IPv6 Packet Creation With Scapy Documentation

Release 1.0

Oliver Eggert

January 19, 2012

CONTENTS

1 Introduction **1**
- 1.1 Structure of this document . . . 1
- 1.2 Basic Scapy Usage . . . 1

2 The Standard IPv6 Header **3**

3 The IPv6 Extension Headers **4**
- 3.1 The Hop-By-Hop Extension Header . . . 4
- 3.2 The Destination Extension Header . . . 5
- 3.3 The Routing Extension Header . . . 5
- 3.4 The Fragment Extension Header . . . 6
- 3.5 Variable Options . . . 6
- 3.6 Examples . . . 7

4 The ICMPv6 Headers **10**
- 4.1 Destination Unreachable . . . 10
- 4.2 Packet too Big . . . 11
- 4.3 Time Exceeded . . . 11
- 4.4 Parameter Problem . . . 11
- 4.5 Echo Request and Echo Reply (Ping-Pong) . . . 12
- 4.6 Examples . . . 12

5 Neighbor Discovery **14**
- 5.1 Router Solicitation . . . 14
- 5.2 Router Advertisement . . . 14
- 5.3 Neighbor Solicitation . . . 15
- 5.4 Neighbor Advertisement . . . 16
- 5.5 Redirect . . . 16
- 5.6 Examples . . . 17

6 Multicast Router Discovery **19**
- 6.1 Multicast Router Solicitation . . . 19
- 6.2 Multicast Router Advertisement . . . 19
- 6.3 Multicast Router Termination . . . 20
- 6.4 Examples . . . 20

7 Multicast Listener Discovery **21**
- 7.1 Examples . . . 21

CHAPTER

ONE

INTRODUCTION

Welcome to “IPv6 Packet Creation With Scapy”. This guide provides a list of some of the most often used IPv6-related header types and how to build them with scapy.

We assume the reader has some familarity with IPv6 and related protocols, such as ICMPv6 or Neighbor Discovery. We won’t explain the meaning of *every* header option.

The guide is being written as part of the project IPv6 Intrusion Detection System, funded by BMBF.

1.1 Structure of this document

IPv6 is a very complex thing made up of various components. To give this guide some structure, we’ll first explain the very core of IP, that is the basic IPv6-Header, the Extension-Headers and the ICMPv6-Headers. We then examine the rest of the IPv6 protocol suite *feature-by-feature*, rather than *stack-by-stack* or *protocol-by-protocol* (for example, Neighbor Discovery is technically speaking part ICMPv6 as well). We hope this keeps the guide more focused.

For each feature, we provide a link to the RFC in which that feature is specified in and show the header structures (the “ascii-art” tables from that RFC). We also show how scapy maps the header fields to class members.

Some features of IPv6 are not within the scope of the project and are therefore not included in this document. This includes *DHCPv6* and *MoblieIP*

1.2 Basic Scapy Usage

Scapy is a tool written in python that allows you to easily create, manipulate, send and receive network packets. We assume the reader has a basic understanding of what scapy is and how to use it, so we won’t go into too much detail. For a more thorough introduction consider reading the official documentation.

However, we’d like to outline some of the more important features:

- Different protocols and header types are represented by different classes, such as `IPv6` or `ICMP`.
- You can “stack” protocols on top of each other, just like a real network stack would, by using the Slash-Operator (`/`): `IPv6()/TCP()` creates a TCP-over-IPv6-Packet.
- Packets can be viewed with the `show()` and `show2()` functions. They can be sent with the `send()`, `sendp()`, `sr()`, `sr1()`, `srp()` functions.
- Most, if not all, header options can be manipulated. They are class members and can be accessed either in the constructor or via normal member access, like this:

```
x=IP(ttl=64)
x.src="127.0.0.1"
```

- Scapy can fill header options. You don't need to provide *all* the information.
- To see what options are available, you *could* use the `ls()` function or **look them up in this guide**.

CHAPTER

TWO

THE STANDARD IPV6 HEADER

The IPv6 Header is defined in RFC2460 and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

In scapy IPv6 packets are represented by the `IPv6`-class. Scapy maps the header fields to the following class members:

```
>>> ls(IPv6)
version    : BitField             = (6)
tc         : BitField             = (0)
fl         : BitField             = (0)
plen       : ShortField           = (None)
nh         : ByteEnumField        = (59)
hlim       : ByteField            = (64)
src        : SourceIP6Field       = (None)
dst        : IP6Field             = ('::1')
```

Note the default values in the brackets. For example, the “version”-field should have the value “6” for IPv6. Empty fields will be filled by scapy when passed to one of the *send()* functions. As stated in the previous chapter, you can manipulate the options in the constructor or via normal member access.:

```
x = IPv6(src='fe80::0123:4567')
x.hlim=255
```

CHAPTER THREE

THE IPV6 EXTENSION HEADERS

Compared with IPv4, the IPv6 header has duplicated in size. To not increase the size further, the basic header holds only those fields that are absolutley neccessary. Optional data has to be provided in extension headers.

RFC2460 defines 4 Extension Headers: the *Hop-By-Hop*, *Destination*, *Routing* and *Fragment* header.

Each of these headers (and the basic IPv6 header as well) has a field *next header*. It contains a number that specifies which header will follow. These values are:

- Hop-By-Hop: 0
- Routing: 43
- Fragment: 44
- Destination: 60

If the next header is part of another protocol (i.e. TCP or UDP) then you have to use their protocol numbers (6 or 17, respectively). For a list of all protocol numbers, see here.

Should no next header follow, i.e. an IPv6 packet with no payload, then the value of *next header* should be 59.

3.1 The Hop-By-Hop Extension Header

The Hop-By-Hop header is represented by the `IPv6ExtHdrHopByHop`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The class members are:

```
>>> ls(IPv6ExtHdrHopByHop)
nh         : ByteEnumField        = (59)
len        : FieldLenField        = (None)
autopad    : _PhantomAutoPadField = (1)
options    : _HopByHopOptionsField = ([])
```

See section *Variable Options* below for a list of possible options.

3.2 The Destination Extension Header

The Destination Header is represented by the class :py:class'IPv6ExtHdrDestOpt'-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The class members are:

```
>>> ls(IPv6ExtHdrDestOpt)
nh         : ByteEnumField        = (59)
len        : FieldLenField        = (None)
autopad    : _PhantomAutoPadField = (1)
options    : _HopByHopOptionsField = ([])
```

See section *Variable Options* below for a list of possible options.

3.3 The Routing Extension Header

The Routing header is represented by the `IPv6ExtHdrRouting`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |  Routing Type | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                       type-specific data                      .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

There is currently only the Routing Header Type 0. It is deprecated and should not be used (see RFC5095). The type-specific data for RH Type 0 looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Reserved                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                           Address[1]                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                           Address[2]                          +
|                                                               |
```

```
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                               .                               .
.                               .                               .
.                               .                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                           Address[n]                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Although being deprecated, you can still build the RH Type 0 Header with scapy. The class members are:

```
>>> ls(IPv6ExtHdrRouting)
nh         : ByteEnumField        = (59)
len        : FieldLenField        = (None)
type       : ByteField            = (0)
segleft    : ByteField            = (None)
reserved   : BitField             = (0)
addresses  : IP6ListField         = ([])
```

3.4 The Fragment Extension Header

The Fragment header is represented by the `IPv6ExtHdrFragment`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |   Reserved    |      Fragment Offset    |Res|M|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Identification                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The class members are:

```
>>> ls(IPv6ExtHdrFragment)
nh         : ByteEnumField        = (59)
res1       : BitField             = (0)
offset     : BitField             = (0)
res2       : BitField             = (0)
m          : BitField             = (0)
id         : IntField             = (None)
```

Note: the fields *res1* and *res2* are reserved and currently unused. The *offset* is an unsigned integer that counts the offset in steps of 8 Bytes. $m = 0$ means that this is the last fragment, $m = 1$ means that more fragments will follow. The *id* identiefies the original packet that was split up into multiple fragments.

3.5 Variable Options

The "options"-field within the Hop-By-Hop and Destination-Header is used to carry a variable number of options. When RFC 2460 was originally specified, those were not explicitly defined. It was merely stated that both headers will be needed, but not what options they should provide. Over the years some suggestions have been made and you can now find a list of officially specified options here.

All Option Headers have the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
|  Option Type   |  Opt Data Len |  Option Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
```

The 8-bit *Option Type* specifies what option this is, the 8-bit *Option Data Lenght* specifies the lenght of the option, followed by the *Option Data*. Since it is possible to have more than one option in a single Hop-By-Hop or Destination Header you might need to pad your option, so that each new option aligns naturally and that the whole Header has a length that is a multiple of 8 octets. For that, you may use the *Pad1* and *PadN* option. See RFC2460. Scapy can configure the padding automatically.

Scapy currently supports the following:

```
>>> ls(Pad1)
otype      : _OTypeField         = (0)

>>> ls(PadN)
otype      : _OTypeField         = (1)
optlen     : FieldLenField       = (None)
optdata    : StrLenField         = ('')

>>> ls(RouterAlert)
otype      : _OTypeField         = (5)
optlen     : ByteField           = (2)
value      : ShortEnumField      = (None)

>>> ls(Jumbo)
otype      : _OTypeField         = (194)
optlen     : ByteField           = (4)
jumboplen  : IntField            = (None)

>>> ls(HAO)
otype      : _OTypeField         = (201)
optlen     : ByteField           = (16)
hoa        : IP6Field            = ('::')
```

Explaining all options of those headers is not within the scope of this guide. The official list provides links to the relevant RFCs.

There are some options that scapy does not have separate classes for. These are: *Tunnel Encapsulation Limit*, *Quick-Start*, *CALIPSO*, *Endpoint Identification* and *RPL-Option*. You would have to build the hex-string and pass it to *IPv6ExtHdrHopByHop.options* yourself should you want to use those features.

3.6 Examples

In this example we send an IPv6-Jumbogram with a spoofed source address. This will show you:

- how to create and modify an IPv6-Packet,
- how to add an Extension Header,
- how the variable extension header options work.

As this is the first example in this guide it will be a bit more detailed than the others.

3.6.1 Step 1:

We create and modify the IPv6-Packet. We need to specify the destination address and, as we don't want scapy to automatically use our real address, the spoofed source address:

```
base = IPv6()
base.dst = 'fe80::1234'
base.src = 'fe80::dead:beef'
```

We let scapy figure out all the other settings.

3.6.2 Step 2:

We create an Extension Header. The *Jumbogram*-Option needs to go into the Hop-By-Hop Header:

```
extension = IPv6ExtHdrHopByHop()
jumbo = Jumbo()
```

Now let's have a look at the Jumbogram. We have a maximum of 32 bit to specify the payload length of the jumbogram. In this example we simply choose a big number like, $2^{30}$and paste the jumbo-option into the hop-by-hop-header:

```
jumbo.jumboplen = 2**30
extension.options = jumbo
```

3.6.3 Step 3:

Stack the headers, inspect the result and pass them to the send function:

```
packet = base/extension
packet.show2()
###[ IPv6 ]###
  version= 6L
  tc= 0L
  fl= 0L
  plen= 8
  nh= Hop-by-Hop Option Header
  hlim= 64
  src= fe80::dead:beef
  dst= fe80::1234
###[ IPv6 Extension Header - Hop-by-Hop Options Header ]###
     nh= No Next Header
     len= 0
     autopad= On
     \options\
      |###[ Jumbo Payload ]###
      |  otype= Jumbo Payload [11: discard+ICMP not mcast, 0: Don't change en-route]
      |  optlen= 4
      |  jumboplen= 1073741824

send(packet)
```

Done!

Hint: All of this *can* be done in one line:

```
(IPv6(dst='fe80::1234',src='fe80::dead:beef')/IPv6ExtHdrHopByHop(options=Jumbo(jumboplen=2**30)))
```

CHAPTER

FOUR

THE ICMPV6 HEADERS

The *Internet Control Message Protocol for the Internet Protocol Version 6 (ICMPv6)* provides additional features for IPv6. It is defined in RFC2463.

ICMPv6 defines separate headers for four *error messages* (Destination Unreachable, Packet Too Big, Time Exceeded and Parameter Problem) and two *informational messages* (Echo Request and Echo Reply).

As is the case with the IPv6-Extension headers, each ICMPv6 Header has a type that identifies it. The default values for the various types can be seen in the following sections, as scapy gives the default value of the *type*-field. Additional Features, such as Autoconfiguration and Neighbor Discovery, are also done via ICMPv6, but discussed in separate chapters.

4.1 Destination Unreachable

The Destination Unreachable header is represented by the `ICMPv6DestUnreach`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    As much of invoking packet                 |
+                as will fit without the ICMPv6 packet          +
|                exceeding the minimum IPv6 MTU [IPv6]          |
```

The class members are:

```
>>> ls(ICMPv6DestUnreach)
type       : ByteEnumField        = (1)
code       : ByteEnumField        = (0)
cksum      : XShortField          = (None)
unused     : XIntField            = (0)
```

RFC2463 defines the following the following reasons for being unable to deliver a packet. Enter these into the *code*-Field:

```
0 - no route to destination
1 - communication with destination administratively prohibited
2 - (not assigned)
3 - address unreachable
4 - port unreachable
```

4.2 Packet too Big

The Packet Too Big header is represented by the `ICMPv6PacketTooBig`- class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             MTU                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    As much of invoking packet                 |
+               as will fit without the ICMPv6 packet           +
|               exceeding the minimum IPv6 MTU [IPv6]           |
```

The class members are:

```
>>> ls(ICMPv6PacketTooBig)
type       : ByteEnumField        = (2)
code       : ByteField            = (0)
cksum      : XShortField          = (None)
mtu        : IntField             = (1280)
```

According to the RFC, the *code*-field will be ignored by the receiver.

4.3 Time Exceeded

The Time Exceeded header is represented by the `ICMPv6TimeExceeded`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Unused                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    As much of invoking packet                 |
+               as will fit without the ICMPv6 packet           +
|               exceeding the minimum IPv6 MTU [IPv6]           |
```

The class members are:

```
>>> ls(ICMPv6TimeExceeded)
type       : ByteEnumField        = (3)
code       : ByteField            = ({0: 'hop limit exceeded in transit',
                                      1: 'fragment reassembly time exceeded'})
cksum      : XShortField          = (None)
unused     : XIntField            = (0)
```

In this class you can already see the possible values for the *code*-field and their meaning.

4.4 Parameter Problem

The Parameter Problem header is represented by the `ICMPv6ParamProblem`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Pointer                            |
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    As much of invoking packet                 |
+                as will fit without the ICMPv6 packet          +
|                exceeding the minimum IPv6 MTU [IPv6]          |
```

The class members are:

```
>>> ls(ICMPv6ParamProblem)
type       : ByteEnumField        = (4)
code       : ByteEnumField        = (0)
cksum      : XShortField          = (None)
ptr        : IntField             = (6)
```

The possible values for the *code* field are:

```
0 - erroneous header field encountered
1 - unrecognized Next Header type encountered
2 - unrecognized IPv6 option encountered
```

The *pointer* field will point to the offset in the packet where the error occured (if it fits into this reply)

4.5 Echo Request and Echo Reply (Ping-Pong)

The Echo Request header is represented by the `ICMPv6EchoRequest`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Data ...
+-+-+-+-+-
```

The class members are:

```
>>> ls(ICMPv6EchoRequest)
type       : ByteEnumField        = (128)
code       : ByteField            = (0)
cksum      : XShortField          = (None)
id         : XShortField          = (0)
seq        : XShortField          = (0)
data       : StrField             = ('')
```

The Echo Reply header looks just the same. It is represented by `ICMPv6EchoReply`. They only differ in the *type*-field: The **EchoRequest** has Type 128, the **EchoReply** has Type 129.

4.6 Examples

In this example we'll build a ping (Echo Request) with a custom payload:

```
payload = "foo.bar " * 50

base=IPv6(dst='fe80::1234')
extension=ICMPv6EchoRequest(data=payload)

packet = base/extension
```

```
packet.show()

###[ IPv6 ]###
  version= 6
  tc= 0
  fl= 0
  plen= None
  nh= ICMPv6
  hlim= 64
  src= ::1
  dst= fe80::1234
###[ ICMPv6 Echo Request ]###
     type= Echo Request
     code= 0
     cksum= None
     id= 0x0
     seq= 0x0
     data= 'foo.bar foo.bar foo.bar [...] foo.bar foo.bar '

send(packet)
```

Note: We inserted the brackets to not have to show 50 *foo.bar*s. But they all show up in scapy's output.

CHAPTER

FIVE

NEIGHBOR DISCOVERY

As a replacement for ARP, the *Neighbor Discovery Protocol (NDP)* was introduced in IPv6. With NDP, machines on one link can find out about other machines, local routers, and determine link-layer addresses. It is defined in RFC 4861.

NDP messages are modularly built and consist of five *core* headers, explained below, and five *nd_opts* headers.

5.1 Router Solicitation

The Router Solicitation header is represented by the `ICMPv6ND_RS`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Reserved                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

The class members are:

```
>>> ls(ICMPv6ND_RS)
type       : ByteEnumField        = (133)
code       : ByteField            = (0)
cksum      : XShortField          = (None)
res        : IntField             = (0)
```

Available options for this header are:

- *src_ll_addr*

5.2 Router Advertisement

The Router Advertisment header is represented by the `ICMPv6ND_RA`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Cur Hop Limit |M|O|  Reserved |       Router Lifetime         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Reachable Time                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|                          Retrans Timer                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

The class members are:

```
>>> ls(ICMPv6ND_RA)
type        : ByteEnumField        = (134)
code        : ByteField            = (0)
cksum       : XShortField          = (None)
chlim       : ByteField            = (0)
M           : BitField             = (0)
O           : BitField             = (0)
H           : BitField             = (0)
prf         : BitEnumField         = (1)
P           : BitField             = (0)
res         : BitField             = (0)
routerlifetime : ShortField           = (1800)
reachabletime : IntField             = (0)
retranstimer : IntField             = (0)
```

The meaning of the fields *H*, *prf* and *P* are currently unknown.

Available options for this header are:

- *src_ll_addr*
- *mtu*
- *prefix_info*.

5.3 Neighbor Solicitation

The Neighbor Solicitation header is representend by the `ICMPv6ND_NS`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Reserved                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Target Address                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

The class members are:

```
>>> ls(ICMPv6ND_NS)
type        : ByteEnumField        = (135)
code        : ByteField            = (0)
cksum       : XShortField          = (None)
R           : BitField             = (0)
```

```
S          : BitField              = (0)
O          : BitField              = (0)
res        : XBitField             = (0)
tgt        : IP6Field              = ('::')
```

The meaning of the fields *R*, *S* and *O* are currently unknown.

Available options for this header are:

- *src_ll_addr*

5.4 Neighbor Advertisement

The Neighbor Advertisement header is represented by the `ICMPv6ND_NA`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|S|O|                     Reserved                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Target Address                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

The class members are:

```
>>> ls(ICMPv6ND_NA)
type       : ByteEnumField         = (136)
code       : ByteField             = (0)
cksum      : XShortField           = (None)
R          : BitField              = (1)
S          : BitField              = (0)
O          : BitField              = (1)
res        : XBitField             = (0)
tgt        : IP6Field              = ('::')
```

Available options for this header are:

- *target_ll_addr*

5.5 Redirect

The Redirect header is represented by the `ICMPv6ND_Redirect`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Reserved                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|                                                               |
+                                                               +
|                                                               |
+                       Target Address                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                     Destination Address                       +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

The class members are:

```
>>> ls(ICMPv6ND_Redirect)
type       : ByteEnumField         = (137)
code       : ByteField             = (0)
cksum      : XShortField           = (None)
res        : XIntField             = (0)
tgt        : IP6Field              = ('::')
dst        : IP6Field              = ('::')
```

The available options for this header are:

- *target_ll_addr*
- *redir_hdr*

5.6 Examples

Let's build a router solicitation with a source link-layer address option:

```
base=IPv6(dst='fe80::1234')
router_solicitation=ICMPv6ND_RS()
src_ll_addr=ICMPv6NDOptSrcLLAddr(lladdr='01:23:45:67:89:ab')

packet=base/router_solicitation/src_ll_addr
packet.show()

###[ IPv6 ]###
  version= 6
  tc= 0
  fl= 0
  plen= None
  nh= ICMPv6
  hlim= 255
  src= ::1
  dst= fe80::1234
###[ ICMPv6 Neighbor Discovery - Router Solicitation ]###
     type= Router Solicitation
```

```
     code= 0
     cksum= None
     res= 0
###[ ICMPv6 Neighbor Discovery Option - Source Link-Layer Address ]###
        type= 1
        len= 1
        lladdr= 01:23:45:67:89:ab

send(packet)
```

CHAPTER

SIX

MULTICAST ROUTER DISCOVERY

The *Multicast Router Discovery* can be used to determine which router has multicast support enabled. It is specified in RFC4286.

It introduces three new ICMPv6 Headers:

6.1 Multicast Router Solicitation

The Multicast Router Solicitation header is represented by the `ICMPv6MRD_Solicitation`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |   Reserved    |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The class members are:

```
>>> ls(ICMPv6MRD_Solicitation)
    type       : ByteEnumField         = (152)
    res        : ByteField             = (0)
    cksum      : XShortField           = (None)
```

6.2 Multicast Router Advertisement

The Multicast Router Advertisment header is represented by the `ICMPv6MRD_Advertisement`-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     |  Ad. Interval |            Checksum           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Query Interval        |      Robustness Variable      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The class members are:

```
>>> ls(ICMPv6MRD_Advertisement)
type       : ByteEnumField         = (151)
advinter   : ByteField             = (20)
cksum      : XShortField           = (None)
queryint   : ShortField            = (0)
robustness : ShortField            = (0)
```

6.3 Multicast Router Termination

The Multicast Router Termination header is represented by the ICMPv6MRD_Termination-class and looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |   Reserved    |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The class members are:

```
>>> ls(ICMPv6MRD_Termination)
    type       : ByteEnumField        = (153)
    res        : ByteField            = (0)
    cksum      : XShortField          = (None)
```

6.4 Examples

We create a Multicast Router Advertisement with a custom advertisement interval. Let's do this as a one-liner:

```
>>> (IPv6(dst='fe80::1234')/ICMPv6MRD_Advertisement(advinter=40)).show2()
###[ IPv6 ]###
  version= 6L
  tc= 0L
  fl= 0L
  plen= 8
  nh= ICMPv6
  hlim= 1
  src= ::1
  dst= fe80::1234
###[ ICMPv6 Multicast Router Discovery Advertisement ]###
     type= Multicast Router Advertisement
     advinter= 40
     cksum= 0x57df
     queryint= 0
     robustness= 0
```

CHAPTER

SEVEN

MULTICAST LISTENER DISCOVERY

With *Multicast Listener Discovery*, multicast routers can find out about local nodes that want to receive and send multicast packets. It is specified in RFC2710.

When sending MLD-Packets make sure you include a IPv6 Router Alert (see the example). Multicast Listener Discovery adds three new ICMPv6 Headers, which all have the same structure:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Maximum Response Delay    |          Reserved             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Multicast Address                       +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

They only differ in *type*:

- type = 130 means it's a *Multicast Listener Query* message,
- type = 131 means it's a *Multicast Listener Report* message,
- type = 132 means it's a *Multicast Listener Done* message.

They are represented by the `ICMPv6MLQuery`-, `ICMPv6MLDone`- and `ICMPv6MLReport`-class, respectively. All three classes have the same members:

```
>>> ls(ICMPv6MLReport)
type       : ByteEnumField        = (131)
code       : ByteField            = (0)
cksum      : XShortField          = (None)
mrd        : ShortField           = (0)
reserved   : ShortField           = (0)
mladdr     : IP6Field             = (None)
```

7.1 Examples

To send a Multicast Listener Discovery packet you need to do some extra configuration on the base header. RFC2710 states that:

> "All MLD messages described in this document are sent with a link-local IPv6 Source Address, an IPv6 Hop Limit of 1, and an IPv6 Router Alert option [RTR-ALERT] in a Hop-by-Hop Options header."

So let's build that:

```
base = IPv6(src='fe80::dead:beef', dst='fe80::1234', hlim=1)
hbh = IPv6ExtHdrHopByHop(options = RouterAlert())
mlq = ICMPv6MLQuery()

packet=base/hbh/mlq
packet.show()

###[ IPv6 ]###
  version= 6
  tc= 0
  fl= 0
  plen= None
  nh= Hop-by-Hop Option Header
  hlim= 1
  src= fe80::dead:beef
  dst= fe80::1234
###[ IPv6 Extension Header - Hop-by-Hop Options Header ]###
     nh= No Next Header
     len= None
     autopad= On
     \options\
      |###[ Router Alert ]###
      |  otype= Router Alert [00: skip, 0: Don't change en-route]
      |  optlen= 2
      |  value= None
###[ MLD - Multicast Listener Query ]###
        type= MLD Query
        code= 0
        cksum= None
        mrd= 10000
        reserved= 0
        mladdr= ::
```