# Appendix A

Excerpts from RFCs

## A.1 Routing Header Pseudo Code

This section contains the pseudo code for the processing of the Routing header, excerpted from the RFC 1883.[1] See also Section 3.2.5.

```
if Segments Left = 0 {
   proceed to process the next header in the packet, whose
   type is identified by the Next Header field in the
   Routing header
}
else if Hdr Ext Len is odd or greater than 46 {
      send an ICMP Parameter Problem, Code 0, message to
      the Source Address, pointing to the Hdr Ext Len
      field, and discard the packet
}
else {
   compute n, the number of addresses in the Routing
   header, by dividing Hdr Ext Len by 2

   if Segments Left is greater than n {
      send an ICMP Parameter Problem, Code 0, message to
      the Source Address, pointing to the Segments Left
      field, and discard the packet
   }
   else {
      decrement Segments Left by 1;
      compute i, the index of the next address to be
      visited in the address vector, by subtracting
      Segments Left from n

      if Address [i] or the IPv6 Destination Address is
      multicast {
         discard the packet
      }
      else {
         swap the IPv6 Destination Address and Address[i]

         if bit i of the Strict/Loose Bit map has value 1
         and the new Destination Address is not the
         address of a neighbor of this node {
            send an ICMP Destination Unreachable - Not a
            Neighbor message to the Source Address and
            discard the packet
         }
```

```
                else if the IPv6 Hop Limit is less than or equal
                to 1 { send an ICMP Time Exceeded - Hop Limit
                Exceeded in Transit message to the Source Address
                and discard the packet
                }
                else {
                    decrement the Hop Limit by 1

                    resubmit the packet to the IPv6 module for
                    transmission to the new destination
                }
            }
        }
    }
```

## A.2 Example of Routing Header Processing

This section contains an example of the application of the algorithm reported in Appendix A.1, excerpted from the RFC 1883. [1] Let's consider the case of a node S that transmits to a node D by using a Routing header that forces the packet being routed through intermediate nodes I1, I2, and I3. The following are significant values assumed by the IPv6 header and by the Routing header fields while the packet propagates:

As the packet travels from S to I1:

```
Source Address = S                  Hdr Ext Len = 6
Destination Address = I1           Segments Left = 3
                                      Address[1] = I2
(if bit 0 of the Bit Map is 1,        Address[2] = I3
 S and I1 must be neighbors;          Address[3] = D
 this is checked by S)
```

As the packet travels from I1 to I2:

```
Source Address = S                  Hdr Ext Len = 6
Destination Address = I2           Segments Left = 2
                                      Address[1] = I1
(if bit 1 of the Bit Map is 1,        Address[2] = I3
 I1 and I2 must be neighbors;          Address[3] = D
 this is checked by I1)
```

As the packet travels from I2 to I3:

```
Source Address = S                    Hdr Ext Len = 6
Destination Address = I3            Segments Left = 1
                                      Address[1] = I1
(if bit 2 of the Bit Map is 1,        Address[2] = I2
 I2 and I3 must be neighbors;          Address[3] = D
 this is checked by I2)
```

As the packet travels from I3 to D:

```
Source Address = S                    Hdr Ext Len = 6
Destination Address = D             Segments Left = 0
                                      Address[1] = I1
(if bit 3 of the Bit Map is 1,        Address[2] = I2
 I3 and D must be neighbors;          Address[3] = I3
 this is checked by I3)
```

## A.3 Processing of ICMPv6 Packets

This section contains a description of the processing of ICMPv6 packets, excerpted from the RFC 1885. [2] See also Section 5.3.

Implementations MUST observe the following rules when processing ICMPv6 messages (from [3]):

a If an ICMPv6 error message of unknown type is received, it MUST be passed to the upper layer.

b If an ICMPv6 informational message of unknown type is received, it MUST be silently discarded.

c Every ICMPv6 error message (type < 128) includes as much of the IPv6 offending (invoking) packet (the packet that caused the error) as will fit without making the error message packet exceed 576 octets.

d In those cases where the internet-layer protocol is required to pass an ICMPv6 error message to the upper-layer protocol, the upper-layer protocol type is extracted from the original packet (contained in the body of the ICMPv6 error message) and used to select the appropriate upper-layer protocol entity to handle the error.

If the original packet had an unusually large amount of extension headers, it is possible that the upper-layer protocol type may not be present in the ICMPv6 message, due to truncation of the original packet to meet the 576-octet limit. In that case, the error message is silently dropped after any IPv6-layer processing.

e An ICMPv6 error message MUST NOT be sent as a result of receiving:
   1 an ICMPv6 error message, or
   2 a packet destined to an IPv6 multicast address (there are two exceptions to this rule: (1) the Packet Too Big Message—Section 3.2—to allow Path MTU discovery to work for IPv6 multicast, and (2) the Parameter Problem Message, Code 2—Section 3.4—reporting an unrecognized IPv6 option that has the Option Type highest-order two bits set to 10), or
   3 a packet sent as a link-layer multicast, (the exception from e.2 applies to this case too), or
   4 a packet sent as a link-layer broadcast, (the exception from e.2 applies to this case too), or
   5 a packet whose source address does not uniquely identify a single node—e.g., the IPv6 Unspecified Address, an IPv6 multicast address, or an address known by the ICMP message sender to be an IPv6 anycast address.

f Finally, to each sender of an erroneous data packet, an IPv6 node MUST limit the rate of ICMPv6 error messages sent, in order to limit the bandwidth and forwarding costs incurred by the error messages when a generator of erroneous packets does not respond to those error messages by ceasing its transmissions.

There are a variety of ways of implementing the rate-limiting function, for example:

1 Timer-based—for example, limiting the rate of transmission of error messages to a given source, or to any source, to at most once every T milliseconds.
2 Bandwidth-based—for example, limiting the rate at which error messages are sent from a particular interface to some fraction F of the attached link's bandwidth.

The limit parameters (e.g., T or F in the above examples) MUST be configurable for the node, with a conservative default value (e.g., T = 1 second, NOT 0 seconds, or F = 2 percent, NOT 100 percent).

## A.4 Addresses to Be Used During the Testing Phase

Addresses to be used during IPv6 tests and in particular in 6bone (see also Chapter 12 and Appendix C, Section C.6) are described in the RFC 1887 [4] of which this appendix contains the most significant part.

The address format for the IPv6 test address is consistent with the provider-based unicast address allocation which is as follows:

```
| 3 |  5 bits  |  16 bits | 8 |   24 bits  | 8 |     64 bits    |
+---+----------+----------+---+------------+---+----------------+
|010|RegistryID|ProviderID|RES|SubscriberID|RES|Intra-Subscriber|
+---+----------+----------+---+------------+---+----------------+
```

The specific allocation of each field of the test address format is as follows:

```
| 3 |  5 bits  |  16 bits | 8 |   24 bits  | 8 | 16 bits|48 bits|
+---+----------+----------+---+------------+---+--------+-------+
|   |          |Autonomous|   |    IPv4    |   | Subnet | Intf. |
|010|  11111   |  System  |RES|   Network  |RES|        |       |
|   |          |  Number  |   |   Address  |   | Address|  ID   |
+---+----------+----------+---+------------+---+--------+-------+
```

where:

010 This is the Format Prefix used to identify provider-based unicast addresses.

11111 This is a Registry ID reserved by the IANA. The initial use of addresses in this Registry ID for IPv6 testing is temporary. All users of these addresses will be required to renumber at some time in the future.

Autonomous System Number This is the current autonomous system number assigned to the provider providing internet service to an IPv6 testers organization. For example for IPv6 testers receiving internet service from BBN Barrnet would use autonomous system number 189. This would be coded in the autonomous system field of the address as follows:

0000 0000 1011 1101 (binary)

The values for the autonomous system number of an organization's provider can be obtained from that provider, or can be looked up in the "whois" database maintained by the internic.net.

RES This field is reserved and must be set to zero.

IPv4 Network Address This is based on the current IPv4 routable address for the subscriber which the interface is connected. It is formed by taking the high order 24 bits of the IPv4 address. For example for an IPv4 address (in IPv4 syntax):

| IPv4 Address |
|---|
| 39.11.22.1 |

the value to put in this field of IPv6 address is:

| IPv4 Format | Hex |
|---|---|
| 39.11.22 | 270B16 |

This technique for generating values for this field only works for subscribers which have IPv4 subscriber prefixes less than equal to 24 bits long. There may be subscribers using IPv4 addresses with longer subscriber prefixes, but this conflict is expected to be very rare. Subscribers with subscriber prefixes larger than 24 bits should use the remaining bits in the IPv4 prefix as the high order bits in the Subnet Address field.

Subnet Address The Subnet ID identifies a specific physical link on which the interface is located. There can be multiple subnets on the same physical link. A specific subnet can not span multiple physical links. The assignment of values for this field is left to an individual subscriber. One possible algorithm to generate values for this field is to use the bits in the IPv4 address which identify the IPv4 subnet.

Interface ID This is the unique identifier of the interface on the link, usually the 48-bit IEEE 802 MAC address of the interface if available.

The following registration form to 6bone contains an example of the application of the techniques described previously:

```
site:            Politecnico di Torino
location:        Torino, ITALY
loc-string:      45 03 52.2n 07 39 43.2e 250m
prefix:          5f15:5000::/32
ping:            5f15:5000:82c0:0e00:bd:800:2bb5:a7a8
tunnel:          130.192.26.254 204.123.2.236 DIGITAL-CA
tunnel:          130.192.26.254 131.175.5.37  CEFRIEL
tunnel:          130.192.26.254 156.148.3.24  CRS4
tunnel:          130.192.26.254 163.162.17.77 CSELT
contact:         silvano.gai@polito.it
status:          operational since 11/1996
remark:          OpenBSD/NRL, DEC RouteAbout Access EW/IPv6
remark:          locally using Bind 4.9.3
changed:         rivetti@csp.it, spera@csp.it 19961220
source:          RIPE
```

The address **5f15:5000:82c0:0e00:bd:800:2bb5:a7a8** is coded following the rules of RFC 1887 [4] and the result is as follows:

- 5f ♦ FP = 010, Registry ID = 11111;
- 1550 ♦ AS = 5456;
- 00 ♦ Reserved;
- 82c00e ♦ IPv4 Network Address = 130.192.15;
- 00 ♦ Reserved;
- bd ♦ Subnet Address = 189;
- 800:2bb5:a7a8® MAC Address.

# A.5 MTU of a Tunnel and Fragmentation

Tunnels are widely used during the migration from IPv4 to IPv6 (see Chapter 12). This section contains the algorithm to transmit an IPv6 packet over a tunnel, when the packet is longer than the tunnel's MTU. This algorithm is described in the RFC 1933 [5].

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 576
        if packet is larger than 576 bytes
                Send IPv6 ICMP "packet too big" with
                MTU = 576.
                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.

# A.6 Transmission of IP Packets

During the migration from IPv4 to IPv6 many nodes will adopt the dual-stack approach (see Chapter 12). When an application requests the dual-stack to transmit a packet, determining whether to transmit an IPv4 packet or an IPv6 packet and whether to use tunnels is necessary. A possible algorithm to make these decisions is described in the RFC 1933[5] and reported in the following.

> This section presents a combined IPv4 and IPv6 sending algorithm that IPv6/IPv4 nodes can use. The algorithm can be used to determine when to send IPv4 packets, when to send IPv6 packets, and when to perform automatic and configured tunneling. It illustrates how the techniques of dual IP layer, configured tunneling, and automatic tunneling can be used together. Note this is just an example to show how the techniques can be combined; IPv6/IPv6 implementations may provide different algorithms. This algorithm has the following properties:
>
> - Sends IPv4 packets to all IPv4 destinations.
> - Sends IPv6 packets to all IPv6 destinations on the same link.
> - Using automatic tunneling, sends IPv6 packets encapsulated in IPv4 to IPv6 destinations with IPv4-compatible addresses that are located off-link.
> - Sends IPv6 packets to IPv6 destinations located off-link when IPv6 routers are present.
> - Using the default IPv6 tunnel, sends IPv6 packets encapsulated in IPv4 to IPv6 destinations with IPv6-only addresses when no IPv6 routers are present.
>
> The algorithm is as follows:
>
> 1 If the address of the end node is an IPv4 address then:
>
> 1.1 If the destination is located on an attached link, then send an IPv4 packet addressed to the end node.
>
> 1.2 If the destination is located off-link, then;
>
> 1.2.1 If there is an IPv4 router on link, then send an IPv4 format packet. The IPv4 destination address is the IPv4 address of the end node. The datalink address is the data-link address of the IPv4 router.

1.2.2 Else, the destination is treated as "unreachable" because it is located off link and there are no on-link routers.

2 If the address of the end node is an IPv4-compatible Pv6 address (i.e. bears the prefix 0:0:0:0:0:0), then:

2.1 If the destination is located on an attached link, then send an IPv6 format packet (not encapsulated). The IPv6 destination address is the IPv6 address of the end node. The datalink address is the datalink address of the end node.

2.2 If the destination is located off-link, then:

2.2.1 If there is an IPv4 router on an attached link, then send an IPv6 packet encapsulated in IPv4. The IPv6 destination address is the address of the end node. The IPv4 destination address is the low-order 32-bits of the end node's address. The datalink address is the datalink address of the IPv4 router.

2.2.2 Else, if there is an IPv6 router on an attached link, then send an IPv6 format packet. The IPv6 destination address is the IPv6 address of the end node. The datalink address is the datalink address of the IPv6 router.

2.2.3 Else, the destination is treated as "unreachable" because it is located off-link and there are no on-link routers.

3 If the address of the end node is an IPv6-only address, then:

3.1 If the destination is located on an attached link, then send an IPv6 format packet. The IPv6 destination address is the IPv6 address of the end node. The datalink address is the datalink address of the end node.

3.2 If the destination is located off-link, then:

3.2.1 If there is an IPv6 router on an attached link, then send an IPv6 format packet. The IPv6 destination address is the IPv6 address of the end node. The datalink address is the datalink address of the Ipv6 router.

3.2.2 Else, if the destination is reachable via a configured tunnel, and there is an IPv4 router on an attached link, then send an IPv6 packet encapsulated in IPv4. The Ipv6 destination address is the address of the end node. The IPv4 destination address is the configured IPv4 address of the tunnel endpoint. The datalink address is the data-link address of the IPv4 router.

3.2.3 Else, the destination is treated as "unreachable" because it is located off-link and there are no on-link IPv6 routers.

A summary of these sending rules are given in the table below:

| End Node Address Type | End Node On Link? | IPv4 Router On Link? | IPv6 Router On Link? | Packet Format To Send | IPv6 Dest Addr | IPv4 Dest Addr | DLink Dest Addr |
|---|---|---|---|---|---|---|---|
| IPv4 | Yes | N/A | N/A | IPv4 | N/A | E4 | EL |
| IPv4 | No | Yes | N/A | IPv4 | N/A | E4 | RL |
| IPv4 | No | No | N/A | UNRCH | N/A | N/A | N/A |
| IPv4-compat | Yes | N/A | N/A | IPv6 | E6 | N/A | EL |
| IPv4-compat | No | Yes | N/A | IPv6/4 | E6 | E4 | RL |
| IPv4-compat | No | No | Yes | IPv6 | E6 | N/A | RL |
| IPv4-compat | No | No | No | UNRCH | N/A | N/A | N/A |
| IPv6-only | Yes | N/A | N/A | IPv6 | E6 | N/A | EL |
| IPv6-only | No | N/A | Yes | IPv6 | E6 | N/A | RL |
| IPv6-only | No | Yes | No | IPv6/4 | E6 | T4 | RL |
| IPv6-only | No | No | No | UNRCH | N/A | N/A | N/A |

Key to Abbreviations

| | |
|---|---|
| N/A: | Not applicable or does not matter. |
| E6: | IPv6 address of end node. |
| E4: | IPv4 address of end node (low-order 32-bits of IPv4-compatible address). |
| EL: | Datalink address of end node. |
| T4: | IPv4 address of the tunnel endpoint. |
| R6: | IPv6 address of router. |
| R4: | IPv4 address of router. |
| RL: | Datalink address of router. |
| Ipv4: | IPv4 packet format. |
| Ipv6: | IPv6 packet format. |
| Ipv6/4: | IPv6 encapsulated in IPv4 packet format. |
| UNRCH: | Destination is unreachable. Don't send a packet. |

# References

[1]S.E. Deering, R. Hinden, *RFC 1883: Internet Protocol, Version 6 (IPv6) Specification,* December 1995.

[2]A. Conta, S. Deering, *RFC 1885: Internet Control Message Protocol (ICMPv6),* December 1995.

[3]S.E. Deering, *RFC 1112: Host extensions for IP multicasting,* August 1989.

[4]Y. Rekhter, T. Li, *RFC 1887: An Architecture for IPv6 Unicast Address Allocation,* December 1995.

[5]R. Gilligan, E. Nordmar, *RFC 1933: Transition Mechanisms for IPv6 Hosts and Routers,* April 1996.