                Basic Transition Mechanisms for IPv6 Hosts and Routers

Status of This Memo

Copyright Notice

Abstract

   This document specifies IPv4 compatibility mechanisms that can be
   implemented by IPv6 hosts and routers. Two mechanisms are specified,
   dual stack and configured tunneling. Dual stack implies providing
   complete implementations of both versions of the Internet Protocol
   (IPv4 and IPv6), and configured tunneling provides a means to carry
   IPv6 packets over unmodified IPv4 routing infrastructures.

   This document obsoletes RFC 2893.

Table of 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
   two 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.

   The mechanisms specified here are:

   - 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: A technique for
      establishing point-to-point tunnels by encapsulating IPv6 packets
      within IPv4 headers to carry them over IPv4 routing
      infrastructures.

   The mechanisms defined here are intended to be the core of a
   "transition toolbox" -- a growing collection of techniques that
   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 basic set of transition mechanisms, but
   these are not the only tools available. Additional transition and
   compatibility mechanisms are specified in other documents.

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 in
         this memo.

      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.

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(es) are determined by configuration information on
        tunnel endpoints. All tunnels are assumed to be bidirectional.
        The tunnel provides a (virtual) point-to-point link to the IPv6
        layer, using the configured IPv4 addresses as the lower-layer
        endpoint addresses.

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

    The key words 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 [RFC2119].

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 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. Here we use
    a rather loose notion of "stack". A stack being enabled has IP
    addresses assigned, but whether or not any particular application is
    available on the stacks is explicitly not defined. 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 configured tunneling technique, which is described in Section 3,
   may or may not be used in addition to the dual IP layer operation.

2.1. Address Configuration

   Because the nodes 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
   [RFC2462] and/or DHCPv6) to acquire their IPv6 addresses.

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
   "AAAA" has been defined for IPv6 addresses [RFC3596]. 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 "AAAA" records. Note
   that the lookup of A versus AAAA records is independent of whether
   the DNS packets are carried in IPv4 or IPv6 packets and that there is
   no assumption that the DNS servers know the IPv4/IPv6 capabilities of
   the requesting node.

   The issues and operational guidelines for using IPv6 with DNS are
   described at more length in other documents, e.g., [DNSOPV6].

   DNS resolver libraries on IPv6/IPv4 nodes MUST be capable of handling
   both AAAA and A records. However, when a query locates an AAAA
   record holding an IPv6 address, and an A record holding an IPv4
   address, the resolver library MAY order the results returned to the
   application in order to influence the version of IP packets used to
   communicate with that specific node -- IPv6 first, or IPv4 first.

   The applications SHOULD be able to specify whether they want IPv4,
   IPv6, or both records [RFC3493]. That defines which address families
   the resolver looks up. If there is not an application choice, or if
   the application has requested both, the resolver library MUST NOT
   filter out any records.

   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 actual ordering mechanisms are out of scope of this memo.
Address selection is described at more length in [RFC3484].

3. Configured 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.

Configured tunneling can be used in all of the above cases, but it is
most likely to be used router-to-router due to the need to explicitly
configure the tunneling endpoints.

The underlying mechanisms for tunneling are:

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

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

   - The encapsulator may need to maintain soft-state information for
      each tunnel recording such parameters as the MTU of the tunnel in
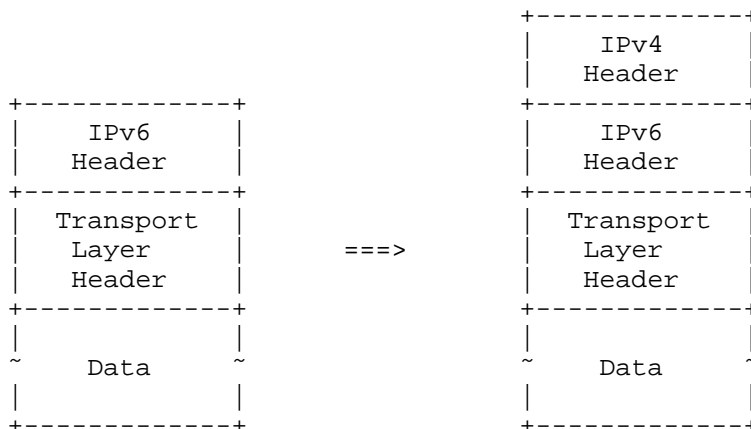   order to process IPv6 packets forwarded into the tunnel.

   In configured tunneling, the tunnel endpoint addresses are determined
   in the encapsulator from configuration information stored for each
   tunnel. When an IPv6 packet is transmitted over a tunnel, the
   destination and source addresses for the encapsulating IPv4 header
   are set as described in Section 3.5.

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

   The decapsulator matches the received protocol-41 packets to the
   tunnels it has configured, and allows only the packets in which IPv4
   source addresses match the tunnels configured on the decapsulator.
   Therefore, the operator must ensure that the tunnel's IPv4 address
   configuration is the same both at the encapsulator and the
   decapsulator.

3.1. Encapsulation

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

```
                                              +-------------+
                                              |    IPv4     |
                                              |   Header    |
                     +-------------+          +-------------+
                     |    IPv6     |          |    IPv6     |
                     |   Header    |          |   Header    |
                     +-------------+          +-------------+
                     |  Transport  |          |  Transport  |
                     |    Layer    |   ===>   |    Layer    |
                     |   Header    |          |   Header    |
                     +-------------+          +-------------+
                     |             |          |             |
                     ~    Data     ~          ~    Data     ~
                     |             |          |             |
                     +-------------+          +-------------+
```

                    Encapsulating IPv6 in IPv4

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

   - Determine when to fragment and when to report an ICMPv6 "packet
      too big" error back to the source.

   - How to reflect ICMPv4 errors from routers along the tunnel path
      back to the source as ICMPv6 errors.

   Those issues are discussed in the following sections.

3.2. Tunnel MTU and Fragmentation

   Naively, the encapsulator could view encapsulation as IPv6 using IPv4
   as a link layer with a very large MTU (65535-20 bytes at most; 20
   bytes "extra" are needed for the encapsulating IPv4 header). The
   encapsulator would only need to report ICMPv6 "packet too big" errors
   back to the source for packets that exceed this MTU. However, such a
   scheme would be inefficient or non-interoperable for three reasons
   and therefore MUST NOT be used:

   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
      [KM97].

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

   3) The encapsulator has no way of knowing that the decapsulator is
      able to defragment such IPv4 packets (see Section 3.6 for
      details), and has no way of knowing that the decapsulator is able
      to handle such a large IPv6 Maximum Receive Unit (MRU).

   Hence, the encapsulator MUST NOT treat the tunnel as an interface
   with an MTU of 64 kilobytes, but instead either use the fixed static
   MTU or OPTIONAL dynamic MTU determination based on the IPv4 path MTU
   to the tunnel endpoint.

   If both the mechanisms are implemented, the decision of which to use
   SHOULD be configurable on a per-tunnel endpoint basis.

3.2.1. Static Tunnel MTU

   A node using static tunnel MTU treats the tunnel interface as having
   a fixed-interface MTU. By default, the MTU MUST be between 1280 and
   1480 bytes (inclusive), but it SHOULD be 1280 bytes. If the default
   is not 1280 bytes, the implementation MUST have a configuration knob
   that can be used to change the MTU value.

   A node must be able to accept a fragmented IPv6 packet that, after
   reassembly, is as large as 1500 octets [RFC2460]. This memo also
   includes requirements (see Section 3.6) for the amount of IPv4
   reassembly and IPv6 MRU that MUST be supported by all the
   decapsulators. These ensure correct interoperability with any fixed
   MTUs between 1280 and 1480 bytes.

   A larger fixed MTU than supported by these requirements must not be
   configured unless it has been administratively ensured that the
   decapsulator can reassemble or receive packets of that size.

   The selection of a good tunnel MTU depends on many factors, at least:

   - Whether the IPv4 protocol-41 packets will be transported over
      media that may have a lower path MTU (e.g., IPv4 Virtual Private
      Networks); then picking too high a value might lead to IPv4
      fragmentation.

   - Whether the tunnel is used to transport IPv6 tunneled packets
      (e.g., a mobile node with an IPv6-in-IPv4 configured tunnel, and
      an IPv6-in-IPv6 tunnel interface); then picking too low a value
      might lead to IPv6 fragmentation.

   If layered encapsulation is believed to be present, it may be prudent
   to consider supporting dynamic MTU determination instead as it is
   able to minimize fragmentation and optimize packet sizes.

   When using the static tunnel MTU, the Don't Fragment bit MUST NOT be
   set in the encapsulating IPv4 header. As a result, the encapsulator
   should not receive any ICMPv4 "packet too big" messages as a result
   of the packets it has encapsulated.

3.2.2. Dynamic Tunnel MTU

   The dynamic MTU determination is OPTIONAL. However, if it is
   implemented, it SHOULD have the behavior described in this document.

   The fragmentation inside the tunnel can be reduced to a minimum by
   having the encapsulator track the IPv4 path MTU across the tunnel,
   using the IPv4 Path MTU Discovery Protocol [RFC1191] and recording

the resulting path MTU. The IPv6 layer in the encapsulator 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 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 [RFC2460].) In this case, the IPv6 layer has to "see" a link
layer with an MTU of 1280 bytes and the encapsulator has to use IPv4
fragmentation in order to forward the 1280 byte IPv6 packets.

The encapsulator SHOULD 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 ICMPv6 "packet
too big" message per [RFC1981]:

```
    if (IPv4 path MTU - 20) is less than 1280
            if packet is larger than 1280 bytes
                    Send ICMPv6 "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 encapsulator or by some router along
                    the IPv4 path.
            endif
    else
            if packet is larger than (IPv4 path MTU - 20)
                    Send ICMPv6 "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
```

Encapsulators that have a large number of tunnels may choose between
dynamic versus static tunnel MTUs on a per-tunnel endpoint basis. In
cases where the number of tunnels that any one node is using is
large, it is helpful to observe that this state information can be
cached and discarded when not in use.

Note that using dynamic tunnel MTU is subject to IPv4 path MTU
blackholes should the ICMPv4 "packet too big" messages be dropped by
firewalls or not generated by the routers [RFC1435, RFC2923].

3.3. Hop Limit

   IPv6-over-IPv4 tunnels are modeled as "single-hop" from the IPv6
   perspective. The tunnel is opaque to users of the network, and it is
   not detectable by network diagnostic tools such as traceroute.

   The single-hop model is implemented by having the encapsulators and
   decapsulators 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 [RFC3232][ASSIGNED].
   Implementations MAY provide a mechanism to allow the administrator to
   configure the IPv4 TTL as the IP Tunnel MIB [RFC4087].

3.4. Handling ICMPv4 Errors

   In response to encapsulated packets it has sent into the tunnel, the
   encapsulator might receive ICMPv4 error messages from IPv4 routers
   inside the tunnel. These packets are addressed to the encapsulator
   because it is the IPv4 source of the encapsulated packet.
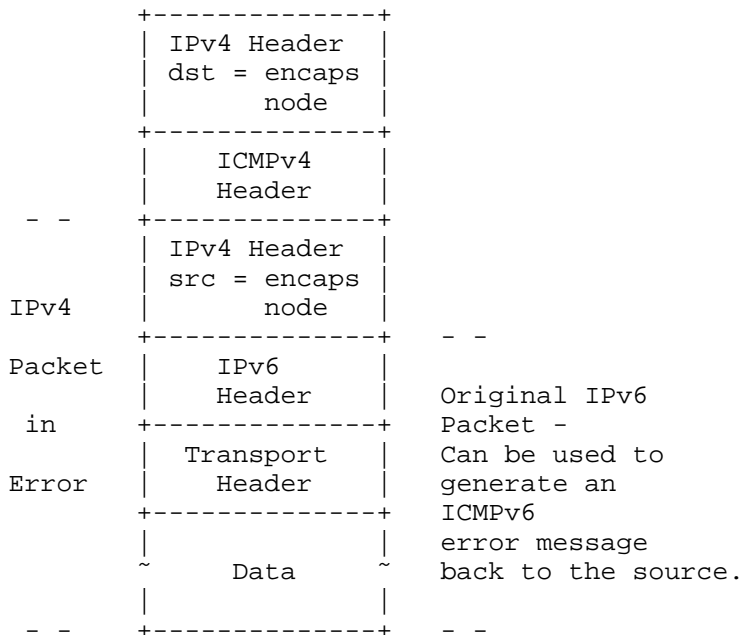
   ICMPv4 error handling is only applicable to dynamic MTU
   determination, even though the functions could be used with static
   MTU tunnels as well.

   The ICMPv4 "packet too big" error messages are handled according to
   IPv4 Path MTU Discovery [RFC1191] and the resulting path MTU is
   recorded in the IPv4 layer. The recorded path MTU is used by IPv6 to
   determine if an ICMPv6 "packet too big" error has to be generated as
   described in Section 3.2.2.

   The handling of other types of ICMPv4 error messages depends on how
   much information is available from 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. See
   [RFC1812].

If sufficient data bytes from the offending packet are available, the
encapsulator MAY extract the encapsulated IPv6 packet and use it to
generate an ICMPv6 message directed back to the originating IPv6
node, as shown below:

```
                      +--------------+
                      | IPv4 Header  |
                      | dst = encaps |
                      |     node     |
                      +--------------+
                      |    ICMPv4    |
                      |    Header    |
        - -     +--------------+
                      | IPv4 Header  |
                      | src = encaps |
            IPv4  |     node     |
                      +--------------+   - -
            Packet  |    IPv6      |
                      |    Header    |   Original IPv6
             in     +--------------+   Packet -
                      |  Transport   |   Can be used to
            Error  |    Header    |   generate an
                      +--------------+   ICMPv6
                      |              |   error message
                      ~     Data     ~   back to the source.
                      |              |
        - -     +--------------+   - -
```

            ICMPv4 Error Message Returned to Encapsulating Node

When receiving ICMPv4 errors as above and the errors are not "packet
too big", it would be useful to log the error as an error related to
the tunnel. Also, if sufficient headers are available, then the
originating node MAY send an ICMPv6 error of type "unreachable" with
code "address unreachable" to the IPv6 source. (The "address
unreachable" code is appropriate since, from the perspective of IPv6,
the tunnel is a link and that code is used for link-specific errors
[RFC2463]).

Note that when the IPv4 path MTU is exceeded, and sufficient bytes of
payload associated with the ICMPv4 errors are not available, or
ICMPv4 errors do not cause the generation of ICMPv6 errors in case
there is enough payload, there will be at least two packet drops
instead of at least one (the case of a single layer of MTU
discovery). Consider a case where an IPv6 host is connected to an
IPv4/IPv6 router, which is connected to a network where an ICMPv4
error about too big packet size is generated. First, the router
needs to learn the tunnel (IPv4) MTU that causes at least one packet

loss, and then the host needs to learn the (IPv6) MTU from the router
that causes at least one packet loss. Still, in all cases there can
be more than one packet loss if there are multiple large packets in
flight at the same time.

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 unless otherwise specified. (See [RFC2983] and [RFC3168]
         Section 9.1 for issues relating to the Type-of-Service byte and
         tunneling.)

      Total Length:

         Payload length from IPv6 header plus length of IPv6 and IPv4
         headers (i.e., IPv6 payload length plus 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 an implementation-specific manner, as described in
         Section 3.3.

Protocol:

    41 (Assigned payload type number for IPv6).

Header Checksum:

    Calculate the checksum of the IPv4 header [RFC791].

Source Address:

    An IPv4 address of the encapsulator: either configured by the
    administrator or an address of the outgoing interface.

Destination Address:

    IPv4 address of the tunnel endpoint.

When encapsulating the packets, the node must ensure that it will use
the correct source address so that the packets are acceptable to the
decapsulator as described in Section 3.6. Configuring the source
address is appropriate particularly in cases in which automatic
selection of source address may produce different results in a
certain period of time. This is often the case with multiple
addresses, and multiple interfaces, or when routes may change
frequently. Therefore, it SHOULD be possible to administratively
specify the source address of a tunnel.

3.6. Decapsulation

When an IPv6/IPv4 host or a router receives an IPv4 datagram that is
addressed to one of its own IPv4 addresses or a joined multicast
group address, and the value of the protocol field is 41, the packet
is potentially a tunnel packet and needs to be verified to belong to
one of the configured tunnel interfaces (by checking
source/destination addresses), reassembled (if fragmented at the IPv4
level), and have the IPv4 header removed and the resulting IPv6
datagram be submitted to the IPv6 layer code on the node.

The decapsulator MUST verify that the tunnel source address is
correct before further processing packets, to mitigate the problems
with address spoofing (see Section 4). This check also applies to
packets that are delivered to transport protocols on the
decapsulator. This is done by verifying that the source address is
the IPv4 address of the encapsulator, as configured on the
decapsulator. Packets for which the IPv4 source address does not
match MUST be discarded and an ICMP message SHOULD NOT be generated;

however, if the implementation normally sends an ICMP message when
receiving an unknown protocol packet, such an error message MAY be
sent (e.g., ICMPv4 Protocol 41 Unreachable).

A side effect of this address verification is that the node will
silently discard packets with a wrong source address and packets that
were received by the node but not directly addressed to it (e.g.,
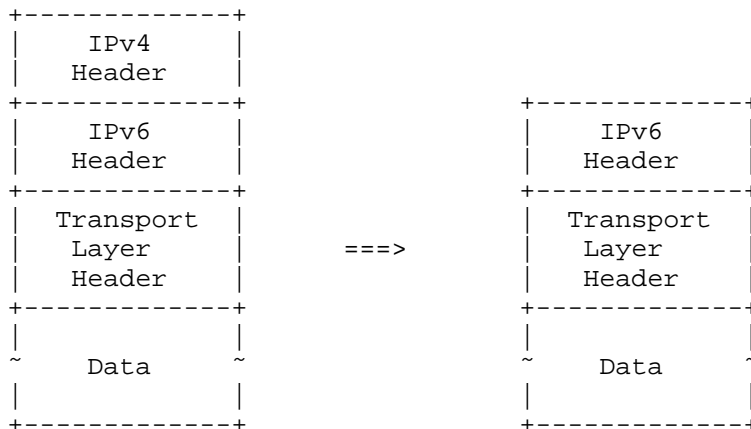broadcast addresses).

Independent of any other forms of IPv4 ingress filtering the
administrator of the node may have configured, the implementation MAY
perform ingress filtering, i.e., check that the packet is arriving
from the interface in the direction of the route toward the tunnel
end-point, similar to a Strict Reverse Path Forwarding (RPF) check
[RFC3704]. As this may cause problems on tunnels that are routed
through multiple links, it is RECOMMENDED that this check, if done,
is disabled by default. The packets caught by this check SHOULD be
discarded; an ICMP message SHOULD NOT be generated by default.

The decapsulator MUST be capable of having, on the tunnel interfaces,
an IPv6 MRU of at least the maximum of 1500 bytes and the largest
(IPv6) interface MTU on the decapsulator.

The decapsulator MUST be capable of reassembling an IPv4 packet that
is (after the reassembly) the maximum of 1500 bytes and the largest
(IPv4) interface MTU on the decapsulator. The 1500-byte number is a
result of encapsulators that use the static MTU scheme in Section
3.2.1, while encapsulators that use the dynamic scheme in Section
3.2.2 can cause up to the largest interface MTU on the decapsulator
to be received. (Note that it is strictly the interface MTU on the
last IPv4 router *before* the decapsulator that matters, but for most
links the MTU is the same between all neighbors.)

This reassembly limit allows dynamic tunnel MTU determination by the
encapsulator to take advantage of larger IPv4 path MTUs. An
implementation MAY have a configuration knob that can be used to set
a larger value of the tunnel reassembly buffers than the above
number, but it MUST NOT be set below the above number.

The decapsulation is shown below:

```
        +-------------+
        |    IPv4     |
        |   Header    |
        +-------------+                     +-------------+
        |    IPv6     |                     |    IPv6     |
        |   Header    |                     |   Header    |
        +-------------+                     +-------------+
        |  Transport  |                     |  Transport  |
        |   Layer     |         ===>        |   Layer     |
        |   Header    |                     |   Header    |
        +-------------+                     +-------------+
        |             |                     |             |
        ~    Data     ~                     ~    Data     ~
        |             |                     |             |
        +-------------+                     +-------------+
```

                   Decapsulating IPv6 from IPv4

   The decapsulator performs IPv4 reassembly before decapsulating the
   IPv6 packet.

   When decapsulating the packet, the IPv6 header is not modified.
   (However, see [RFC2983] and [RFC3168] section 9.1 for issues relating
   to the Type of Service byte and tunneling.) If the packet is
   subsequently forwarded, its hop limit is decremented by one.

   The encapsulating IPv4 header is discarded, and the resulting packet
   is checked for validity when submitted to the IPv6 layer. When
   reconstructing the IPv6 packet, the length MUST be determined from
   the IPv6 payload length since the IPv4 packet might be padded (thus
   have a length that is larger than the IPv6 packet plus the IPv4
   header being removed).

   After the decapsulation, the node MUST silently discard a packet with
   an invalid IPv6 source address. The list of invalid source addresses
   SHOULD include at least:

   - all multicast addresses (FF00::/8)

   - the loopback address (::1)

   - all the IPv4-compatible IPv6 addresses [RFC3513] (::/96),
      excluding the unspecified address for Duplicate Address Detection
      (::/128)

   - all the IPv4-mapped IPv6 addresses (::ffff:0:0/96)

In addition, the node should be configured to perform ingress
filtering [RFC2827][RFC3704] on the IPv6 source address, similar to
on any of its interfaces, e.g.:

1) if the tunnel is toward the Internet, the node should be
   configured to check that the site's IPv6 prefixes are not used as
   the source addresses, or

2) if the tunnel is toward an edge network, the node should be
   configured to check that the source address belongs to that edge
   network.

The prefix lists in the former typically need to be manually
configured; the latter could be verified automatically, e.g., by
using a strict unicast RPF check, as long as an interface can be
designated to be toward an edge.

It is RECOMMENDED that the implementations provide a single knob to
make it easier to for the administrators to enable strict ingress
filtering toward edge networks.

3.7. Link-Local Addresses

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

The interface identifier [RFC3513] for such an interface may be based
on the 32-bit IPv4 address of an underlying interface, or formed
using some other means, as long as it is unique from the other tunnel
endpoint with a reasonably high probability.

Note that it may be desirable to form the link-local address in a
fashion that minimizes the probability and the effect of having to
renumber the link-local address in the event of a topology or
hardware change.

If an IPv4 address is used for forming the IPv6 link-local address,
the interface identifier is the IPv4 address, prepended by zeros.
Note that the "Universal/Local" bit is zero, indicating that the
interface identifier is not globally unique. The link-local address
is formed by appending the interface identifier to the prefix
FE80::/64.

When the host has more than one IPv4 address in use on the physical
interface concerned, a choice of one of these IPv4 addresses is made

by the administrator or the implementation when forming the link-
local address.

```
+-------+-------+-------+-------+-------+-------+------+------+
|  FE      80      00      00      00      00      00     00  |
+-------+-------+-------+-------+-------+-------+------+------+
|  00      00      00      00   |        IPv4 Address        |
+-------+-------+-------+-------+-------+-------+------+------+
```

3.8. Neighbor Discovery over Tunnels

   Configured tunnel implementations MUST at least accept and respond to
   the probe packets used by Neighbor Unreachability Detection (NUD)
   [RFC2461]. The 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 configured tunnels
   specified in this document are assumed to NOT have a link-layer
   address, even though the link-layer (IPv4) does have an address.
   This means that:

   - the sender of Neighbor Discovery packets SHOULD NOT include Source
      Link Layer Address options or Target Link Layer Address options on
      the tunnel link.

   - the receiver MUST, while otherwise processing the Neighbor
      Discovery packet, silently ignore the content of any Source Link
      Layer Address options or Target Link Layer Address options
      received on the tunnel link.

   Not using link-layer address options is consistent with how Neighbor
   Discovery is used on other point-to-point links.

4. Threat Related to Source Address Spoofing

   The specification above contains rules that apply tunnel source
   address verification in particular and ingress filtering
   [RFC2827][RFC3704] in general to packets before they are
   decapsulated. When IP-in-IP tunneling (independent of IP versions)
   is used, it is important that this not be used to bypass any ingress
   filtering in use for non-tunneled packets. Thus, the rules in this
   document are derived based on should ingress filtering be used for
   IPv4 and IPv6, the use of tunneling should not provide an easy way to
   circumvent the filtering.

In this case, without specific ingress filtering checks in the
decapsulator, it would be possible for an attacker to inject a packet
with:

- Outer IPv4 source: real IPv4 address of attacker

- Outer IPv4 destination: IPv4 address of decapsulator

- Inner IPv6 source: Alice, which is either the decapsulator or a
  node close to it

- Inner IPv6 destination: Bob

Even if all IPv4 routers between the attacker and the decapsulator
implement IPv4 ingress filtering, and all IPv6 routers between the
decapsulator and Bob implement IPv6 ingress filtering, the above
spoofed packets will not be filtered out. As a result, Bob will
receive a packet that looks like it was sent from Alice even though
the sender was some unrelated node.

The solution to this is to have the decapsulator accept only
encapsulated packets from the explicitly configured source address
(i.e., the other end of the tunnel) as specified in Section 3.6.
While this does not provide complete protection in the case ingress
filtering has not been deployed, it does provide a significant
increase in security. The issue and the remainder threats are
discussed at more length in Security Considerations.

5. Security Considerations

   Generic security considerations of using IPv6 are discussed in a
   separate document [V6SEC].

   An implementation of tunneling needs to be aware that although a
   tunnel is a link (as defined in [RFC2460]), the threat model for a
   tunnel might be rather different than for other links, since the
   tunnel potentially includes all of the Internet.

   Several mechanisms (e.g., Neighbor Discovery) depend on Hop Count
   being 255 and/or the addresses being link local for ensuring that a
   packet originated on-link, in a semi-trusted environment. Tunnels
   are more vulnerable to a breach of this assumption than physical
   links, as an attacker anywhere in the Internet can send an IPv6-in-
   IPv4 packet to the tunnel decapsulator, causing injection of an
   encapsulted IPv6 packet to the configured tunnel interface unless the
   decapsulation checks are able to discard packets injected in such a
   manner.

   Therefore, this memo specifies that the decapsulators make these
   steps (as described in Section 3.6) to mitigate this threat:

   - IPv4 source address of the packet MUST be the same as configured
      for the tunnel end-point;

   - Independent of any IPv4 ingress filtering the administrator may
      have configured, the implementation MAY perform IPv4 ingress
      filtering to check that the IPv4 packets are received from an
      expected interface (but as this may cause some problems, it may be
      disabled by default);

   - IPv6 packets with several, obviously invalid IPv6 source addresses
      received from the tunnel MUST be discarded (see Section 3.6 for
      details); and

   - IPv6 ingress filtering should be performed (typically requiring
      configuration from the operator), to check that the tunneled IPv6
      packets are received from an expected interface.

   Especially the first verification is vital: to avoid this check, the
   attacker must be able to know the source of the tunnel (ranging from
   difficult to predictable) and be able to spoof it (easier).

   If the remainder threats of tunnel source verification are considered
   to be significant, a tunneling scheme with authentication should be
   used instead, e.g., IPsec [RFC2401] (preferable) or Generic Routing
   Encapsulation with a pre-configured secret key [RFC2890]. As the
   configured tunnels are set up more or less manually, setting up the
   keying material is probably not a problem. However, setting up
   secure IPsec IPv6-in-IPv4 tunnels is described in another document
   [V64IPSEC].

   If the tunneling is done inside an administrative domain, proper
   ingress filtering at the edge of the domain can also eliminate the
   threat from outside of the domain. Therefore, shorter tunnels are
   preferable to longer ones, possibly spanning the whole Internet.

   In addition, an implementation MUST treat interfaces to different
   links as separate, e.g., to ensure that Neighbor Discovery packets
   arriving on one link do not affect other links. This is especially
   important for tunnel links.

   When dropping packets due to failing to match the allowed IPv4 source
   addresses for a tunnel the node should not "acknowledge" the
   existence of a tunnel, otherwise this could be used to probe the
   acceptable tunnel endpoint addresses. For that reason, the
   specification says that such packets MUST be discarded, and an ICMP

error message SHOULD NOT be generated, unless the implementation
normally sends ICMP destination unreachable messages for unknown
protocols; in such a case, the same code MAY be sent. As should be
obvious, not returning the same ICMP code if an error is returned for
other protocols may hint that the IPv6 stack (or the protocol 41
tunneling processing) has been enabled -- the behaviour should be
consistent on how the implementation otherwise behaves to be
transparent to probing.

6. Acknowledgements

   We would like to thank the members of the IPv6 working group, the
   Next Generation Transition (ngtrans) working group, and the v6ops
   working group for their many contributions and extensive review of
   this document. Special thanks are due to (in alphabetical order) Jim
   Bound, Ross Callon, Tim Chown, Alex Conta, Bob Hinden, Bill Manning,
   John Moy, Mohan Parthasarathy, Chirayu Patel, Pekka Savola, and Fred
   Templin for many helpful suggestions. Pekka Savola helped in editing
   the final revisions of the specification.

7. References

7.1. Normative References

   [RFC791]   Postel, J., "Internet Protocol", STD 5, RFC 791, September
               1981.

   [RFC1191]   Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191,
               November 1990.

   [RFC1981]   McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery
               for IP version 6", RFC 1981, August 1996.

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

   [RFC2460]   Deering, S. and R. Hinden, "Internet Protocol, Version 6
               (IPv6) Specification", RFC 2460, December 1998.

   [RFC2463]   Conta, A. and S. Deering, "Internet Control Message
               Protocol (ICMPv6) for the Internet Protocol Version 6
               (IPv6) Specification", RFC 2463, December 1998.

7.2. Informative References

   [ASSIGNED] IANA, "Assigned numbers online database",
              http://www.iana.org/numbers.html

   [DNSOPV6] Durand, A., Ihren, J., and Savola P., "Operational
              Considerations and Issues with IPv6 DNS", Work in
              Progress, October 2004.

   [KM97]   Kent, C., and J. Mogul, "Fragmentation Considered
              Harmful". In Proc. SIGCOMM '87 Workshop on Frontiers in
              Computer Communications Technology. August 1987.

   [V6SEC]   Savola, P., "IPv6 Transition/Co-existence Security
              Considerations", Work in Progress, October 2004.

   [V64IPSEC] Graveman, R., et al., "Using IPsec to Secure IPv6-over-
              IPv4 Tunnels", Work in Progress, December 2004.

   [RFC1435] Knowles, S., "IESG Advice from Experience with Path MTU
              Discovery", RFC 1435, March 1993.

   [RFC1812] Baker, F., "Requirements for IP Version 4 Routers", RFC
              1812, June 1995.

   [RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the
              Internet Protocol", RFC 2401, November 1998.

   [RFC2461] Narten, T., Nordmark, E., and W. Simpson, "Neighbor
              Discovery for IP Version 6 (IPv6)", RFC 2461, December
              1998.

   [RFC2462] Thomson, S. and T. Narten, "IPv6 Stateless Address
              Autoconfiguration", RFC 2462, December 1998.

   [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.

   [RFC2890] Dommety, G., "Key and Sequence Number Extensions to GRE",
              RFC 2890, September 2000.

   [RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC
              2923, September 2000.

   [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC
              2983, October 2000.

   [RFC3056] Carpenter, B. and K. Moore, "Connection of IPv6 Domains
              via IPv4 Clouds", RFC 3056, February 2001.

   [RFC3168]   Ramakrishnan, K., Floyd, S., and D. Black, "The Addition
               of Explicit Congestion Notification (ECN) to IP", RFC
               3168, September 2001.

   [RFC3232]   Reynolds, J., "Assigned Numbers: RFC 1700 is Replaced by
               an On-line Database", RFC 3232, January 2002.

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

   [RFC3493]   Gilligan, R., Thomson, S., Bound, J., McCann, J., and W.
               Stevens, "Basic Socket Interface Extensions for IPv6", RFC
               3493, February 2003.

   [RFC3513]   Hinden, R. and S. Deering, "Internet Protocol Version 6
               (IPv6) Addressing Architecture", RFC 3513, April 2003.

   [RFC3596]   Thomson, S., Huitema, C., Ksinant, V., and M. Souissi,
               "DNS Extensions to Support IP Version 6", RFC 3596,
               October 2003.

   [RFC3704]   Baker, F. and P. Savola, "Ingress Filtering for Multihomed
               Networks", BCP 84, RFC 3704, March 2004.

   [RFC4087]   Thaler, D., "IP Tunnel MIB", RFC 4087, June 2005.

8. Changes from RFC 2893

   The motivation for the bulk of these changes are to simplify the
   document to only contain the mechanisms of wide-spread use.

   RFC 2893 contains a mechanism called automatic tunneling. But a much
   more general mechanism is specified in RFC 3056 [RFC3056] which gives
   each node with a (global) IPv4 address a /48 IPv6 prefix i.e., enough
   for a whole site.

   The following changes have been performed since RFC 2893:

   - Removed references to A6 and retained AAAA.

   - Removed automatic tunneling and use of IPv4-compatible addresses.

   - Removed default Configured Tunnel using IPv4 "Anycast Address"

   - Removed Source Address Selection section since this is now covered
      by another document ([RFC3484]).

   - Removed brief mention of 6over4.

- Split into normative and non-normative references and other
   reference cleanup.

- Dropped "or equal" in if (IPv4 path MTU - 20) is less than or
   equal to 1280.

- Dropped this: 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.

- Described Static MTU and Dynamic MTU cases separately; clarified
   that the dynamic path MTU mechanism is OPTIONAL but if it is
   implemented it should follow the rules in section 3.2.2.

- Specified Static MTU to default to a MTU of 1280 to 1480 bytes,
   and that this may be configurable. Discussed the issues with
   using Static MTU at more length.

- Specified minimal rules for IPv4 reassembly and IPv6 MRU to
   enhance interoperability and to minimize blacholes.

- Restated the "currently underway" language about Type-of-Service,
   and loosely point at [RFC2983] and [RFC3168].

- Fixed reference to Assigned Numbers to be to online version (with
   proper pointer to "Assigned Numbers is obsolete" RFC).

- Clarified text about ingress filtering e.g., that it applies to
   packet delivered to transport protocols on the decapsulator as
   well as packets being forwarded by the decapsulator, and how the
   decapsulator's checks help when IPv4 and IPv6 ingress filtering is
   in place.

- Removed unidirectional tunneling; assume all tunnels are
   bidirectional, between endpoint addresses (not nodes).

- Removed the guidelines for advertising addresses in DNS as
   slightly out of scope, referring to another document for the
   details.

- Removed the SHOULD requirement that the link-local addresses
   should be formed based on IPv4 addresses.

   - Added a SHOULD for implementing a knob to be able to set the
     source address of the tunnel, and add discussion why this is
     useful.

   - Added stronger wording for source address checks: both IPv4 and
     IPv6 source addresses MUST be checked, and RPF-like ingress
     filtering is optional.

   - Rewrote security considerations to be more precise about the
     threats of tunneling.

   - Added a note about considering using TTL=255 when encapsulating.

   - Added more discussion in Section 3.2 why using an "infinite" IPv6
     MTU leads to likely interoperability problems.

   - Added an explicit requirement that if both MTU determination
     methods are used, choosing one should be possible on a per-tunnel
     basis.

   - Clarified that ICMPv4 error handling is only applicable to dynamic
     MTU determination.

   - Removed/clarified DNS record filtering; an API is a SHOULD and if
     it does not exist, MUST NOT filter anything. Decree ordering out
     of scope, but refer to RFC3484.

   - Add a note that the destination IPv4 address could also be a
     multicast address.

   - Make it RECOMMENDED to provide a toggle to perform strict ingress
     filtering on an interface.

   - Generalize the text on the data in ICMPv4 messages.

   - Made a lot of miscellaneous editorial cleanups.

Authors' Addresses

   Erik Nordmark
   Sun Microsystems
   17 Network Circle
   Menlo Park, CA 94025
   USA

   Phone: +1 650 786 2921
   EMail: erik.nordmark@sun.com

   Robert E. Gilligan
   Intransa, Inc.
   2870 Zanker Rd., Suite 100
   San Jose, CA 95134 USA

   Phone : +1 408 678 8600
   Fax :   +1 408 678 8800
   EMail: bob.gilligan@acm.org