INTERNET-DRAFT
April 14, 2000
Obsoletes: 1933

R. E. Gilligan
FreeGate Corp.
E.Nordmark
Sun Microsystems, Inc.

Transition Mechanisms for IPv6 Hosts and Routers
<draft-ietf-ngtrans-mech-06.txt>

Status of this Memo

This draft expires on October 14, 2000

Abstract

This document specifies IPv4 compatibility mechanisms that can be implemented by IPv6 hosts and routers. These mechanisms include providing complete implementations of both versions of the Internet Protocol (IPv4 and IPv6), and tunneling IPv6 packets over IPv4 routing infrastructures. They are designed to allow IPv6 nodes to maintain complete compatibility with IPv4, which should greatly simplify the deployment of IPv6 in the Internet, and facilitate the eventual transition of the entire Internet to IPv6.

This document obsoletes RFC 1933.

Contents

1. Introduction

The key to a successful IPv6 transition is compatibility with the large installed base of IPv4 hosts and routers. Maintaining compatibility with IPv4 while deploying IPv6 will streamline the task of transitioning the Internet to IPv6. This specification defines a set of mechanisms that IPv6 hosts and routers may implement in order to be compatible with IPv4 hosts and routers.

The mechanisms in this document are designed to be employed by IPv6 hosts and routers that need to interoperate with IPv4 hosts and utilize IPv4 routing infrastructures. We expect that most nodes in the Internet will need such compatibility for a long time to come, and perhaps even indefinitely.

However, IPv6 may be used in some environments where interoperability with IPv4 is not required. IPv6 nodes that are designed to be used in such environments need not use or even implement these mechanisms.

The mechanisms specified here include:

- Dual IP layer (also known as Dual Stack): A technique for providing complete support for both Internet protocols -- IPv4 and IPv6 -- in hosts and routers.

- Configured tunneling of IPv6 over IPv4: Point-to-point tunnels made by encapsulating IPv6 packets within IPv4 headers to carry them over IPv4 routing infrastructures.

- IPv4-compatible IPv6 addresses: An IPv6 address format that employs embedded IPv4 addresses.

- Automatic tunneling of IPv6 over IPv4: A mechanism for using IPv4-compatible addresses to automatically tunnel IPv6 packets over IPv4 networks.

The mechanisms defined here are intended to be part of a "transition toolbox" -- a growing collection of techniques which implementations and users may employ to ease the transition. The tools may be used as needed. Implementations and sites decide which techniques are appropriate to their specific needs. This document defines the initial core set of transition mechanisms, but these are not expected to be the only tools available. Additional transition and compatibility mechanisms are expected to be developed in the future, with new documents being written to specify them.

1.1. Terminology

The following terms are used in this document:

Types of Nodes

IPv4-only node:

A host or router that implements only IPv4. An IPv4-only node does not understand IPv6. The installed base of IPv4 hosts and routers existing before the transition begins are IPv4-only nodes.

IPv6/IPv4 node:

A host or router that implements both IPv4 and IPv6.

IPv6-only node:

A host or router that implements IPv6, and does not implement IPv4. The operation of IPv6-only nodes is not addressed here.

IPv6 node:

Any host or router that implements IPv6. IPv6/IPv4 and IPv6-only nodes are both IPv6 nodes.

IPv4 node:

Any host or router that implements IPv4. IPv6/IPv4 and IPv4-only nodes are both IPv4 nodes.

Types of IPv6 Addresses

IPv4-compatible IPv6 address:

An IPv6 address bearing the high-order 96-bit prefix 0:0:0:0:0:0, and an IPv4 address in the low-order 32-bits. IPv4-compatible addresses are used by IPv6/IPv4 nodes which perform automatic tunneling,

IPv6-native address:

The remainder of the IPv6 address space. An IPv6 address that bears a prefix other than 0:0:0:0:0:0.

Techniques Used in the Transition

IPv6-over-IPv4 tunneling:

The technique of encapsulating IPv6 packets within IPv4 so that they can be carried across IPv4 routing infrastructures.

Configured tunneling:

IPv6-over-IPv4 tunneling where the IPv4 tunnel endpoint address is determined by configuration information on the encapsulating node. The tunnels can be either unidirectional or bidirectional. Bidirectional configured tunnels behave as virtual point-to-point links.

Automatic tunneling:

IPv6-over-IPv4 tunneling where the IPv4 tunnel endpoint address is determined from the IPv4 address embedded in the IPv4-compatible destination address of the IPv6 packet being tunneled.

IPv4 multicast tunneling:

IPv6-over-IPv4 tunneling where the IPv4 tunnel endpoint address is determined using Neighbor Discovery [7]. Unlike configured tunneling this does not require any address configuration and unlike automatic tunneling it does not require the use of IPv4-compatible addresses. However, the mechanism assumes that the IPv4 infrastructure supports IPv4 multicast. Specified in [3] and not further discussed in this document.

Other transition mechanisms, including other tunneling mechanisms, are outside the scope of this document.

Modes of operation of IPv6/IPv4 nodes

IPv6-only operation:

An IPv6/IPv4 node with its IPv6 stack enabled and its IPv4 stack disabled.

IPv4-only operation:

An IPv6/IPv4 node with its IPv4 stack enabled and its IPv6 stack disabled.

IPv6/IPv4 operation:

An IPv6/IPv4 node with both stacks enabled.

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 [16].

1.2. Structure of this Document

The remainder of this document is organized as follows:

- Section 2 discusses the operation of nodes with a dual IP layer, IPv6/IPv4 nodes.
- Section 3 discusses the common mechanisms used in both of the IPv6-over-IPv4 tunneling techniques.
- Section 4 discusses configured tunneling.
- Section 5 discusses automatic tunneling and the IPv4-compatible IPv6 address format.

2. Dual IP Layer Operation

The most straightforward way for IPv6 nodes to remain compatible with IPv4-only nodes is by providing a complete IPv4 implementation. IPv6 nodes that provide a complete IPv4 and IPv6 implementations are called "IPv6/IPv4 nodes." IPv6/IPv4 nodes have the ability to send and receive both IPv4 and IPv6 packets. They can directly interoperate with IPv4 nodes using IPv4 packets, and also directly interoperate with IPv6 nodes using IPv6 packets.

Even though a node may be equipped to support both protocols, one or the other stack may be disabled for operational reasons. Thus IPv6/IPv4 nodes may be operated in one of three modes:

- With their IPv4 stack enabled and their IPv6 stack disabled.
- With their IPv6 stack enabled and their IPv4 stack disabled.
- With both stacks enabled.

IPv6/IPv4 nodes with their IPv6 stack disabled will operate like IPv4-only nodes. Similarly, IPv6/IPv4 nodes with their IPv4 stacks disabled will operate like IPv6-only nodes. IPv6/IPv4 nodes MAY provide a configuration switch to disable either their IPv4 or IPv6 stack.

The dual IP layer technique may or may not be used in conjunction with the IPv6-over-IPv4 tunneling techniques, which are described in sections 3, 4 and 5. An IPv6/IPv4 node that supports tunneling MAY support only configured tunneling, or both configured and automatic tunneling. Thus three modes of tunneling support are possible:

- IPv6/IPv4 node that does not perform tunneling.

- IPv6/IPv4 node that performs configured tunneling only.

- IPv6/IPv4 node that performs configured tunneling and automatic tunneling.

2.1. Address Configuration

Because they support both protocols, IPv6/IPv4 nodes may be configured with both IPv4 and IPv6 addresses. IPv6/IPv4 nodes use IPv4 mechanisms (e.g. DHCP) to acquire their IPv4 addresses, and IPv6 protocol mechanisms (e.g. stateless address autoconfiguration) to acquire their IPv6-native addresses. Section 5.2 describes a mechanism by which IPv6/IPv4 nodes that support automatic tunneling MAY use IPv4 protocol mechanisms to acquire their IPv4-compatible IPv6 address.

2.2. DNS

The Domain Naming System (DNS) is used in both IPv4 and IPv6 to map between hostnames and IP addresses. A new resource record type named "A6" has been defined for IPv6 addresses [6] with support for an earlier record named "AAAA". Since IPv6/IPv4 nodes must be able to interoperate directly with both IPv4 and IPv6 nodes, they must provide resolver libraries capable of dealing with IPv4 "A" records as well as IPv6 "A6" and "AAAA" records.

DNS resolver libraries on IPv6/IPv4 nodes MUST be capable of handling both A6/AAAA and A records. However, when a query locates an A6/AAAA record holding an IPv6 address, and an A record holding an IPv4 address, the resolver library MAY filter or order the results

returned to the application in order to influence the version of IP packets used to communicate with that node. In terms of filtering, the resolver library has three alternatives:

- Return only the IPv6 address to the application.

- Return only the IPv4 address to the application.

- Return both addresses to the application.

If it returns only the IPv6 address, the application will communicate with the node using IPv6. If it returns only the IPv4 address, the application will communicate with the node using IPv4. If it returns both addresses, the application will have the choice which address to use, and thus which IP protocol to employ.

If it returns both, the resolver MAY elect to order the addresses -- IPv6 first, or IPv4 first. Since most applications try the addresses in the order they are returned by the resolver, this can affect the IP version "preference" of applications.

The decision to filter or order DNS results is implementation specific. IPv6/IPv4 nodes MAY provide policy configuration to control filtering or ordering of addresses returned by the resolver, or leave the decision entirely up to the application.

An implementation MUST allow the application to control whether or not such filtering takes place.

2.3. Advertising Addresses in the DNS

There are some constraint placed on the use of the DNS during transition. Most of these are obvious but are stated here for completeness.

The recommendation is that A6/AAAA records for a node should not be added to the DNS until all of these are true:

1) The address is assigned to the interface on the node.

2) The address is configured on the interface.

3) The interface is on a link which is connected to the IPv6 infrastructure.

If an IPv6 node is isolated from an IPv6 perspective (e.g. it is not

connected to the 6bone to take a concrete example) constraint #3 would mean that it should not have an address in the DNS.

This works great when other dual stack nodes tries to contact the isolated dual stack node. There is no IPv6 address in the DNS thus the peer doesn't even try communicating using IPv6 but goes directly to IPv4 (we are assuming both nodes have A records in the DNS.)

However, this does not work well when the isolated node is trying to establish communication. Even though it does not have an IPv6 address in the DNS it will find A6/AAAA records in the DNS for the peer. Since the isolated node has IPv6 addresses assigned to at least one interface it will try to communicate using IPv6. If it has no IPv6 route to the 6bone (e.g. because the local router was upgraded to advertise IPv6 addresses using Neighbor Discovery but that router doesn't have any IPv6 routes) this communication will fail. Typically this means a few minutes of delay as TCP times out. The TCP specification says that ICMP unreachable messages could be due to routing transients thus they should not immediately terminate the TCP connection. This means that the normal TCP timeout of a few minutes apply. Once TCP times out the application will hopefully try the IPv4 addresses based on the A records in the DNS, but this will be painfully slow.

A possible implication of the recommendations above is that, if one enables IPv6 on a node on a link without IPv6 infrastructure, and choose to add A6/AAAA records to the DNS for that node, then external IPv6 nodes that might see these A6/AAAA records will possibly try to reach that node using IPv6 and suffer delays or communication failure due to unreachability. (A delay is incurred if the application correctly falls back to using IPv4 if it can not establish communication using IPv6 addresses. If this fallback is not done the application would fail to communicate in this case.) Thus it is suggested that either the recommendations be followed, or care be taken to only do so with nodes that will not be impacted by external accessing delays and/or communication failure.

In the future when a site or node removes the support for IPv4 the above recommendations apply to when the A records for the node(s) should be removed from the DNS.

3. Common Tunneling Mechanisms

In most deployment scenarios, the IPv6 routing infrastructure will be built up over time. While the IPv6 infrastructure is being deployed, the existing IPv4 routing infrastructure can remain functional, and can be used to carry IPv6 traffic. Tunneling provides a way to utilize an existing IPv4 routing infrastructure to carry IPv6 traffic.

IPv6/IPv4 hosts and routers can tunnel IPv6 datagrams over regions of IPv4 routing topology by encapsulating them within IPv4 packets. Tunneling can be used in a variety of ways:

- Router-to-Router. IPv6/IPv4 routers interconnected by an IPv4 infrastructure can tunnel IPv6 packets between themselves. In this case, the tunnel spans one segment of the end-to-end path that the IPv6 packet takes.

- Host-to-Router. IPv6/IPv4 hosts can tunnel IPv6 packets to an intermediary IPv6/IPv4 router that is reachable via an IPv4 infrastructure. This type of tunnel spans the first segment of the packet's end-to-end path.

- Host-to-Host. IPv6/IPv4 hosts that are interconnected by an IPv4 infrastructure can tunnel IPv6 packets between themselves. In this case, the tunnel spans the entire end-to-end path that the packet takes.

- Router-to-Host. IPv6/IPv4 routers can tunnel IPv6 packets to their final destination IPv6/IPv4 host. This tunnel spans only the last segment of the end-to-end path.

Tunneling techniques are usually classified according to the mechanism by which the encapsulating node determines the address of the node at the end of the tunnel. In the first two tunneling methods listed above -- router-to-router and host-to-router -- the IPv6 packet is being tunneled to a router. The endpoint of this type of tunnel is an intermediary router which must decapsulate the IPv6 packet and forward it on to its final destination. When tunneling to a router, the endpoint of the tunnel is different from the destination of the packet being tunneled. So the addresses in the IPv6 packet being tunneled can not provide the IPv4 address of the tunnel endpoint. Instead, the tunnel endpoint address must be determined from configuration information on the node performing the tunneling. We use the term "configured tunneling" to describe the type of tunneling where the endpoint is explicitly configured.

In the last two tunneling methods -- host-to-host and router-to-host

-- the IPv6 packet is tunneled all the way to its final destination. In this case, the destination address of both the IPv6 packet and the encapsulating IPv4 header identify the same node! This fact can be exploited by encoding information in the IPv6 destination address that will allow the encapsulating node to determine tunnel endpoint IPv4 address automatically. Automatic tunneling employs this technique, using an special IPv6 address format with an embedded IPv4 address to allow tunneling nodes to automatically derive the tunnel endpoint IPv4 address. This eliminates the need to explicitly configure the tunnel endpoint address, greatly simplifying configuration.

The two tunneling techniques -- automatic and configured -- differ primarily in how they determine the tunnel endpoint address. Most of the underlying mechanisms are the same:

- The entry node of the tunnel (the encapsulating node) creates an encapsulating IPv4 header and transmits the encapsulated packet.

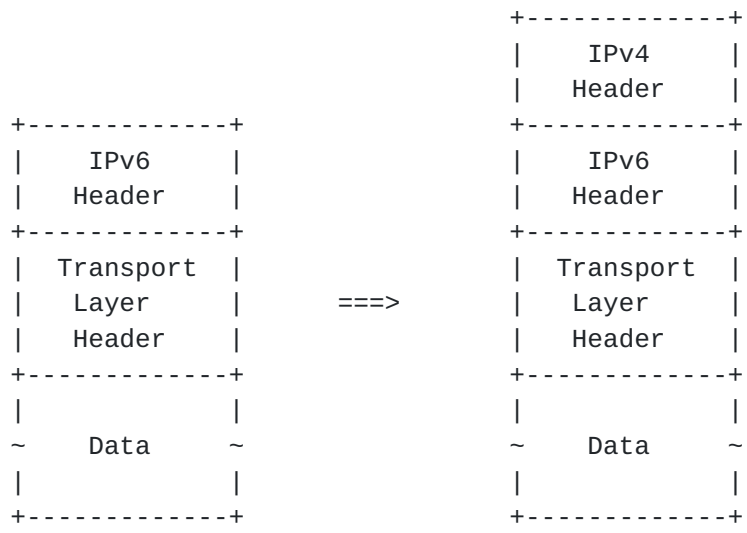
- The exit node of the tunnel (the decapsulating node) receives the encapsulated packet, reassembles the packet if needed, removes the IPv4 header, updates the IPv6 header, and processes the received IPv6 packet.

- The encapsulating node MAY need to maintain soft state information for each tunnel recording such parameters as the MTU of the tunnel in order to process IPv6 packets forwarded into the tunnel. Since the number of tunnels that any one host or router may be using may grow to be quite large, this state information can be cached and discarded when not in use.

The remainder of this section discusses the common mechanisms that apply to both types of tunneling. Subsequent sections discuss how the tunnel endpoint address is determined for automatic and configured tunneling.

3.1. Encapsulation

The encapsulation of an IPv6 datagram in IPv4 is shown below:



Encapsulating IPv6 in IPv4

In addition to adding an IPv4 header, the encapsulating node also has to handle some more complex issues:

- Determine when to fragment and when to report an ICMP "packet too big" error back to the source.

- How to reflect IPv4 ICMP errors from routers along the tunnel path back to the source as IPv6 ICMP errors.

Those issues are discussed in the following sections.

3.2. Tunnel MTU and Fragmentation

The encapsulating node could view encapsulation as IPv6 using IPv4 as a link layer with a very large MTU (65535-20 bytes to be exact; 20 bytes "extra" are needed for the encapsulating IPv4 header). The encapsulating node would need only to report IPv6 ICMP "packet too big" errors back to the source for packets that exceed this MTU. However, such a scheme would be inefficient for two reasons:

1) It would result in more fragmentation than needed. IPv4 layer fragmentation SHOULD be avoided due to the performance problems caused by the loss unit being smaller than the retransmission unit [11].

2) Any IPv4 fragmentation occurring inside the tunnel would have to be reassembled at the tunnel endpoint. For tunnels that terminate at a router, this would require additional memory to reassemble the IPv4 fragments into a complete IPv6 packet before that packet could be forwarded onward.

The fragmentation inside the tunnel can be reduced to a minimum by having the encapsulating node track the IPv4 Path MTU across the tunnel, using the IPv4 Path MTU Discovery Protocol [8] and recording the resulting path MTU. The IPv6 layer in the encapsulating node can then view a tunnel as a link layer with an MTU equal to the IPv4 path MTU, minus the size of the encapsulating IPv4 header.

Note that this does not completely eliminate IPv4 fragmentation in the case when the IPv4 path MTU would result in an IPv6 MTU less than 1280 bytes. (Any link layer used by IPv6 has to have an MTU of at least 1280 bytes [4].) In this case the IPv6 layer has to "see" a link layer with an MTU of 1280 bytes and the encapsulating node has to use IPv4 fragmentation in order to forward the 1280 byte IPv6 packets.

The encapsulating node can employ the following algorithm to determine when to forward an IPv6 packet that is larger than the tunnel's path MTU using IPv4 fragmentation, and when to return an IPv6 ICMP "packet too big" message:

```
        if (IPv4 path MTU - 20) is less than or equal to 1280
                if packet is larger than 1280 bytes
                        Send IPv6 ICMP "packet too big" with MTU = 1280.
                        Drop packet.
                else
                        Encapsulate but do not set the Don't Fragment
                        flag in the IPv4 header.  The resulting IPv4
                        packet might be fragmented by the IPv4 layer on
                        the encapsulating node or by some router along
                        the IPv4 path.
                endif
        else
                if packet is larger than (IPv4 path MTU - 20)
                        Send IPv6 ICMP "packet too big" with
                        MTU = (IPv4 path MTU - 20).
                        Drop packet.
                else
                        Encapsulate and set the Don't Fragment flag
                        in the IPv4 header.
                endif
        endif
```

Encapsulating nodes that have a large number of tunnels might not be able to store the IPv4 Path MTU for all tunnels. Such nodes can, at the expense of additional fragmentation in the network, avoid using the IPv4 Path MTU algorithm across the tunnel and instead use the MTU of the link layer (under IPv4) in the above algorithm instead of the IPv4 path MTU.

In this case the Don't Fragment bit MUST NOT be set in the encapsulating IPv4 header.

3.3. Hop Limit

IPv6-over-IPv4 tunnels are modeled as "single-hop". That is, the IPv6 hop limit is decremented by 1 when an IPv6 packet traverses the tunnel. The single-hop model serves to hide the existence of a tunnel. The tunnel is opaque to users of the network, and is not detectable by network diagnostic tools such as traceroute.

The single-hop model is implemented by having the encapsulating and decapsulating nodes process the IPv6 hop limit field as they would if they were forwarding a packet on to any other datalink. That is, they decrement the hop limit by 1 when forwarding an IPv6 packet. (The originating node and final destination do not decrement the hop limit.)

The TTL of the encapsulating IPv4 header is selected in an implementation dependent manner. The current suggested value is published in the "Assigned Numbers RFC. Implementations MAY provide a mechanism to allow the administrator to configure the IPv4 TTL such as the one specified in the IP Tunnel MIB [17].

3.4. Handling IPv4 ICMP errors

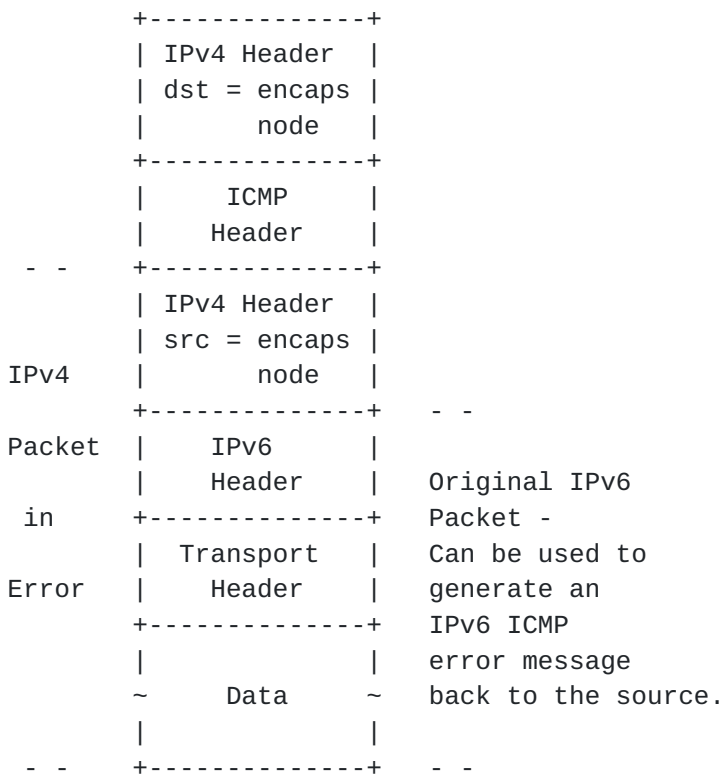
In response to encapsulated packets it has sent into the tunnel, the encapsulating node might receive IPv4 ICMP error messages from IPv4 routers inside the tunnel. These packets are addressed to the encapsulating node because it is the IPv4 source of the encapsulated packet.

The ICMP "packet too big" error messages are handled according to IPv4 Path MTU Discovery [8] and the resulting path MTU is recorded in the IPv4 layer. The recorded path MTU is used by IPv6 to determine if an IPv6 ICMP "packet too big" error has to be generated as described in section 3.2.

The handling of other types of ICMP error messages depends on how much information is included in the "packet in error" field, which holds the encapsulated packet that caused the error.

Many older IPv4 routers return only 8 bytes of data beyond the IPv4 header of the packet in error, which is not enough to include the address fields of the IPv6 header. More modern IPv4 routers are likely to return enough data beyond the IPv4 header to include the entire IPv6 header and possibly even the data beyond that.

If the offending packet includes enough data, the encapsulating node MAY extract the encapsulated IPv6 packet and use it to generate an IPv6 ICMP message directed back to the originating IPv6 node, as shown below:



IPv4 ICMP Error Message Returned to Encapsulating Node

3.5. IPv4 Header Construction

When encapsulating an IPv6 packet in an IPv4 datagram, the IPv4 header fields are set as follows:

Version:

4

IP Header Length in 32-bit words:

5 (There are no IPv4 options in the encapsulating header.)

Type of Service:

0. [Note that work underway in the IETF is redefining the Type of Service byte and as a result future RFCs might define a different behavior for the ToS byte when tunneling.]

Total Length:

Payload length from IPv6 header plus length of IPv6 and IPv4 headers (i.e. a constant 60 bytes).

Identification:

Generated uniquely as for any IPv4 packet transmitted by the system.

Flags:

Set the Don't Fragment (DF) flag as specified in section 3.2. Set the More Fragments (MF) bit as necessary if fragmenting.

Fragment offset:

Set as necessary if fragmenting.

Time to Live:

Set in implementation-specific manner.

Protocol:

41 (Assigned payload type number for IPv6)

Header Checksum:

Calculate the checksum of the IPv4 header.

Source Address:

IPv4 address of outgoing interface of the encapsulating node.

Destination Address:

IPv4 address of tunnel endpoint.

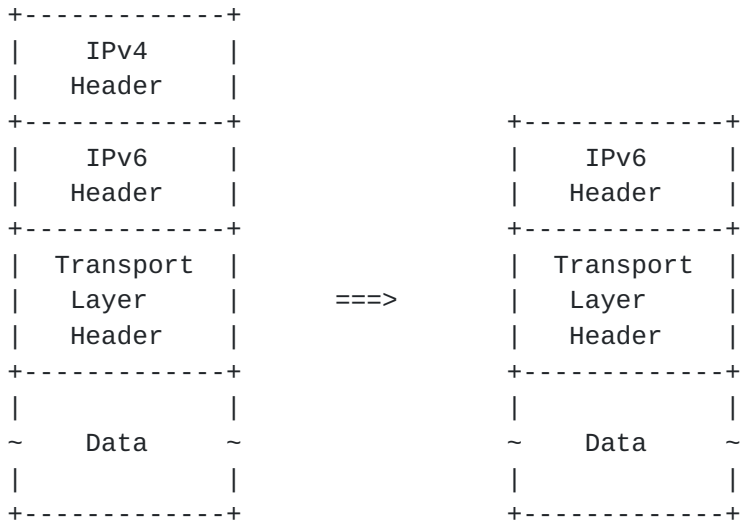
Any IPv6 options are preserved in the packet (after the IPv6 header).

3.6. Decapsulation

When an IPv6/IPv4 host or a router receives an IPv4 datagram that is addressed to one of its own IPv4 address, and the value of the protocol field is 41, it reassembles if the packet if it is fragmented at the IPv4 level, then it removes the IPv4 header and submits the IPv6 datagram to its IPv6 layer code.

The decapsulating node MUST be capable of reassembling an IPv4 packet that is 1300 bytes (1280 bytes plus IPv4 header).

The decapsulation is shown below:



Decapsulating IPv6 from IPv4

When decapsulating the packet, the IPv6 header is not modified. [Note that work underway in the IETF is redefining the Type of Service byte and as a result future RFCs might define a different

behavior for the ToS byte when decapsulating a tunneled packet.] If the packet is subsequently forwarded, its hop limit is decremented by one.

As part of the decapsulation the node SHOULD silently discard a packet with an invalid IPv4 source address such as a multicast address, a broadcast address, 0.0.0.0, and 127.0.0.1. In general it SHOULD apply the rules for martian filtering in [18] and ingress filtering [13] on the IPv4 source address.

The encapsulating IPv4 header is discarded.

After the decapsulation the node SHOULD silently discard a packet with an invalid IPv6 source address. This includes IPv6 multicast addresses, the unspecified address, and the loopback address but also IPv4-compatible IPv6 source addresses where the IPv4 part of the address is an (IPv4) multicast address, broadcast address, 0.0.0.0, or 127.0.0.1. In general it SHOULD apply the rules for martian filtering in [18] and ingress filtering [13] on the IPv4-compatible source address.

The decapsulating node performs IPv4 reassembly before decapsulating the IPv6 packet. All IPv6 options are preserved even if the encapsulating IPv4 packet is fragmented.

After the IPv6 packet is decapsulated, it is processed almost the same as any received IPv6 packet. The only difference being that a decapsulated packet MUST NOT be forwarded unless the node has been explicitly configured to forward such packets for the given IPv4 source address. This configuration can be implicit in e.g., having a configured tunnel which matches the IPv4 source address. This restriction is needed to prevent tunneling to be used as a tool to circumvent ingress filtering [13].

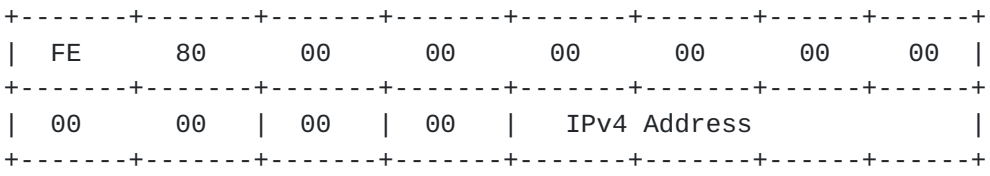
3.7. Link-Local Addresses

Both the configured and automatic tunnels are IPv6 interfaces (over the IPv4 "link layer") thus MUST have link-local addresses. The link-local addresses are used by routing protocols operating over the tunnels.

The Interface Identifier [14] for such an Interface SHOULD be the 32-bit IPv4 address of that interface, with the bytes in the same order in which they would appear in the header of an IPv4 packet, padded at the left with zeros to a total of 64 bits. Note that the "Universal/Local" bit is zero, indicating that the Interface

Identifier is not globally unique. When the host has more than one IPv4 address in use on the physical interface concerned, an administrative choice of one of these IPv4 addresses is made.

The IPv6 Link-local address [14] for an IPv4 virtual interface is formed by appending the Interface Identifier, as defined above, to the prefix FE80::/64.



3.8. Neighbor Discovery over Tunnels

Automatic tunnels and unidirectional configured tunnels are considered to be unidirectional. Thus the only aspects of Neighbor Discovery [7] and Stateless Address Autoconfiguration [5] that apply to these tunnels is the formation of the link-local address.

If an implementation provides bidirectional configured tunnels it MUST at least accept and respond to the probe packets used by Neighbor Unreachability Detection [7]. Such implementations SHOULD also send NUD probe packets to detect when the configured tunnel fails at which point the implementation can use an alternate path to reach the destination. Note that Neighbor Discovery allows that the sending of NUD probes be omitted for router to router links if the routing protocol tracks bidirectional reachability.

For the purposes of Neighbor Discovery the automatic and configured tunnels specified in this document as assumed to NOT have a link-layer address, even though the link-layer (IPv4) does have address. This means that a sender of Neighbor Discovery packets

- SHOULD NOT include Source Link Layer Address options or Target Link Layer Address options on the tunnel link.

- MUST silently ignore any received SLLA or TLLA options on the tunnel link.

4. Configured Tunneling

In configured tunneling, the tunnel endpoint address is determined from configuration information in the encapsulating node. For each tunnel, the encapsulating node must store the tunnel endpoint address. When an IPv6 packet is transmitted over a tunnel, the tunnel endpoint address configured for that tunnel is used as the destination address for the encapsulating IPv4 header.

The determination of which packets to tunnel is usually made by routing information on the encapsulating node. This is usually done via a routing table, which directs packets based on their destination address using the prefix mask and match technique.

4.1. Default Configured Tunnel

IPv6/IPv4 hosts that are connected to datalinks with no IPv6 routers MAY use a configured tunnel to reach an IPv6 router. This tunnel allows the host to communicate with the rest of the IPv6 Internet (i.e. nodes with IPv6-native addresses). If the IPv4 address of an IPv6/IPv4 router bordering the IPv6 backbone is known, this can be used as the tunnel endpoint address. This tunnel can be configured into the routing table as an IPv6 "default route". That is, all IPv6 destination addresses will match the route and could potentially traverse the tunnel. Since the "mask length" of such a default route is zero, it will be used only if there are no other routes with a longer mask that match the destination. The default configured tunnel can be used in conjunction with automatic tunneling, as described in section 5.4.

4.2. Default Configured Tunnel using IPv4 "Anycast Address"

The tunnel endpoint address of such a default tunnel could be the IPv4 address of one IPv6/IPv4 router at the border of the IPv6 backbone. Alternatively, the tunnel endpoint could be an IPv4 "anycast address". With this approach, multiple IPv6/IPv4 routers at the border advertise IPv4 reachability to the same IPv4 address. All of these routers accept packets to this address as their own, and will decapsulate IPv6 packets tunneled to this address. When an IPv6/IPv4 node sends an encapsulated packet to this address, it will be delivered to only one of the border routers, but the sending node will not know which one. The IPv4 routing system will generally carry the traffic to the closest router.

Using a default tunnel to an IPv4 "anycast address" provides a high degree of robustness since multiple border router can be provided, and, using the normal fallback mechanisms of IPv4 routing, traffic will automatically switch to another router when one goes down. However, care must be taking when using such a default tunnel to prevent different IPv4 fragments from arriving at different routers for reassembly. This can be prevented by either avoiding fragmentation of the encapsulated packets (by ensuring an IPv4 MTU of at least 1300 bytes) or by preventing frequent changes to IPv4 routing.

4.3. Ingress Filtering

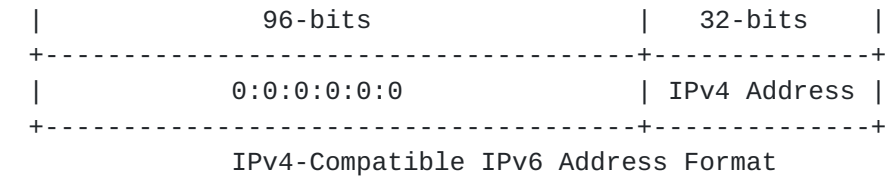
The decapsulating node MUST verify that the tunnel source address is acceptable before forwarding decapsulated packets to avoid circumventing ingress filtering [13]. Note that packets which are delivered to transport protocols on the decapsulating node SHOULD NOT be subject to these checks. For bidirectional configured tunnels this is done by verifying that the source address is the IPv4 address of the other end of the tunnel. For unidirectional configured tunnels the decapsulating node MUST be configured with a list of source IPv4 address prefixes that are acceptable. Such a list MUST default to not having any entries i.e. the node has to be explicitly configured to forward decapsulated packets received over unidirectional configured tunnels.

5. Automatic Tunneling

In automatic tunneling, the tunnel endpoint address is determined by the IPv4-compatible destination address of the IPv6 packet being tunneled. Automatic tunneling allows IPv6/IPv4 nodes to communicate over IPv4 routing infrastructures without pre-configuring tunnels.

5.1. IPv4-Compatible Address Format

IPv6/IPv4 nodes that perform automatic tunneling are assigned IPv4-compatible address. An IPv4-compatible address is identified by an all-zeros 96-bit prefix, and holds an IPv4 address in the low-order 32-bits. IPv4-compatible addresses are structured as follows:



IPv4-Compatible IPv6 Address Format

IPv4-compatible addresses are assigned exclusively to nodes that support automatic tunneling. A node SHOULD be configured with an IPv4-compatible address only if it is prepared to accept IPv6 packets destined to that address encapsulated in IPv4 packets destined to the embedded IPv4 address.

An IPv4-compatible address is globally unique as long as the IPv4 address is not from the private IPv4 address space [15]. An implementation SHOULD behave as if its IPv4-compatible address(es) are assigned to the node's automatic tunneling interface, even if the implementation does not implement automatic tunneling using a concept of interfaces. Thus the IPv4-compatible address SHOULD NOT be viewed as being attached to e.g. an Ethernet interface i.e. implications should not use the Neighbor Discovery mechanisms like NUD [7] at the Ethernet. Any such interactions should be done using the encapsulated packets i.e. over the automatic tunneling (conceptual) interface.

5.2. IPv4-Compatible Address Configuration

An IPv6/IPv4 node with an IPv4-compatible address uses that address as one of its IPv6 addresses, while the IPv4 address embedded in the low-order 32-bits serves as the IPv4 address for one of its interfaces.

An IPv6/IPv4 node MAY acquire its IPv4-compatible IPv6 addresses via IPv4 address configuration protocols. It MAY use any IPv4 address configuration mechanism to acquire its IPv4 address, then "map" that address into an IPv4-compatible IPv6 address by pre-pending it with the 96-bit prefix 0:0:0:0:0:0. This mode of configuration allows IPv6/IPv4 nodes to "leverage" the installed base of IPv4 address configuration servers.

The specific algorithm for acquiring an IPv4-compatible address using IPv4-based address configuration protocols is as follows:

1) The IPv6/IPv4 node uses standard IPv4 mechanisms or protocols to acquire the IPv4 address for one of its interfaces. These

include:

- The Dynamic Host Configuration Protocol (DHCP) [2]
- The Bootstrap Protocol (BOOTP) [1]
- The Reverse Address Resolution Protocol (RARP) [9]
- Manual configuration
- Any other mechanism which accurately yields the node's own IPv4 address

2) The node uses this address as the IPv4 address for this interface.

3) The node prepends the 96-bit prefix 0:0:0:0:0:0 to the 32-bit IPv4 address that it acquired in step (1). The result is an IPv4-compatible IPv6 address with one of the node's IPv4-addresses embedded in the low-order 32-bits. The node uses this address as one of its IPv6 addresses.

5.3. Automatic Tunneling Operation

In automatic tunneling, the tunnel endpoint address is determined from the packet being tunneled. If the destination IPv6 address is IPv4-compatible, then the packet can be sent via automatic tunneling. If the destination is IPv6-native, the packet can not be sent via automatic tunneling.

A routing table entry can be used to direct automatic tunneling. An implementation can have a special static routing table entry for the prefix 0:0:0:0:0:0/96. (That is, a route to the all-zeros prefix with a 96-bit mask.) Packets that match this prefix are sent to a pseudo-interface driver which performs automatic tunneling. Since all IPv4-compatible IPv6 addresses will match this prefix, all packets to those destinations will be auto-tunneled.

Once it is delivered to the automatic tunneling module, the IPv6 packet is encapsulated within an IPv4 header according to the rules described in section 3. The source and destination addresses of the encapsulating IPv4 header are assigned as follows:

Destination IPv4 address:

Low-order 32-bits of IPv6 destination address

Source IPv4 address:

IPv4 address of interface the packet is sent via

The automatic tunneling module always sends packets in this encapsulated form, even if the destination is on an attached datalink.

The automatic tunneling module MUST NOT send to IPv4 broadcast or multicast destinations. It MUST drop all IPv6 packets destined to IPv4-compatible destinations when the embedded IPv4 address is broadcast, multicast, the unspecified (0.0.0.0) address, or the loopback address (127.0.0.1). Note that the sender can only tell if an address is a network or subnet broadcast for broadcast addresses assigned to directly attached links.

5.4. Use With Default Configured Tunnels

Automatic tunneling is often used in conjunction with the default configured tunnel technique. "Isolated" IPv6/IPv4 hosts -- those with no on-link IPv6 routers -- are configured to use automatic tunneling and IPv4-compatible IPv6 addresses, and have at least one default configured tunnel to an IPv6 router. That IPv6 router is configured to perform automatic tunneling as well. These isolated hosts send packets to IPv4-compatible destinations via automatic tunneling and packets for IPv6-native destinations via the default configured tunnel. IPv4-compatible destinations will match the 96-bit all-zeros prefix route discussed in the previous section, while IPv6-native destinations will match the default route via the configured tunnel. Reply packets from IPv6-native destinations are routed back to the an IPv6/IPv4 router which delivers them to the original host via automatic tunneling. Further examples of the combination of tunneling techniques are discussed in [12].

5.5. Source Address Selection

When an IPv6/IPv4 node originates an IPv6 packet, it must select the source IPv6 address to use. IPv6/IPv4 nodes that are configured to perform automatic tunneling may be configured with global IPv6-native addresses as well as IPv4-compatible addresses. The selection of which source address to use will determine what form the return traffic is sent via. If the IPv4-compatible address is used, the return traffic will have to be delivered via automatic tunneling, but if the IPv6-native address is used, the return traffic will not be

automatic-tunneled. In order to make traffic as symmetric as possible, the following source address selection preference is RECOMMENDED:

Destination is IPv4-compatible:

Use IPv4-compatible source address associated with IPv4 address of outgoing interface

Destination is IPv6-native:

Use IPv6-native address of outgoing interface

If an IPv6/IPv4 node has no global IPv6-native address, but is originating a packet to an IPv6-native destination, it MAY use its IPv4-compatible address as its source address.

5.6. Ingress Filtering

The decapsulating node MUST verify that the encapsulated packets are acceptable before forwarding decapsulated packets to avoid circumventing ingress filtering [13]. Note that packets which are delivered to transport protocols on the decapsulating node SHOULD NOT be subject to these checks. Since automatic tunnels always encapsulate to the destination (i.e. the IPv4 destination will be the destination) any packet received over an automatic tunnel SHOULD NOT be forwarded.

6. Acknowledgments

We would like to thank the members of the IPng working group and the Next Generation Transition (ngtrans) working group for their many contributions and extensive review of this document. Special thanks are due to Jim Bound, Ross Callon, and Bob Hinden for many helpful suggestions and to John Moy for suggesting the IPv4 "anycast address" default tunnel technique.

7. Security Considerations

Tunneling is not known to introduce any security holes except for the possibility to circumvent ingress filtering [13]. This is prevented by requiring that decapsulating routers only forward packets if they have been configured to accept encapsulated packets from the IPv4 source address in the receive packet. Additionally, in the case of automatic tunneling, nodes are required by not forwarding the decapsulated packets since automatic tunneling ends the tunnel and the destination.

8. Authors' Addresses

Robert E. Gilligan
FreeGate Corp
1208 E. Arques Ave
Sunnyvale, CA 94086
USA

Phone: +1-408-617-1004
Fax: +1-408-617-1010
Email: gilligan@freegate.com

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

Phone: +1-650-786-5166
Fax: +1-650-786-5896
Email: nordmark@eng.sun.com

9. References

[1] Croft, W., and J. Gilmore, "Bootstrap Protocol", RFC 951, September 1985.

[2] Droms, R., "Dynamic Host Configuration Protocol", RFC 1541. October 1993.

[3] Carpenter, B., and Jung, C. "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels", RFC 2529, March 1999.

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

[5] Thomson, S., and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC 2462, December 1998.

[6] Crawford, M., Thomson, S., and C. Huitema. "DNS Extensions to Support IPv6 Address Allocation and Renumbering", draft-ietf-ipngwg-dns-lookups-07.txt

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

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

[9] Finlayson, R., Mann, T., Mogul, J., and M. Theimer, "Reverse Address Resolution Protocol", RFC 903, June 1984.

[10] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[11] Kent, C., and J. Mogul, "Fragmentation Considered Harmful". In Proc. SIGCOMM '87 Workshop on Frontiers in Computer Communications Technology. August 1987.

[12] Callon, R. and Haskin, D., "Routing Aspects of IPv6 Transition", RFC 2185. September 1997.

[13] Ferguson, P., and Senie, D., "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", RFC 2267, January 1998.

[14] Hinden, R., and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[15] Rechter, Y., Moskowitz, B., Karrenberg, D., de Groot, G.J., and Lear, E. "Address Allocation for Private Internets", RFC 1918, February 1996.

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

[17] D. Thaler, "IP Tunnel MIB", RFC 2667, August 1999.

[18] F. Baker, "Requirements for IP Version 4 Routers", RFC 1812, June 1995.

10. Changes from RFC 1933

- Deleted section 3.1.1 (IPv4 loopback address) in order to prevent it from being mis-construed as requiring routers to filter the address ::127.0.0.1, which would put another test in the forwarding path for IPv6 routers.

- Deleted section 4.4 (Default Sending Algorithm). This section allowed nodes to send packets in "raw form" to IPv4-compatible destinations on the same datalink. Implementation experience has shown that this adds complexity which is not justified by the minimal savings in header overhead.

- Added definitions for operating modes for IPv6/IPv4 nodes.

- Revised DNS section to clarify resolver filtering and ordering options.

- Re-wrote the discussion of IPv4-compatible addresses to clarify that they are used exclusively in conjunction with the automatic tunneling mechanism. Re-organized document to place definition of IPv4-compatible address format with description of automatic tunneling.

- Changed the term "IPv6-only address" to "IPv6-native address" per current usage.

- Updated to algorithm for determining tunnel MTU to reflect the change in the IPv6 minimum MTU from 576 to 1280 bytes [4].

- Deleted the definition for the term "IPv6-in-IPv4 encapsulation." It has not been widely used.

- Revised IPv4-compatible address configuration section (5.2) to recognize multiple interfaces.

- Added discussion of source address selection when using IPv4-compatible addresses.

- Added section on the combination of the default configured tunneling technique with hosts using automatic tunneling.

- Added prohibition against automatic tunneling to IPv4 broadcast or multicast destinations.

- Clarified that configured tunnels can be unidirectional or

bidirectional.

- Added description of bidirectional virtual links as another type of tunnels. Nodes MUST respond to NUD probes on such links and SHOULD send NUD probes.

- Added reference to [16] specification as an alternative for tunneling over a multicast capable IPv4 cloud.

- Clarified that IPv4-compatible addresses are assigned exclusively to nodes that support automatic tunnels i.e. nodes that can receive such packets.

- Added text about formation of link-local addresses and use of Neighbor Discovery on tunnels.

- Added restriction that decapsulated packets not be forwarded unless the source address is acceptable to the decapsulating router.

- Clarified that decapsulating nodes MUST be capable of reassembling an IPv4 packet that is 1300 bytes (1280 bytes plus IPv4 header).

- Clarified that when using a default tunnel to an IPv4 "anycast address" the network must either have an IPv4 MTU of least 1300 bytes (to avoid fragmentation of minimum size IPv6 packets) or be configured to avoid frequent changes to IPv4 routing to the "anycast address" (to avoid different IPv4 fragments arriving at different tunnel endpoints).

- Using A6/AAAA instead of AAAA to reference IPv6 address records in the DNS.

- Specified when to put IPv6 addresses in the DNS.

- Added reference to the tunnel mib for TTL specification for the tunnels.

- Added a table of contents.

- Added recommendations for use of source and target link layer address options for the tunnel links.

- Added checks in the decapsulation checking both an IPv4-compatible IPv6 source address and the outer IPv4 source addresses for multicast, broadcast, all-zeros etc.