IPv4 Extension Headers and UDP Encapsulated Extension Headers
draft-herbert-ipv4-udpencap-eh-00

Abstract

This specification defines extension headers for IPv4 and a method to encapsulate extension headers in UDP to facilitate transmission over the Internet. The goal is to provide a uniform and feasible method of extensibility that is shared between IPv4 and IPv6.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1 Introduction .................................................. 3
  1.1 IPv4 extension headers .................................. 3
  1.2 Encapsulating extension headers in UDP ................. 3
2 IPv4 extension headers ........................................ 4
  2.1 Extension Header Order .................................... 6
  2.2 Options .................................................... 7
  2.3 Hop-by-Hop Options Header ................................. 9
  2.4 Routing Header ............................................. 10
  2.5 Fragment Header ........................................... 11
  2.6 Destination Options Header ................................. 18
  2.7 No Next Header .............................................. 18
  2.8 Interaction with standard IPv4 mechanisms ................. 19
    2.8.1 IPv4 options and IPv4 extension headers ............... 19
    2.8.2 IPv4 fragmentation and IPv4 extension headers ......... 19
3 Encapsulating extension headers in UDP ...................... 20
  3.1 Encapsulation format ....................................... 20
  3.2 GUE magic numbers ......................................... 21
  3.3 Operation .................................................. 22
    3.3.1 Sender processing ..................................... 22
    3.3.2 Destination Processing ................................. 23
    3.3.3 Intermediate device processing ......................... 23
4 Security Considerations ...................................... 25
5 IANA Considerations ......................................... 25
6 References .................................................. 25
  6.1 Normative References ...................................... 25
  6.2 Informative References .................................... 25
Author’s Address ................................................. 27
1 Introduction

This specification defines extension headers for IPv4 and a method to encapsulate extension headers in UDP to facilitate transmission over the Internet.

1.1 IPv4 extension headers

IPv4 options were defined in [RFC791] as the means of extending the IP protocol. IPv4 options have not been successful. Early router implementations, and even those today, either don’t process IPv4 options or relegate them to a slow path effectively making them unusable for serious applications. IPv4 options are limited to forty bytes length and, unlike TCP options, no IP options have been defined that are critical to communications. The upshot is that IPv4 options have long not been considered an option for deployment [IPNOPT].

IPv6 took a different approach. Extensibility of IPv6 is provided by extension headers. Optional internet-layer information is encoded in separate headers that may be placed between the IPv6 header and the upper-layer header in a packet [RFC8200]. IPv6 extension headers have had mixed success in deployment in that some intermediate devices have trouble processing them [RFC7872], however there are several active proposals in IETF that would make use of them (e.g. [FAST], [MTUOPT], [IOAM], [SRV6EH]).

This specification proposes that extension headers, those defined for IPv6, should be usable with IPv4 as a common method of extensibility. Using extension headers with IPv4 is logically straightforward. The IPv4 Protocol field is effectively re-designated to be a Next Header field with the same meaning and semantics as the IPv6 Next Header field. In this manner, an IPv4 packet can contain any defined IPv6 extension headers that are recast as IPv4 extension headers. These include Hop-by-Hop Options, Routing Header, Fragment, Destination Options, Authentication, and Encapsulating Security Payload. In cases where an extension header contains IPv6 specific information, the extension header can be adapted for use with IPv4. For instance, a Routing Header carrying IPv6 addresses to visit could be adapted to carry IPv4 addresses.

1.2 Encapsulating extension headers in UDP

Deep Packet Inspection (DPI) is a common technique of middleboxes that has ossified Internet protocols in several ways. Attempts to use extension headers with IPv4 would likely be problematic for intermediate devices doing DPI. To address this, extension headers can be encapsulated in UDP using Generic UDP Encapsulation. The idea is to insert a shim GUE/UDP header between an IPv4 (or IPv6) header...
and the extension headers. To nodes that don’t understand extension headers, encapsulated extension headers are transparent and packets appear to be simple UDP/IP packets. To nodes that understand extension headers and the encapsulation, the GUE/UDP header is treated as an extension header itself that appears before any other extension headers.

Hop-by-Hop options are intended to be parsed, processed, and possibly modified by intermediate nodes in a path. When Hop-by-Hop options are encapsulated in UDP, consideration needs to be given on how to ensure robustness. Per [RFC7605], UDP port numbers only have meaning at the transport endpoints, so if an intermediate node attempts to interpret a UDP payload based solely on port number it may be incorrect. If a node were to modify a UDP payload whose type it has misinterpreted, then systematic silent data corruption ensues. To mitigate this issue, a magic number can be set in the UDP data that indicates the payload type. A magic number identifies the payload as being GUE with high probability to minimize the risk of misinterpretation.

Note that the solution to encapsulate extension headers can be used for both IPv4 and IPv6. Encapsulation serves as workaround for paths that have problems processing IPv6 extension headers.

2 IPv4 extension headers

IPv4 extension headers are optional internet-layer information encoded in separate headers that may be placed between the IPv4 header and the upper-layer header in a packet. IPv4 extension headers are based on IPv6 extension headers and share the same basic properties and semantics [RFC8200].

Extension headers are numbered from IANA IP Protocol Numbers [IANA-PN], the same values are used for IPv4 and IPv6. When processing a sequence of Next Header values in a packet, the first one that is not an extension header [IANA-EH] indicates that the next item in the packet is the corresponding upper-layer header. A special "No Next Header" value is used if there is no upper-layer header.

As illustrated in these examples, an IPv4 packet MAY carry zero, one, or more extension headers, each identified by the Next Header field of the preceding header or the Protocol field of the IPv4 header:

```
+---------------+------------------------
|  IPv4 header  | TCP header + data      |
| Protocol = TCP |
+---------------+------------------------
```
Extension headers (except for the Hop-by-Hop Options header) MUST NOT be processed, inserted, or deleted by any node along a packet’s delivery path, until the packet reaches the node (or each of the set of nodes, in the case of multicast) identified in the Destination Address field of the IPv4 header.

The Hop-by-Hop Options header MUST NOT be inserted or deleted, but MAY be examined or processed by any node along a packet’s delivery path, until the packet reaches the node (or each of the set of nodes, in the case of multicast) identified in the Destination Address field of the IPv4 header. The Hop-by-Hop Options header, when present, MUST immediately follow the IPv4 header. Its presence is indicated by the value zero in the Protocol field of the IPv4 header.

At the destination node, normal demultiplexing on the Protocol field of the IPv4 header invokes the module to process the first extension header, or the upper-layer header if no extension header is present. The contents and semantics of each extension header determine whether or not to proceed to the next header. Therefore, extension headers MUST be processed strictly in the order they appear in the packet; a receiver MUST NOT, for example, scan through a packet looking for a particular kind of extension header and process that header prior to processing all preceding ones.

If, as a result of processing a header, the destination node is required to proceed to the next header but the Next Header value in the current header is unrecognized by the node, it SHOULD discard the packet and send an ICMP Parameter Problem message to the source of the packet, with an ICMP Code value of 1 ("unrecognized Next Header type encountered") and the ICMP Pointer field containing the offset of the unrecognized value within the original packet. The same action should be taken if a node encounters a Next Header value of zero in any header other than an IPv6 or IPv4 header.
Each extension header is an integer multiple of 8 octets long, in order to retain 8-octet alignment for subsequent headers. Multi-octet fields within each extension header are aligned on their natural boundaries, i.e., fields of width \( n \) octets are placed at an integer multiple of \( n \) octets from the start of the header, for \( n = 1, 2, 4, \) or 8.

A full implementation of IPv4 with extension headers includes implementation of the following extension headers:

- Hop-by-Hop Options
- Fragment
- Destination Options
- Routing
- Authentication
- Encapsulating Security Payload

The first four are specified in this document; IPv4 support for the last two are specified in [RFC4302] and [RFC4303], respectively. The current list of IPv4 extension headers is assumed to be the same as the list of IPv6 extension headers which can be found at [IANA-EH].

### 2.1 Extension Header Order

When more than one extension header is used in the same packet, it is RECOMMENDED that those headers appear in the following order:

1. IPv4 header
2. Hop-by-Hop Options header
3. Destination Options header (note 1)
4. Routing header
5. Fragment header
6. Authentication header (note 2)
7. Encapsulating Security Payload header (note 2)
8. Destination Options header (note 3)
9. Upper-Layer header

Note 1: for options to be processed by the first destination that appears in the IPv4 Destination Address field plus subsequent destinations listed in the Routing header.

Note 2: additional recommendations regarding the relative order of the Authentication and Encapsulating Security Payload headers are given in [RFC4303].

Note 3: for options to be processed only by the final destination of the packet.
Each extension header should occur at most once, except for the Destination Options header, which should occur at most twice (once before a Routing header and once before the upper-layer header).

If a legacy IPv4 destination node, one that does not support IPv4 extension headers, receives a packet with extension headers then the packet will be processed as having an unknown protocol. The packet will be discarded and an ICMP error should be generated.

IPv4 nodes that support extension headers MUST accept and attempt to process extension headers in any order and occurring any number of times in the same packet, except for the Hop-by-Hop Options header, which is restricted to appear immediately after an IPv4 header only. Nonetheless, it is strongly RECOMMENDED that sources of IPv4 packets adhere to the above recommended order until and unless subsequent specifications revise that recommendation.

### 2.2 Options

Two of the currently defined extension headers specified in this document -- the Hop-by-Hop Options header and the Destination Options header -- carry a variable number of "options" that are type-length-value (TLV). The format is:

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

- **Option Type**: 8-bit identifier of the type of option.
- **Opt Data Len**: 8-bit unsigned integer. Length of the Option Data field of this option, in octets.
- **Option Data**: Variable-length field. Option-Type-specific data.

The sequence of options within a header MUST be processed strictly in the order they appear in the header; a receiver MUST NOT scan through the header looking for a particular kind of option and process that option prior to processing all preceding ones.

The Option Type identifiers are internally encoded such that their highest-order 2 bits specify the action that MUST be taken if the processing IPv4 node does not recognize the Option Type:

- 00 - skip over this option and continue processing the header.
- 01 - discard the packet.
10 - discard the packet and, regardless of whether or not the packet’s Destination Address was a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet’s Source Address, pointing to the unrecognized Option Type.

11 - discard the packet and, only if the packet’s Destination Address was not a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet’s Source Address, pointing to the unrecognized Option Type.

The third-highest-order bit of the Option Type specifies whether or not the Option Data of that option can change en route to the packet’s final destination. When an Authentication header is present in the packet, for any option whose data may change en route, its entire Option Data field MUST be treated as zero-valued octets when computing or verifying the packet’s authenticating value.

0 - Option Data does not change en route

1 - Option Data may change en route

The three high-order bits described above are to be treated as part of the Option Type, not independent of the Option Type. That is, a particular option is identified by a full 8-bit Option Type, not just the low-order 5 bits of an Option Type.

The same Option Type numbering space is used for both the Hop-by-Hop Options header and the Destination Options header. However, the specification of a particular option may restrict its use to only one of those two headers.

Individual options may have specific alignment requirements, to ensure that multi-octet values within Option Data fields fall on natural boundaries. The alignment requirement of an option is specified using the notation xn+y, meaning the Option Type must appear at an integer multiple of x octets from the start of the header, plus y octets. For example:

\[2n\] means any 2-octet offset from the start of the header.

\[8n+2\] means any 8-octet offset from the start of the header, plus 2 octets.

There are two padding options that are used when necessary to align subsequent options and to pad out the containing header to a multiple of 8 octets in length. These padding options MUST be recognized by all IPv4 implementations that support extension headers:
Pad1 option (alignment requirement: none)

```
+-----------+
| 0         |
+-----------+
```

NOTE! the format of the Pad1 option is a special case -- it does not have length and value fields.

The Pad1 option is used to insert 1 octet of padding into the Options area of a header. If more than one octet of padding is required, the PadN option, described next, SHOULD be used, rather than multiple Pad1 options.

PadN option (alignment requirement: none)

```
+---------------------------------------------------+
| 1 | Opt Data Len | Option Data |
+---------------------------------------------------+
```

The PadN option is used to insert two or more octets of padding into the Options area of a header. For N octets of padding, the Opt Data Len field contains the value N-2, and the Option Data consists of N-2 zero-valued octets.

2.3 Hop-by-Hop Options Header

The Hop-by-Hop Options header is used to carry optional information that MAY be examined and processed by any node along a packet’s delivery path. The Hop-by-Hop Options header is identified by a Next Header value of 0 in the IPv4 header and has the following format:

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

Next Header 8-bit selector. Identifies the type of header immediately following the Hop-by-Hop Options header. Values are defined in [IANA-PN].

Hdr Ext Len 8-bit unsigned integer. Length of the Hop-by-Hop Options header in 8-octet units, not
including the first 8 octets.

Options Variable-length field, of length such that the complete Hop-by-Hop Options header is an integer multiple of 8 octets long. Contains one or more TLV-encoded options, as described in Section 2.2.

The only hop-by-hop options defined in this document are the Pad1 and PadN options specified in Section 2.2.

2.4 Routing Header

The Routing header is used by an IPv4 source to list one or more intermediate nodes to be "visited" on the way to a packet’s destination. This function is very similar to IPv4’s Loose Source and Record Route option. The Routing header is identified by a Next Header value of 43 in the immediately preceding header and has the following format:

```
+---------------------------------------------+
|    Next Header    |   Hdr Ext Len   |  Routing Type   | Segments Left |
+---------------------------------------------+
                                  |               |                 |
                                  |               |                 |
                                  |               |                 |
                                  |               |                 |
                                  |               |                 |
                                  |               |                 |
```

Next Header 8-bit selector. Identifies the type of header immediately following the Routing header. Values are defined in [IANA-PN].

Hdr Ext Len 8-bit unsigned integer. Length of the Routing header in 8-octet units, not including the first 8 octets.

Routing Type 8-bit identifier of a particular Routing header variant.

Segments Left 8-bit unsigned integer. Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination.

Type-specific data Variable-length field, of format determined by the Routing Type, and of length such that the
complete Routing header is an integer multiple of 8 octets long.

If, while processing a received packet, a node encounters a Routing header with an unrecognized Routing Type value, the required behavior of the node depends on the value of the Segments Left field, as follows:

- If Segments Left is zero, the node MUST ignore the Routing header and proceed to process the next header in the packet, whose type is identified by the Next Header field in the Routing header.
- If Segments Left is non-zero, the node MUST discard the packet and send an ICMP Parameter Problem, Code 0, message to the packet’s Source Address, pointing to the unrecognized Routing Type.

If, after processing a Routing header of a received packet, an intermediate node determines that the packet is to be forwarded onto a link whose link MTU is less than the size of the packet, the node MUST discard the packet and send an ICMP Packet Too Big message to the packet’s Source Address.

### 2.5 Fragment Header

The Fragment header is used by an IPv4 source to send a packet larger than would fit in the path MTU to its destination. This can be used as an alternative to canonical IPv4 fragmentation. As in IPv6, fragmentation using the Fragment Header is performed only by source nodes. Section 2.8.1 discusses the interaction between IPv4 extension headers and standard IPv4 fragmentation.

A Fragment header is identified by a Next Header value of 44 in the immediately preceding header and has the following format:

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

- **Next Header**: 8-bit selector. Identifies the initial header type of the Fragmentable Part of the original packet (defined below). Uses the same values as the IPv4 Protocol field [IANA-PN].
- **Reserved**: 8-bit reserved field. Initialized to zero for transmission; ignored on reception.
Fragment Offset 13-bit unsigned integer. The offset, in 8-octet units, of the data following this header, relative to the start of the Fragmentable Part of the original packet.

Res 2-bit reserved field. Initialized to zero for transmission; ignored on reception.

M flag 1 = more fragments; 0 = last fragment.

Identification 32 bits. See description below.

In order to send a packet that is too large to fit in the MTU of the path to its destination, a source node may divide the packet into fragments and send each fragment as a separate packet, to be reassembled at the receiver.

For every packet that is to be fragmented, the source node generates an Identification value. The Identification must be different than that of any other fragmented packet sent recently* with the same Source Address and Destination Address. If a Routing header is present, the Destination Address of concern is that of the final destination.

* "recently" means within the maximum likely lifetime of a packet, including transit time from source to destination and time spent awaiting reassembly with other fragments of the same packet. However, it is not required that a source node knows the maximum packet lifetime. Rather, it is assumed that the requirement can be met by implementing an algorithm that results in a low identification reuse frequency. Examples of algorithms meeting this requirement are described in [RFC7739].

The initial unfragmented packet is referred to as the "original packet" and is considered to consist of three parts, as shown:

original packet:

```
+------------------+-------------------------+---//----------------+
|  Per-Fragment    | Extension & Upper-Layer |   Fragmentable      |
|    Headers       |       Headers           |      Part           |
+------------------+-------------------------+---//----------------+
```

The Per-Fragment headers must consist of the IPv4 header plus any extension headers that must be processed by nodes en route to the destination, that is, all headers up to and including the Routing header if present, else the Hop-by-Hop Options header if present, else no extension headers.
The Extension headers are all other extension headers that are not included in the Per-Fragment headers part of the packet. For this purpose, the Encapsulating Security Payload (ESP) is not considered an extension header. The Upper-Layer header is the first upper-layer header that is not an IPv4 extension header. Examples of upper-layer headers include TCP, UDP, IPv4, IPv6, ICMPv4, and as noted ESP.

The Fragmentable Part consists of the rest of the packet after the upper-layer header or after any header (i.e., initial IPv4 header or extension header) that contains a Next Header value of No Next Header.

The Fragmentable Part of the original packet is divided into fragments. The lengths of the fragments MUST be chosen such that the resulting fragment packets fit within the MTU of the path to the packet’s destination(s). Each complete fragment, except possibly the last ("rightmost") one, is an integer multiple of 8 octets long.

The fragments are transmitted in separate "fragment packets" as illustrated:

original packet:

```
+-----------------+-----------------+--------+--------+-//-+--------+
|  Per-Fragment  |Ext & Upper-Layer|  first | second |    |  last  |
|    Headers     |    Headers      |        |        |....|        |
+-----------------+-----------------+--------+--------+-//-+--------+
```

fragment packets:

```
+-----------------+-----------------+--------+--------+-//-+--------+
|  Per-Fragment  |Ext & Upper-Layer|  first | second |    |  last  |
|    Headers     |    Headers      |        |        |....|        |
+-----------------+-----------------+--------+--------+-//-+--------+

+-----------------+-----------------+--------+--------+-//-+--------+
|  Per-Fragment  |Fragment         |  first |        |    |        |
|    Headers     |    Header       |        |        |....|        |
+-----------------+-----------------+--------+--------+-//-+--------+

+-----------------+-----------------+--------+--------+-//-+--------+
|  Per-Fragment  |Fragment         |  second|        |    |        |
|    Headers     |    Header       |        |        |....|        |
+-----------------+-----------------+--------+--------+-//-+--------+
```

T. Herbert  Expires August 31, 2019  [Page 13]
The first fragment packet is composed of:

1. The Per-Fragment headers of the original packet, with the Total Length of the original IPv4 header changed to contain the length of this fragment packet only, and the Next Header field of the last header of the Per-Fragment headers changed to 44.

2. A Fragment header containing:

   - The Next Header value that identifies the first header after the Per-Fragment headers of the original packet.
   - A Fragment Offset containing the offset of the fragment, in 8-octet units, relative to the start of the Fragmentable Part of the original packet. The Fragment Offset of the first ("leftmost") fragment is 0.
   - An M flag value of 1 as this is the first fragment.
   - The Identification value generated for the original packet.

3. Extension headers, if any, and the Upper-Layer header. These headers must be in the first fragment. Note: This restricts the size of the headers through the Upper-Layer header to the MTU of the path to the packet’s destination(s).

4. The first fragment.

The subsequent fragment packets are composed of:

1. The Per-Fragment headers of the original packet, with the Total Length of the original IPv4 header changed to contain the length of this fragment packet only, and the Next Header field of the last header of the Per-Fragment headers changed to 44.

2. A Fragment header containing:

   - The Next Header value that identifies the first header after the Per-Fragment headers of the original packet.
   - A Fragment Offset containing the offset of the fragment, in 8-octet units, relative to the start of the Fragmentable Part of the original packet.
   - An M flag value of 0 if the fragment is the last
("rightmost") one, else an M flag value of 1.

The Identification value generated for the original packet.

(3) The fragment itself.

Fragments MUST not be created that overlap with any other fragments created from the original packet.

At the destination, fragment packets are reassembled into their original, unfragmented form, as illustrated:

reassembled original packet:

+---------------+-----------------+---------+--------+-//--+--------+
| Per-Fragment  |Ext & Upper-Layer|  first  | second |     | last   |
|    Headers    |     Headers     |frag data|fragment|.....|fragment|
+---------------+-----------------+---------+--------+-//--+--------+

The following rules govern reassembly:

An original packet is reassembled only from fragment packets that have the same Source Address, Destination Address, and Fragment Identification.

The Per-Fragment headers of the reassembled packet consists of all headers up to, but not including, the Fragment header of the first fragment packet (that is, the packet whose Fragment Offset is zero). This includes the IPv4 header and any associated IPv4 options of the first fragment. The Per-Fragment headers of the reassembled packet include the following two changes:

The Next Header field of the last header of the Per-Fragment headers is obtained from the Next Header field of the first fragment’s Fragment header.

The Total Length of the reassembled packet is computed from the length of the Per-Fragment headers (including the IPv4 header) and the length and offset of the last fragment. For example, a formula for computing the Total Length of the reassembled original packet is:

\[
TL_{\text{orig}} = TL_{\text{first}} - FL_{\text{first}} - 8 + (8 \times FO_{\text{last}}) + FL_{\text{last}}
\]

where

\[
TL_{\text{orig}} = \text{Total Length of reassembled packet.}
\]
FL.first = length of fragment following Fragment header of first fragment packet.
FO.last  = Fragment Offset field of Fragment header of last fragment packet.
FL.last  = length of fragment following Fragment header of last fragment packet.

The Fragmentable Part of the reassembled packet is constructed from the fragments following the Fragment headers in each of the fragment packets. The length of each fragment is computed by subtracting from the packet’s Total Length the length of the headers before the fragment itself; its relative position in Fragmentable Part is computed from its Fragment Offset value.

The Fragment header is not present in the final, reassembled packet.

If the fragment is a whole datagram (that is, both the Fragment Offset field and the M flag are zero), then it does not need any further reassembly and should be processed as a fully reassembled packet (i.e., updating Next Header or Protocol, adjust Total Length, removing the Fragment header, etc.). Any other fragments that match this packet (i.e., the same IPv4 Source Address, IPv4 Destination Address, and Fragment Identification) should be processed independently.

The following error conditions may arise when reassembling fragmented packets:

- If insufficient fragments are received to complete reassembly of a packet within 60 seconds or a configured time limit of the reception of the first-arriving fragment of that packet, reassembly of that packet MUST be abandoned and all the fragments that have been received for that packet MUST be discarded. If the first fragment (i.e., the one with a Fragment Offset of zero) has been received, an ICMP Time Exceeded -- Fragment Reassembly Time Exceeded message SHOULD be sent to the source of that fragment.

- If the length of a fragment, as derived from the fragment packet’s Total Length field, is not a multiple of 8 octets and the M flag of that fragment is 1, then that fragment MUST be discarded and an ICMP Parameter Problem, Code 0, message SHOULD be sent to the source of the fragment, pointing to the Total Length field of the fragment packet.

- If the length and offset of a fragment are such that the Total Length of the packet reassembled from that fragment would
exceed 65,535 octets, then that fragment MUST be discarded and
an ICMP Parameter Problem, Code 0, message SHOULD be sent to
the source of the fragment, pointing to the Fragment Offset
field of the fragment packet.

o If the first fragment does not include all headers through an
  Upper-Layer header, then that fragment SHOULD be discarded and
  an ICMP Parameter Problem, Code 3, message should be sent to
  the source of the fragment, with the Pointer field set to zero.

o If any of the fragments being reassembled overlap with any
  other fragments being reassembled for the same packet,
  reassembly of that packet MUST be abandoned and all the
  fragments that have been received for that packet MUST be
discarded, and no ICMP error messages SHOULD be sent.

It should be noted that fragments may be duplicated in the
network. Instead of treating these exact duplicate fragments as
overlapping fragments, an implementation MAY choose to detect
this case and drop exact duplicate fragments while keeping the
other fragments belonging to the same packet.

The following conditions are not expected to occur frequently but are
not considered errors if they do:

The number and content of the headers preceding the Fragment
header of different fragments of the same original packet may
differ. Whatever headers are present, preceding the Fragment
header in each fragment packet, MUST be processed when the packets
arrive and prior to queuing the fragments for reassembly. Only
those headers in the Offset zero fragment packet are retained in
the reassembled packet.

Different IP options may appear in different fragments of the same
original packet. Whatever IP options are present are processed
when the packets arrive prior to processing any extension headers
or queuing the fragments for reassembly. Only the IP options in
the Offset zero fragment packet are retained in the reassembled
packet.

The Next Header values in the Fragment headers of different
fragments of the same original packet may differ. Only the value
from the Offset zero fragment packet is used for reassembly.

Other fields in the IPv4 header may also vary across the fragments
being reassembled. Specifications that use these fields may
provide additional instructions if the basic mechanism of using
the values from the Offset zero fragment is not sufficient. For
example, Section 5.3 of [RFC3168] describes how to combine the Explicit Congestion Notification (ECN) bits from different fragments to derive the ECN bits of the reassembled packet.

### 2.6 Destination Options Header

The Destination Options header is used to carry optional information that need be examined only by a packet’s destination node(s). The Destination Options header is identified by a Next Header value of 60 in the immediately preceding header and has the following format:

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

**Next Header** 8-bit selector. Identifies the type of header immediately following the Destination Options header. Uses the same values as the IPv4 Protocol field [IANA-PN].

**Hdr Ext Len** 8-bit unsigned integer. Length of the Destination Options header in 8-octet units, not including the first 8 octets.

**Options** Variable-length field, of length such that the complete Destination Options header is an integer multiple of 8 octets long. Contains one or more TLV-encoded options, as described in Section 2.2.

The only destination options defined in this document are the Pad1 and PadN options specified in Section 2.2.

### 2.7 No Next Header

The value 59 in the Protocol field of an IPv4 header or Next Header field extension header indicates that there is nothing following that header. If the Total Length field of the IPv4 header indicates the presence of octets past the end of a header whose Next Header field contains 59, those octets must be ignored and passed on unchanged if the packet is forwarded.
2.8 Interaction with standard IPv4 mechanisms

IPv4 extension headers may be used concurrently with IPv4 mechanisms such as IPv4 options and IPv4 fragmentation. This section discusses the interactions.

2.8.1 IPv4 options and IPv4 extension headers

An IPv4 packet MAY contain both IPv4 options and extension headers. IPv4 options are completely independent of IPv4 extension headers. IPv4 options MUST be processed before processing any extension headers per normal requirements of processing the IP header before the IP payload.

2.8.2 IPv4 fragmentation and IPv4 extension headers

An IPv4 packet may be fragmented both by using a Fragment extension header as well as by standard IPv4 fragmentation. The Fragment header can only be set at the source, however intermediate devices can fragment packets using standard IPv4 fragmentation. Standard IPv4 fragmentation at a source node MUST be done only after any extension headers are set in a packet or the packet was fragmented using the Fragment header. Specifically, fragmentation using the extension header MUST NOT be done on packet fragments created by standard IPv4 fragmentation. However, a packet fragment that contains a Fragment header MAY itself be fragmented by standard IPv4 fragmentation. There is no correlation between normal IPv4 fragmentation and the IPv4 Fragment header, the identifier space for each are unrelated and reassembly procedures are independent.

At a destination, if a received packet was fragmented by standard IPv4 fragmentation, it MUST be reassembled before processing any IPv4 extension headers. This requirement ensures that standard IPv4 reassembly is done before reassembly for the Fragment header.

If an IPv4 packet containing Hop-by-Hop options is fragmented using standard IPv4 fragmentation, the Hop-by-Hop Options are not set in each of the packet fragments. An intermediate node MAY process the Hop-by-Hop options in the first fragment if the complete Hop-by-Hop extension header is contained within the fragment. If the Fragment header is used with IPv4 the DF bit (Don’t Fragment) bit SHOULD be set in the IPv4 header and Path MTU discovery mechanisms SHOULD be used.
3 Encapsulating extension headers in UDP

This section defines a means to carry extension headers in Generic UDP Encapsulation (GUE). The diagram below illustrates the protocol stack when extension headers are encapsulated in GUE.

![Protocol Stack Diagram](image)

3.1 Encapsulation format

Extension headers and the trailing transport layer packet can be encapsulated in Variant 0 of GUE. The protocol format is:

```
+-----------------------------+   +-----------------------------+   +-----------------------------+   +-----------------------------+
| IPv4 or IPv6 header         |   | UDP header                  |   | GUE Header                  |   | Extension headers           |
+-----------------------------+   +-----------------------------+   +-----------------------------+   +-----------------------------+
|                             |   | GUE Header                  |   | Extension headers           |
+-----------------------------+   +-----------------------------+   +-----------------------------+   +-----------------------------+
|                             |   | Transport header            |   | Transport payload           |
+-----------------------------+   +-----------------------------+   +-----------------------------+   +-----------------------------+
```

T. Herbert                        Expires August 31, 2019                      
[Page 20]
The pertinent fields in the base GUE header are:

- **Variant** - set 0 for variant 0.
- **C bit** - Control bit. Set to zero indicating a data message.
- **Hlen** - Header length of GUE header in four byte words not including the first four bytes.
- **Proto/ctype** - The type of the encapsulated protocol. This is an IP protocol and may be an extension header. If the payload is something other than an IP protocol or the payload is encrypted or transformed, then this field is set to 59 (No Next Header) -- in this case the type of the payload is determined through other means.
- **M**: Magic number bit. If this bit is set then the GUE magic number option is present. The GUE magic number option is described below.

Any of the GUE options defined in [GUEEXT] may be set in the packet. To facilitate maintaining the correct transport layer checksum across NAT translation, the NAT checksum address option SHOULD be used ([GUEEXT]). The GUE magic number option, defined below, is used to help intermediate nodes correctly identify GUE packets.

If a transport layer protocol is encapsulated in GUE then the IP header for the transport header is taken to be the IP header of the GUE/UDP packet. In particular, an encapsulated transport header may have a checksum that includes the IP addresses in a pseudo header for checksum calculation (TCP or UDP).

### 3.2 GUE magic numbers

GUE magic numbers are used to identify a UDP payload as being a GUE payload with a high degree of probability.

The format of the GUE magic number option is:

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~               Magic value = 0xffd871a2b4e7c965                ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:
o Magic value. A 64 bit value that MUST be set to 0xffd871a2b4e7c965.

The GUE magic number option is present when the M bit is set in the GUE header flags.

3.3 Operation

This section describes the operation of encapsulating extension headers in GUE.

3.3.1 Sender processing

To encapsulate extension headers, a sender inserts a UDP and GUE header between an IP header and the first extension header.

If a sender encapsulates extension headers in GUE then it MUST NOT also set extension headers in the IPv4 or IPv6 header. When extension headers are encapsulated in GUE, the Next Header field of the IPv6 header or the Protocol field of the IPv4 header MUST be set to 17 to indicate UDP.

If the encapsulated transport protocol contains a checksum with a pseudo header and the packet may traverse a NAT, then the NAT Checksum option SHOULD be set to allow the receiver to properly adjust the received transport layer checksum. Other GUE options MAY be set per the discretion of the sender.

If the packet being encapsulated contains a Hop-by-Hop extension header then the Magic Number option MUST be used to allow intermediate nodes to process and potentially modify data in the extension header. Note that in this case the proto/ctype field in the GUE header MUST be zero indicating Hop-by-Hop options extension header.

The following guidelines apply to the source setting the magic number option:

o If the GUE checksum option is used then its payload coverage MUST be zero.

o If the GUE alternate checksum option is used then its payload coverage MUST be zero.

o If the HMAC security option is used then its Payload length MUST be zero.

o The magic number option MUST NOT be set when the GUE
fragmentation or payload transform option is used.

- The remote checksum option MAY be used concurrently with the magic number option under the assumption that intermediate nodes will not modify encapsulated transport checksum fields or attempt to verify an encapsulated transport layer checksum (in the latter case they could do that if they were to take the remote checksum offload option into account).

### 3.3.2 Destination Processing

Encapsulated extension headers in GUE is processed by normal methods of processing GUE. As described in [GUE]:

If a valid data message is received, the UDP header and GUE header are removed from the packet. The outer IP header remains intact and the next protocol in the IP header is set to the protocol from the proto field in the GUE header. The resulting packet is then resubmitted into the protocol stack to process that packet as though it was received with the protocol in the GUE header.

In the case that the GUE packet contains extension headers, the resultant packet after GUE processing is an IPv4 or IPv6 packet with extension headers. When the packet is resubmitted to the protocol stack, processing of the first extension header commences.

Note that if a routing header was encapsulated, the packet may be forwarded to another node. The packet MAY be re-encapsulated in GUE for transmission per the capabilities of the receiving node and network.

### 3.3.3 Intermediate device processing

Intermediate devices MAY process Hop-by-Hop options. In the case that GUE encapsulates Hop-by-Hop options, an intermediate node needs to parse, process, and possibly modify a UDP payload containing the GUE message with encapsulated Hop-by-Hop options. The magic number option is defined to allow intermediate nodes to identify GUE packets that might contain Hop-by-Hop options to process.

Processing of packets with encapsulated Hop-by-Hop options has the following flow:

1) Match destination UDP port number to be GUE.

2) If the GUE variant is not zero or the C bit is set (control message) then discontinue payload processing.
3) If magic number option is not present in the GUE header then discontinue payload processing.

4) If proto/ctype value is not zero (not Hop-by-Hop options) then discontinue payload processing.

5) If the GUE header indicates that a fragment option or payload transform option is present then discontinue payload processing.

6) If the checksum option is present and the payload coverage is non-zero then discontinue payload processing.

7) If the alternate checksum option is present and the payload coverage is non-zero then discontinue payload processing.

8) If the HMAC security option is present and the Payload length is not zero then discontinue payload processing.

9) Compare the magic number value in the GUE header to the defined value. If they are not equal then discontinue payload processing.

10) If the GUE checksum option is present (and payload coverage is zero) then the GUE checksum MAY be validated. If checksum validation fails, then discontinue payload processing.

11) If the alternate checksum is present (and payload coverage is zero) then the alternate checksum MAY be validated. If alternate checksum validation fails, then discontinue payload processing.

12) Process the encapsulated Hop-by-Hop options. If a Hop-by-Hop option is modified then the outer UDP checksum MUST be updated to reflect the change.

Note that intermediate MUST not modify any fields other than data in modifiable Hop-by-Hop options or the UDP checksum which needs to be updated when UDP payload is modified. In particular, intermediate nodes MUST NOT modify the GUE header nor an data aside from that in modifiable Hop-by-Hop options.
4 Security Considerations

This specification enables use of IPv6 extension headers in IPv4. Related security mechanisms of IPv6 extension headers can be applied for use with IPv4 extension headers.

When extension headers are encapsulated in GUE, normal GUE security mechanisms can be used. If an intermediate node might modify GUE payload to process modifiable extension headers, then a GUE security algorithm cannot take input to authenticate the GUE payload. If authentication is necessary, then an Authentication header may be used that treats modifiable data fields as zero-valued octets when computing or verifying the packet’s authenticating value.

5 IANA Considerations

IANA is requested to assign a value in the "GUE flag-fields" registry for the Magic Number option.

<table>
<thead>
<tr>
<th>Flags bits</th>
<th>Field size</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 10</td>
<td>8 bytes</td>
<td>Magic number</td>
<td>This document</td>
</tr>
</tbody>
</table>

6 References

6.1 Normative References


[GUE] Herbert, T., Yong, L., Zia, ),, "Generic UDP Encapsulation", draft-ietf-intarea-gue-06

6.2 Informative References


[IPNOPT] Rodrigo Fonseca, George Manning Porter, Randy H. Katz, Scott Shenker and Ion Stoica, "IP Options are not an option", <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-24.html>

[FAST] Herbert, T., "Firewall and Service Tickets", draft-herbert-fast-03


[IOAM] F. Brockners, S. Bhandari, V. Govindan, C. Pignataro, H. Gredler, J. Leddy, S. Youell, T. Mizrahi, D. Mozes, P. Lapukhov, R. Chang, "Encapsulations for In-situ OAM Data" draft-brockners-inband-oam-transport-05


Author’s Address

Tom Herbert
Quantionum
Santa Clara, CA
USA

Email: tom@quantonium.net