These IPv6 hosts might be IPv4-capable, i.e. include an IPv4 implementation but not be assigned an IPv4 address, or they might not even include an IPv4 implementation.

- A completely new network with new devices that all support IPv6. In this case it might be beneficial to not have to configure the routers within the new network to route IPv4 since none of the

hosts in the new network are configured with IPv4 addresses. But these new IPv6 devices might occasionally need to communicate with some IPv4 nodes out on the Internet.

- An existing network where a large number of IPv6 devices are added. The IPv6 devices might have both an IPv4 and an IPv6 protocol stack but there is not enough global IPv4 address space to give each one of them a permanent IPv4 address. In this case it is more likely that the routers in the network already route IPv4 and are upgraded to dual routers.

However, there are other potential solutions in this area:

- If there is no IPv4 routing inside the network i.e., the cloud that contains the new devices, some possible solutions are to either use the translators specified in this document at the boundary of the cloud, or to use Application Layer Gateways (ALG) on dual nodes at the cloud's boundary. The ALG solution is less flexible in that it is application protocol specific and it is also less robust since an ALG box is likely to be a single point of failure for a connection using that box.

- Otherwise, if IPv4 routing is supported inside the cloud and the implementations support both IPv6 and IPv4 it might suffice to have a mechanism for allocating a temporary address IPv4 and use IPv4 end to end when communicating with IPv4-only nodes. However, it would seem that such a solution would require the pool of temporary IPv4 addresses to be partitioned across all the subnets in the cloud which would either require a larger pool of IPv4 addresses or result in cases where communication would fail due to no available IPv4 address for the node's subnet.

This document specifies an algorithm that is one of the components needed to make IPv6-only nodes interoperate with IPv4-only nodes. Other components, not specified in this document, are a mechanism for the IPv6-only node to somehow acquire a temporary IPv4 address, and a mechanism for providing routing (perhaps using tunneling) to and from the temporary IPv4 address assigned to the node.

The temporary IPv4 address will be used as an IPv4-translated IPv6 address and the packets will travel through a stateless IP/ICMP translator that will translate the packet headers between IPv4 and IPv6 and translate the addresses in those headers between IPv4 addresses on one side and IPv4-translated or IPv4-mapped IPv6 addresses on the other side.

This specification does not cover how an IPv6 node can acquire a temporary IPv4 address and how such a temporary address be registered in the DNS. The DHCP protocol, perhaps with some extensions, could probably be used to acquire temporary addresses with short leases but that is outside the scope of this document. Also, the mechanism for routing this IPv4-translated IPv6 address in the site is not specified in this document.

The figures below show how the Stateless IP/ICMP Translation algorithm (SIIT) can be used initially for small networks (e.g., a single subnet) and later for a site which has IPv6-only hosts in a dual IPv4/IPv6 network. This use assumes a mechanism for the IPv6 nodes to acquire a temporary address from the pool of IPv4 addresses. Note that SIIT is not likely to be useful later during transition when most of the Internet is IPv6 and there are only small islands of IPv4 nodes, since such use would either require the IPv6 nodes to acquire temporary IPv4 addresses from a "distant" SIIT box operated by a different administration, or require that the IPv6 routing contain routes for IPv6-mapped addresses. (The latter is known to be a very bad idea due to the size of the IPv4 routing table that would potentially be injected into IPv6 routing in the form of IPv4-mapped addresses.)

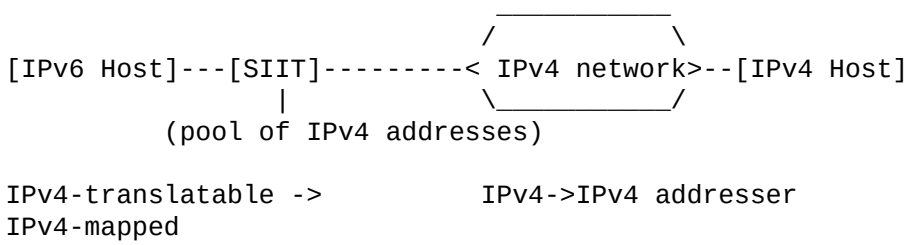


Figure 1. Using SIIT for a single IPv6-only subnet.

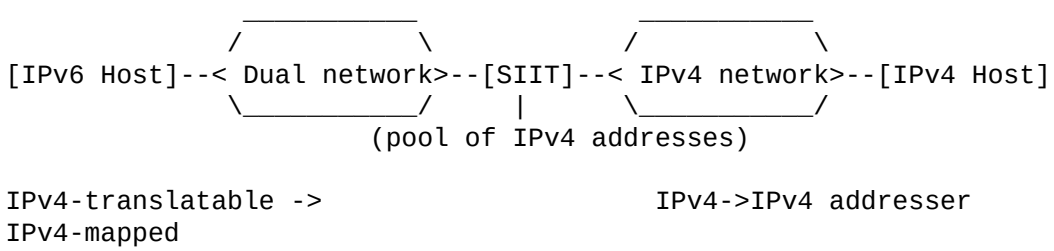


Figure 2. Using SIIT for an IPv6-only or dual cloud (e.g. a site) which contains some IPv6-only hosts as well as IPv4 hosts.

The protocol translators are assumed to fit around some piece of topology that includes some IPv6-only nodes and that may also include IPv4 nodes as well as dual nodes. There has to be a translator on each path used by routing the "translatable" packets in and out of this cloud to ensure that such packets always get translated. This does not require a translator at every physical connection between the cloud and the rest of the Internet since the routing can be used to deliver the packets to the translator.

The IPv6-only node communicating with an IPv4 node through a translator will see an IPv4-mapped address for the peer and use an IPv4-translatable address for its local address for that communication. When the IPv6-only node sends packets the IPv4-mapped address indicates that the translator needs to translate the packets. When the IPv4 node sends packets those will translated to have the IPv4-translatable address as a destination; it is not possible to use an IPv4-mapped or an IPv4-compatible address as a destination since that would either route the packet back to the translator (for the IPv4-mapped address) or make the packet be encapsulated in IPv4 (for the IPv4-compatible address). Thus this specification introduces the new notion of an IPv4-translatable address.

1.1. Applicability and Limitations

The use of this translation algorithm assumes that the IPv6 network is somehow well connected i.e. when an IPv6 node wants to communicate with another IPv6 node there is an IPv6 path between them. Various tunneling schemes exist that can provide such a path, but those mechanisms and their use is outside the scope of this document.

The IPv6 protocol [IPv6] has been designed so that the TCP and UDP pseudo-header checksums are not affected by the translations specified in this document, thus the translator does not need to modify normal TCP and UDP headers. The only exceptions are unfragmented IPv4 UDP packets which need to have a UDP checksum computed since a pseudo-header checksum is required for UDP in IPv6. Also, ICMPv6 include a pseudo-header checksum but it is not present in ICMPv4 thus the checksum in ICMP messages need to be modified by the translator. In addition, ICMP error messages contain an IP header as part of the payload thus the translator need to rewrite those parts of the packets to make the receiver be able to understand the included IP header. However, all of the translator’s operations, including path MTU discovery, are stateless in the sense that the translator operates independently on each packet and does not retain any state from one packet to another. This allows redundant translator boxes without any coordination and a given TCP connection can have the two directions of packets go through different translator boxes.

The translating function as specified in this document does not translate any IPv4 options and it does not translate IPv6 routing headers, hop-by-hop extension headers, or destination options headers. It could be possible to define a translation between source routing in IPv4 and IPv6. However such a translation would not be semantically correct due to the slight differences between the IPv4 and IPv6 source routing. Also, the usefulness of source routing when going through a header translator might be limited since all the IPv6-only routers would need to have an IPv4-translated IPv6 address since the IPv4-only node will send a source route option containing only IPv4 addresses.

At first sight it might appear that the IPsec functionality [IPv6-SA, IPv6-ESP, IPv6-AH] can not be carried across the translator. However, since the translator does not modify any headers above the logical IP layer (IP headers, IPv6 fragment headers, and ICMP messages) packets encrypted using ESP in Transport-mode can be carried through the translator. [Note that this assumes that the key management can operate between the IPv6-only node and the IPv4-only node.] The AH computation covers parts of the IPv4 header fields such as IP addresses, and the identification field (fields that are either immutable or predictable by the sender) [IPv6-AUTH]. While the SIIT algorithm is specified so that those IPv4 fields can be predicted by the IPv6 sender it is not possible for the IPv6 receiver to determine the value of the IPv4 Identification field in packets sent by the IPv4 node. Thus as the translation algorithm is specified in this document it is not possible to use end-to-end AH through the translator.

For ESP Tunnel-mode to work through the translator the IPv6 node would have to be able to both parse and generate "inner" IPv4 headers since the inner IP will be encrypted together with the transport protocol.

Thus in practise, only ESP transport mode is relatively easy to make work through a translator.

IPv4 multicast addresses can not be mapped to IPv6 multicast addresses. For instance, ::ffff:224.1.2.3 is an IPv4 mapped IPv6 address with a class D address, however it is not an IPv6 multicast address. While the IP/ICMP header translation aspect of this memo in theory works for multicast packets this address mapping limitation makes it impossible to apply the techniques in this memo for multicast traffic.

1.2. Assumptions

The IPv6 nodes using the translator must have an IPv4-translated IPv6 address while it is communicating with IPv4-only nodes.

The use of the algorithm assumes that there is an IPv4 address pool used to generate IPv4-translated addresses. Routing needs to be able to route any IPv4 packets, whether generated "outside" or "inside" the translator, destined to addresses in this pool towards the translator. This implies that the address pool can not be assigned to subnets but must be separated from the IPv4 subnets used on the "inside" of the translator.

Fragmented IPv4 UDP packets that do not contain a UDP checksum (i.e. the UDP checksum field is zero) are not of significant use over wide-areas in the Internet and will not be translated by the translator. An informal trace [MILLER] in the backbone showed that out of 34,984,468 IP packets there were 769 fragmented UDP packets with a zero checksum. However, all of them were due to malicious or broken behavior; a port scan and first fragments of IP packets that are not a multiple of 8 bytes.

1.3. Impact Outside the Network Layer

The potential existence of stateless IP/ICMP translators is already taken care of from a protocol perspective in [IPv6]. However, an IPv6 node that wants to be able to use translators needs some additional logic in the network layer.

The network layer in an IPv6-only node, when presented by the application with either an IPv4 destination address or an IPv4-mapped IPv6 destination address, is likely to drop the packet and return some error message to the application. In order to take advantage of translators such a node should instead send an IPv6 packet where the destination address is the IPv4-mapped address and the source address is the node’s temporarily assigned IPv4-translated address. If the node does not have a temporarily assigned IPv4-translated address it should acquire one using mechanisms that are not discussed in this document.

Note that the above also applies to a dual IPv4/IPv6 implementation node which is not configured with any IPv4 address.

There are no extra changes needed to applications to operate through a translator beyond what applications already need to do to operate on a dual node. The applications that have been modified to work on a dual node already have the mechanisms to determine whether they are communicating with an IPv4 or an IPv6 peer. Thus if the applications

need to modify their behavior depending on the type of the peer, such as ftp determining whether to fallback to using the PORT/PASV command when EPRT/EPSV fails (as specified in [FTPEXT]), they already need to do that when running on dual nodes and the presense of translators does not add anything. For example, when using the socket API [BSDAPI] the applications know that the peer is IPv6 if they get an AF_INET6 address from the name service and the address is not an IPv4-mapped address (i.e., IN6_IS_ADDR_V4MAPPED returns false). If this is not the case, i.e., the address is AF_INET or an IPv4-mapped IPv6 address, the peer is IPv4.

One way of viewing the translator, which might help clarify why applications do not need to know that a translator is used, is to look at the information that is passed from the transport layer to the network layer. If the transport passes down an IPv4 address (whether or not is in the IPv4-mapped encoding) this means that at some point there will be IPv4 packets generated. In a dual node the generation of the IPv4 packets takes place in the sending node. In an IPv6-only node conceptually the only difference is that the IPv4 packet is generated by the translator - all the information that the transport layer passed to the network layer will be conveyed to the translator in some form. That form just "happens" to be in the form of an IPv6 header.

2. Terminology

This documents uses the terminology defined in [IPv6] and [TRANS-MECH] with these clarifications:

IPv4 capable node:
: A node which has an IPv4 protocol stack. In order for the stack to be usable the node must be assigned one or more IPv4 addresses.

IPv4 enabled node:
: A node which has an IPv4 protocol stack and is assigned one or more IPv4 addresses. Both IPv4-only and IPv6/IPv4 nodes are IPv4 enabled.

IPv6 capable node:
: A node which has an IPv6 protocol stack. In order for the stack to be usable the node must be assigned one or more IPv6 addresses.

IPv6 enabled node:
: A node which has an IPv6 protocol stack and is assigned one or more IPv6 addresses. Both IPv6-only and IPv6/IPv4 nodes are IPv6 enabled.

2.1. Addresses

In addition to the forms of addresses defined in [ADDR-ARCH] this document also introduces the new form of IPv4-translated address. This is needed to avoid using IPv4-compatible addresses outside the intended use of automatic tunneling. Thus the address forms are:

IPv4-mapped:
An address of the form 0::ffff:a.b.c.d which refers to a node that is not IPv6-capable. In addition to its use in the API this protocol uses IPv4-mapped addresses in IPv6 packets to refer to an IPv4 node.

IPv4-compatible:
An address of the form 0::0:a.b.c.d which refers to an IPv6/IPv4 node that supports automatic tunneling. Such addresses are not used in this protocol.

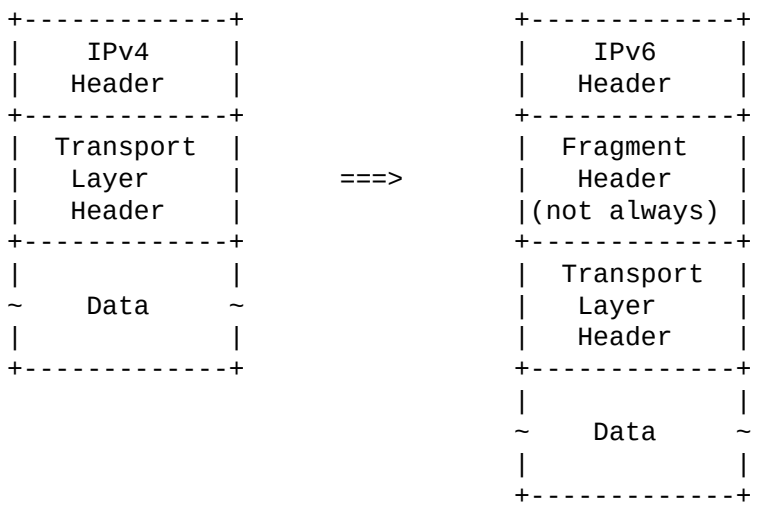
IPv4-translated:
An address of the form 0::ffff:0:a.b.c.d which refers to an IPv6-enabled node. Note that the prefix 0::ffff:0:0:0/96 is chosen to checksum to zero to avoid any changes to the transport protocol's pseudo header checksum.

2.2. Requirements

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [KEYWORDS].

3. Translating from IPv4 to IPv6

When an IPv4-to-IPv6 translator receives an IPv4 datagram addressed to a destination that lies outside of the attached IPv4 island, it translates the IPv4 header of that packet into an IPv6 header. It then forwards the packet based on the IPv6 destination address. The original IPv4 header on the packet is removed and replaced by an IPv6 header. Except for ICMP packets the transport layer header and data portion of the packet are left unchanged.



IPv4-to-IPv6 Translation

One of the differences between IPv4 and IPv6 is that in IPv6 path MTU discovery is mandatory but it is optional in IPv4. This implies that IPv6 routers will never fragment a packet - only the sender can do fragmentation.

When the IPv4 node performs path MTU discovery (by setting the DF bit in the header) the path MTU discovery can operate end-to-end i.e. across the translator. In this case either IPv4 or IPv6 routers might send back ICMP "packet too big" messages to the sender. When these ICMP errors are sent by the IPv6 routers they will pass through a translator which will translate the ICMP error to a form that the IPv4 sender can understand. In this case an IPv6 fragment header is only included if the IPv4 packet is already fragmented.

However, when the IPv4 sender does not perform path MTU discovery the translator has to ensure that the packet does not exceed the path MTU on the IPv6 side. This is done by fragmenting the IPv4 packet so that it fits in 1280 byte IPv6 packet since IPv6 guarantees that 1280 byte packets never need to be fragmented. Also, when the IPv4 sender does not perform path MTU discovery the translator MUST always include an IPv6 fragment header to indicate that the sender allows fragmentation. That is needed should the packet pass through an IPv6-to-IPv4 translator.

The above rules ensure that when packets are fragmented either by the sender or by IPv4 routers that the low-order 16 bits of the fragment identification is carried end-end to ensure that packets are correctly reassembled. In addition, the rules use the presence of an

IPv6 fragment header to indicate that the sender might not be using path MTU discovery i.e. the packet should not have the DF flag set should it later be translated back to IPv4.

Other than the special rules for handling fragments and path MTU discovery the actual translation of the packet header consists of a simple mapping as defined below. Note that ICMP packets require special handling in order to translate the content of ICMP error message and also to add the ICMP pseudo-header checksum.

3.1. Translating IPv4 Headers into IPv6 Headers

If the DF flag is not set and the IPv4 packet will result in an IPv6 packet larger than 1280 bytes the IPv4 packet MUST be fragmented prior to translating it. Since IPv4 packets with DF not set will always result in a fragment header being added to the packet the IPv4 packets must be fragmented so that their length, excluding the IPv4 header, is at most 1232 bytes (1280 minus 40 for the IPv6 header and 8 for the Fragment header). The resulting fragments are then translated independently using the logic described below.

If the DF bit is set and the packet is not a fragment (i.e., the MF flag is not set and the Fragment Offset is zero) then there is no need to add a fragment header to the packet. The IPv6 header fields are set as follows:

Version:
: 6

Traffic Class:
: By default, copied from IP Type Of Service and Precedence field (all 8 bits are copied). According to [DIFFSERV] the semantics of the bits are identical in IPv4 and IPv6. However, in some IPv4 environments these fields might be used with the old semantics of "Type Of Service and Precedence". An implementation of a translator SHOULD provide the ability to ignore the IPv4 "TOS" and always set the IPv6 traffic class to zero.

Flow Label:
: 0 (all zero bits)

Payload Length:
: Total length value from IPv4 header, minus the size of the IPv4 header and IPv4 options, if present.

Next Header:
Protocol field copied from IPv4 header

Hop Limit:
TTL value copied from IPv4 header. Since the translator is a router, as part of forwarding the packet it needs to decrement either the IPv4 TTL (before the translation) or the IPv6 Hop Limit (after the translation). As part of decrementing the TTL or Hop Limit the translator (as any router) needs to check for zero and send the ICMPv4 or ICMPv6 "ttl exceeded" error.

Source Address:
The low-order 32 bits is the IPv4 source address. The high-order 96 bits is the IPv4-mapped prefix (::ffff:0:0/96)

Destination Address:
The low-order 32 bits is the IPv4 destination address. The high-order 96 bits is the IPv4-translated prefix (0::ffff:0:0:0/96)

If IPv4 options are present in the IPv4 packet, they are ignored i.e., there is no attempt to translate them. However, if an unexpired source route option is present then the packet MUST instead be discarded, and an ICMPv4 "destination unreachable/source route failed" (Type 3/Code 5) error message SHOULD be returned to the sender.

If there is need to add a fragment header (the DF bit is not set or the packet is a fragment) the header fields are set as above with the following exceptions:

IPv6 fields:

Payload Length:
Total length value from IPv4 header, plus 8 for the fragment header, minus the size of the IPv4 header and IPv4 options, if present.

Next Header:
Fragment Header (44).

Fragment header fields:

Next Header:
Protocol field copied from IPv4 header.

Fragment Offset:
Fragment Offset copied from the IPv4 header.

M flag:
More Fragments bit copied from the IPv4 header.

Identification:
The low-order 16 bits copied from the Identification field in the IPv4 header. The high-order 16 bits set to zero.

3.2. Translating UDP over IPv4

If a UDP packet has a zero UDP checksum then a valid checksum must be calculated in order to translate the packet. A stateless translator can not do this for fragmented packets but [MILLER] indicates that fragmented UDP packets with a zero checksum appear to only be used for malicious purposes. Thus this is not believed to be a noticeable limitation.

When a translator receives the first fragment of a fragmented UDP IPv4 packet and the checksum field is zero the translator SHOULD drop the packet and generate a system management event specifying at least the IP addresses and port numbers in the packet. When it receives fragments other than the first it SHOULD silently drop the packet, since there is no port information to log.

When a translator receives an unfragmented UDP IPv4 packet and the checksum field is zero the translator MUST compute the missing UDP checksum as part of translating the packet. Also, the translator SHOULD maintain a counter of how many UDP checksums are generated in this manner.

3.3. Translating ICMPv4 Headers into ICMPv6 Headers

All ICMP messages that are to be translated require that the ICMP checksum field be updated as part of the translation since ICMPv6, unlike ICMPv4, has a pseudo-header checksum just like UDP and TCP.

In addition all ICMP packets need to have the Type value translated and for ICMP error messages the included IP header also needs translation.

The actions needed to translate various ICMPv4 messages are:

ICMPv4 query messages:

Echo and Echo Reply (Type 8 and Type 0)
Adjust the type to 128 and 129, respectively, and adjust the ICMP checksum both to take the type change into account and to include the ICMPv6 pseudo-header.

Information Request/Reply (Type 15 and Type 16)
Obsoleted in ICMPv4. Silently drop.

Timestamp and Timestamp Reply (Type 13 and Type 14)
Obsoleted in ICMPv6. Silently drop.

Address Mask Request/Reply (Type 17 and Type 18)
Obsoleted in ICMPv6. Silently drop.

ICMP Router Advertisement (Type 9)
Single hop message. Silently drop.

ICMP Router Solicitation (Type 10)
Single hop message. Silently drop.

Unknown ICMPv4 types
Silently drop.

IGMP messages:

While the MLD messages [MLD] are the logical IPv6 counterparts for the IPv4 IGMP messages all the "normal" IGMP messages are single-hop messages and should be silently dropped by the translator. Other IGMP messages might be used by multicast routing protocols and, since it would be a configuration error to try to have router adjacencies across IPv4/IPv6 translators those packets should also be silently dropped.

ICMPv4 error messages:

Destination Unreachable (Type 3)
For all that are not explicitly listed below set the Type to 1.

Translate the code field as follows:
Code 0, 1 (net, host unreachable):
Set Code to 0 (no route to destination).

Code 2 (protocol unreachable):
Translate to an ICMPv6 Parameter Problem (Type 4, Code 1) and make the Pointer point to the IPv6 Next Header field.

Code 3 (port unreachable):
Set Code to 4 (port unreachable).

Code 4 (fragmentation needed and DF set):
Translate to an ICMPv6 Packet Too Big message (Type 2) with code 0. The MTU field needs to be adjusted for the difference between the IPv4 and IPv6 header sizes. Note that if the IPv4 router did not set the MTU field i.e. the router does not implement [PMTUv4], then the translator must use the plateau values specified in [PMTUv4] to determine a likely path MTU and include that path MTU in the ICMPv6 packet. (Use the greatest plateau value that is less than the returned Total Length field.)

Code 5 (source route failed):
Set Code to 0 (no route to destination). Note that this error is unlikely since source routes are not translated.

Code 6,7:
Set Code to 0 (no route to destination).

Code 8:
Set Code to 0 (no route to destination).

Code 9, 10 (communication with destination host administratively prohibited):
Set Code to 1 (communication with destination administratively prohibited)

Code 11, 12:
Set Code to 0 (no route to destination).

Redirect (Type 5)
Single hop message. Silently drop.

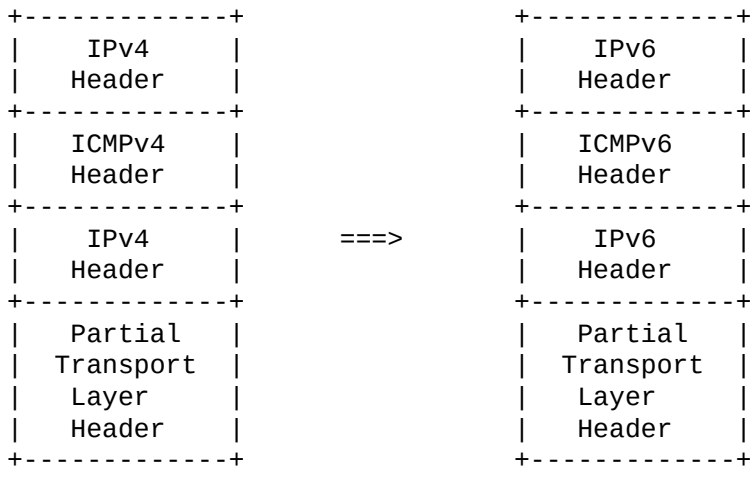
Source Quench (Type 4)
Obsoleted in ICMPv6. Silently drop.

Time Exceeded (Type 11)
Set the Type field to 3. The Code field is unchanged.

Parameter Problem (Type 12)
Set the Type field to 4. The Pointer needs to be updated to point to the corresponding field in the translated include IP header.

3.4. Translating ICMPv4 Error Messages into ICMPv6

There are some differences between the IPv4 and the IPv6 ICMP error message formats as detailed above. In addition, the ICMP error messages contain the IP header for the packet in error which needs to be translated just like a normal IP header. The translation of this "packet in error" is likely to change the length of the datagram thus the Payload Length field in the outer IPv6 header might need to be updated.



IPv4-to-IPv6 ICMP Error Translation

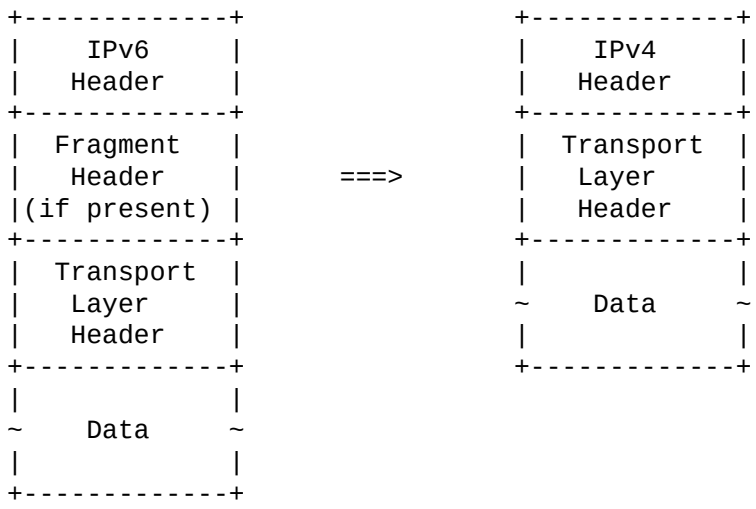
The translation of the inner IP header can be done by recursively invoking the function that translated the outer IP headers.

3.5. Knowing when to Translate

The translator is assumed to know the pool(s) of IPv4 address that are used to represent the internal IPv6-only nodes. Thus if the IPv4 destination field contains an address that falls in these configured sets of prefixes the packet needs to be translated to IPv6.

4. Translating from IPv6 to IPv4

When an IPv6-to-IPv4 translator receives an IPv6 datagram addressed to an IPv4-mapped IPv6 address, it translates the IPv6 header of that packet into an IPv4 header. It then forwards the packet based on the IPv4 destination address. The original IPv6 header on the packet is removed and replaced by an IPv4 header. Except for ICMP packets the transport layer header and data portion of the packet are left unchanged.



IPv6-to-IPv4 Translation

There are some differences between IPv6 and IPv4 in the area of fragmentation and the minimum link MTU that effect the translation. An IPv6 link has to have an MTU of 1280 bytes or greater. The corresponding limit for IPv4 is 68 bytes. Thus, unless there were special measures, it would not be possible to do end-to-end path MTU discovery when the path includes an IPv6-to-IPv4 translator since the IPv6 node might receive ICMP "packet too big" messages originated by an IPv4 router that report an MTU less than 1280. However, [IPv6] requires that IPv6 nodes handle such an ICMP "packet too big" message by reducing the path MTU to 1280 and including an IPv6 fragment header with each packet. This allows end-to-end path MTU discovery across the translator as long as the path MTU is 1280 bytes or greater. When the path MTU drops below the 1280 limit the IPv6 sender will originate 1280 byte packets that will be fragmented by IPv4 routers along the path after being translated to IPv4.

The only drawback with this scheme is that it is not possible to use PMTU to do optimal UDP fragmentation (as opposed to completely avoiding fragmentation) at sender since the presence of an IPv6

Fragment header is interpreted that is it OK to fragment the packet on the IPv4 side. Thus if a UDP application wants to send large packets independent of the PMTU, the sender will only be able to determine the path MTU on the IPv6 side of the translator. If the path MTU on the IPv4 side of the translator is smaller then the IPv6 sender will not receive any ICMP "too big" errors and can not adjust the size fragments it is sending.

Other than the special rules for handling fragments and path MTU discovery the actual translation of the packet header consists of a simple mapping as defined below. Note that ICMP packets require special handling in order to translate the content of ICMP error message and also to add the ICMP pseudo-header checksum.

4.1. Translating IPv6 Headers into IPv4 Headers

If there is no IPv6 Fragment header the IPv4 header fields are set as follows:

Version:
4

Internet Header Length:
5 (no IPv4 options)

Type of Service and Precedence:
By default, copied from the IPv6 Traffic Class (all 8 bits). According to [DIFFSERV] the semantics of the bits are identical in IPv4 and IPv6. However, in some IPv4 environments these bits might be used with the old semantics of "Type Of Service and Precedence". An implementation of a translator SHOULD provide the ability to ignore the IPv6 traffic class and always set the IPv4 "TOS" to zero.

Total Length:
Payload length value from IPv6 header, plus the size of the IPv4 header.

Identification:
All zero.

Flags:
The More Fragments flag is set to zero. The Don't Fragments flag is set to one.

Fragment Offset:
All zero.

Time to Live:
Hop Limit value copied from IPv6 header. Since the translator is a router, as part of forwarding the packet it needs to decrement either the IPv6 Hop Limit (before the translation) or the IPv4 TTL (after the translation). As part of decrementing the TTL or Hop Limit the translator (as any router) needs to check for zero and send the ICMPv4 or ICMPv6 "ttl exceeded" error.

Protocol:
Next Header field copied from IPv6 header.

Header Checksum:
Computed once the IPv4 header has been created.

Source Address:
If the IPv6 source address is an IPv4-translated address then the low-order 32 bits of the IPv6 source address is copied to the IPv4 source address. Otherwise, the source address is set to 0.0.0.0. The use of 0.0.0.0 is to avoid completely dropping e.g. ICMPv6 error messages sent by IPv6-only routers which makes e.g. traceroute present something for the IPv6-only hops.

Destination Address:
IPv6 packets that are translated have an IPv4-mapped destination address. Thus the low-order 32 bits of the IPv6 destination address is copied to the IPv4 destination address.

If any of an IPv6 hop-by-hop options header, destination options header, or routing header with the Segments Left field equal to zero are present in the IPv6 packet, they are ignored i.e., there is no attempt to translate them. However, the Total Length field and the Protocol field would have to be adjusted to "skip" these extension headers.

If a routing header with a non-zero Segments Left field is present then the packet MUST NOT be translated, and an ICMPv6 "parameter problem/ erroneous header field encountered" (Type 4/Code 0) error message, with the Pointer field indicating the first byte of the Segments Left field, SHOULD be returned to the sender.

If the IPv6 packet contains a Fragment header the header fields are set as above with the following exceptions:

Total Length:
Payload length value from IPv6 header, minus 8 for the Fragment header, plus the size of the IPv4 header.

Identification:
Copied from the low-order 16-bits in the Identification field in the Fragment header.

Flags:
The More Fragments flag is copied from the M flag in the Fragment header. The Don't Fragments flag is set to zero allowing this packet to be fragmented by IPv4 routers.

Fragment Offset:
Copied from the Fragment Offset field in the Fragment Header.

Protocol:
Next Header value copied from Fragment header.

4.2. Translating ICMPv6 Headers into ICMPv4 Headers

All ICMP messages that are to be translated require that the ICMP checksum field be updated as part of the translation since ICMPv6, unlike ICMPv4, has a pseudo-header checksum just like UDP and TCP.

In addition all ICMP packets need to have the Type value translated and for ICMP error messages the included IP header also needs translation.

The actions needed to translate various ICMPv6 messages are:

ICMPv6 informational messages:

Echo Request and Echo Reply (Type 128 and 129)
Adjust the type to 0 and 8, respectively, and adjust the ICMP checksum both to take the type change into account and to exclude the ICMPv6 pseudo-header.

MLD Multicast Listener Query/Report/Done (Type 130, 131, 132)
Single hop message. Silently drop.

Neighbor Discover messages (Type 133 through 137)
Single hop message. Silently drop.

Unknown informational messages
Silently drop.

ICMPv6 error messages:

Destination Unreachable (Type 1)
Set the Type field to 3. Translate the code field as follows:
Code 0 (no route to destination):
Set Code to 1 (host unreachable).

Code 1 (communication with destination administratively prohibited):
Set Code to 10 (communication with destination host administratively prohibited).

Code 2 (beyond scope of source address):
Set Code to 1 (host unreachable). Note that this error is very unlikely since the IPv4-translatable source address is considered to have global scope.

Code 3 (address unreachable):
Set Code to 1 (host unreachable).

Code 4 (port unreachable):
Set Code to 3 (port unreachable).

Packet Too Big (Type 2)
Translate to an ICMPv4 Destination Unreachable with code 4. The MTU field needs to be adjusted for the difference between the IPv4 and IPv6 header sizes taking into account whether or not the packet in error includes a Fragment header.

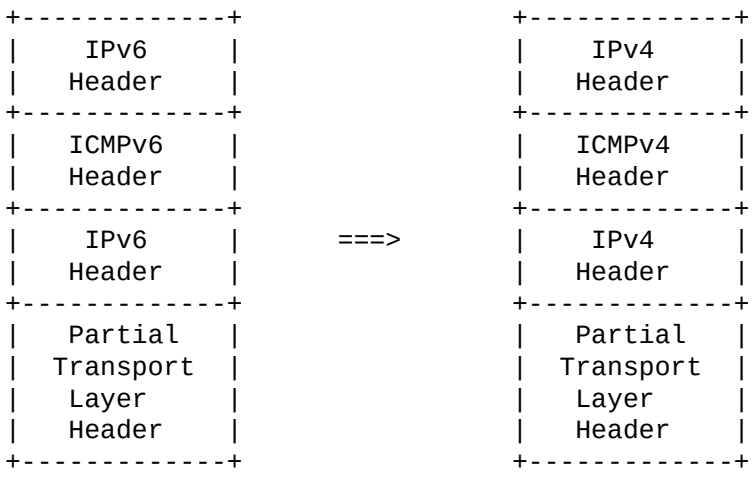
Time Exceeded (Type 3)
Set the Type to 11. The Code field is unchanged.

Parameter Problem (Type 4)
If the Code is 1 translate this to an ICMPv4 protocol unreachable (Type 3, Code 2). Otherwise set the Type to 12 and the Code to zero. The Pointer needs to be updated to point to the corresponding field in the translated include IP header.

Unknown error messages
Silently drop.

4.3. Translating ICMPv6 Error Messages into ICMPv4

There are some differences between the IPv4 and the IPv6 ICMP error message formats as detailed above. In addition, the ICMP error messages contain the IP header for the packet in error which needs to be translated just like a normal IP header. The translation of this "packet in error" is likely to change the length of the datagram thus the Total Length field in the outer IPv4 header might need to be updated.



IPv6-to-IPv4 ICMP Error Translation

The translation of the inner IP header can be done by recursively invoking the function that translated the outer IP headers.

4.4. Knowing when to Translate

When the translator receives an IPv6 packet with an IPv4-mapped destination address the packet will be translated to IPv4.

5. Implications for IPv6-Only Nodes

An IPv6-only node which works through SIIT translators need some modifications beyond a normal IPv6-only node.

As specified in Section 1.3 the application protocols need to handle operation on a dual stack node. In addition the protocol stack needs to be able to:

- Determine when an IPv4-translatable address needs to be allocated and the allocation needs to be refreshed/renewed. This can presumably be done without involving the applications by e.g. handling this under the socket API. For instance, when the connect or sendto socket calls are invoked they could check if the destination is an IPv4-mapped address and in that case allocate/refresh the IPv4-translatable address.

- Ensure, as part of the source address selection mechanism, that when the destination address is an IPv4-mapped address the source address MUST be an IPv4-translatable address. And an IPv4-translatable address MUST NOT be used with other forms of IPv6 destination addresses.

- Should the peer have AAAA/A6 address records the application (or resolver) SHOULD never fall back to looking for A address records even if communication fails using the available AAAA/A6 records. The reason for this restriction is to prevent traffic between two IPv6 nodes (which AAAA/A6 records in the DNS) from accidentally going through SIIT translators twice; from IPv6 to IPv4 and to IPv6 again. It is considered preferable to instead signal a failure to communicate to the application.

6. Security Considerations

The use of stateless IP/ICMP translators does not introduce any new security issues beyond the security issues that are already present in the IPv4 and IPv6 protocols and in the routing protocols which are used to make the packets reach the translator.

As the Authentication Header [IPv6-AUTH] is specified to include the IPv4 Identification field and the translating function not being able to always preserve the Identification field, it is not possible for an IPv6 endpoint to compute AH on received packets that have been translated from IPv4 packets. Thus AH does not work through a translator.

Packets with ESP can be translated since ESP does not depend on header fields prior to the ESP header. Note that ESP transport mode is easier to handle than ESP tunnel mode; in order to use ESP tunnel mode the IPv6 node needs to be able to generate an inner IPv4 header when transmitting packets and remove such an IPv4 header when receiving packets.

References

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

[IPv6] Deering, S. and R. Hinden, Editors, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[IPv4] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[ADDR-ARCH] Deering, S. and R. Hinden, Editors, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[TRANS-MECH] Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 1933, April 1996.

[DISCOVERY] Narten, T., Nordmark, E. and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[IPv6-SA] Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[IPv6-AUTH] Atkinson, R., "IP Authentication Header", RFC 2402, November 1998.

[IPv6-ESP] Atkinson, R., "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[ICMPv4] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[ICMPv6] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)", RFC 2463, December 1998.

[IGMP] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, August 1989.

[PMTUv4] Mogul, J. and S. Deering, "Path MTU Discovery", RFC 1191, November 1990.

[PMTUv6] McCann, J., Deering, S. and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[DIFFSERV] Nichols, K., Blake, S., Baker, F. and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[MLD] Deering, S., Fenner, W. and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, October 1999.

[FTPEXT] Allman, M., Ostermann, S. and C. Metz, "FTP Extensions for IPv6 and NATs.", RFC 2428, September 1998.

[MILLER] G. Miller, Email to the ngtrans mailing list on 26 March 1999.

[BSDAPI] Gilligan, R., Thomson, S., Bound, J. and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 2553, March 1999.

Author's Address

Erik Nordmark
Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
USA

Phone: +1 650 786 5166
Fax: +1 650 786 5896
EMail: nordmark@sun.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.