INTERNET-DRAFT T. Herbert
Intended Status: Proposed Standard Quantonium
Expires: September 9, 2019 L. Yong
Independent
F. Templin
Boeing
March 8, 2019

Extensions for Generic UDP Encapsulation
draft-ietf-intarea-gue-extensions-06

Abstract

This specification defines a set of the initial optional extensions for Generic UDP Encapsulation (GUE).

Status of this Memo

Copyright and License Notice

Table of Contents

1. Introduction

Generic UDP Encapsulation (GUE) [I.D.ietf-gue] is a generic and extensible encapsulation protocol. This specification defines an initial set of optional extensions for variant 0 of GUE. These extensions are the Group Identifier, Security, Fragmentation, Payload Transform, Remote Checksum Offload, Checksum, NAT Address Checksum, and Alternate Checksum.

2. GUE header format with optional extensions

The format of a variant 0 GUE header with optional extensions 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\
|        Source port            |       Destination port        | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ UDP
|           Length              |           Checksum            | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+/
| 0 |C|   Hlen  |  Proto/ctype  |G| SEC |F|T|R|K|N|A|   Rsvd    |\
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
|                  Group identifier (optional)                  | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
|                                                               | |
~                       Security (optional)                     ~ |
|                                                               | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
|                                                               | |
```

```
   +                     Fragmentation (optional)                  + |
   |                                                               | GUE
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   |                   Payload transform (optional)                | |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   |                 Remote checksum offload (optional)            | |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   |                        Checksum (optional)                    | |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   |                 NAT Address Checksum (optional)               | |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   |                                                               | |
   +                  Alternate checksum (optional)                + |
   |                                                               | |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
   |                                                               | |
   ~                     Private data (optional)                   ~ |
   |                                                               | |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+/
```


The contents of the UDP header are described in [I.D.ietf-gue].

The GUE header consists of:

- o Variant: Set to 0 to indicate GUE encapsulation header. Note that variant 1 (direct IP encapsulation) does not allow optional extensions.

- o C: C-bit. Indicates the GUE payload is a control message when set, a data message when not set. GUE optional extensions can be used with either control or data messages unless otherwise stated in the specification of the extension.

- o Hlen: Length in 32-bit words of the GUE header, including optional extension fields and private data but not the first four bytes of the header. Computed as (header_len - 4) / 4. The length of the encapsulated packet is determined from the UDP length and the Hlen: encapsulated_packet_length = UDP_Length - 12 - 4 * Hlen.

- o Proto/ctype: If the C-bit is not set this indicates the IP protocol number for the packet in the payload; if the C bit is set this is the type of control message in the payload. The next header begins at the offset provided by Hlen. When the payload transform option or fragmentation option is used this field SHOULD be set to protocol number 59 for a data message, or zero for a control message, to indicate there is no parsable protocol in the payload.

- G: Indicates the the group identifier extension field is present. The group identifier option is described in section 3.
- SEC: Indicates security extension field is present. The security option is described in section 4.
- F: Indicates fragmentation extension field is present. The fragmentation option is described in section 5.
- T: Indicates payload transform extension field is present. The payload transform option is described in section 6.
- R: Indicates the remote checksum extension field is present. The remote checksum offload option is described in section 7.
- K: Indicates checksum extension field is present. The checksum option is described in section 8.
- N: Indicates NAT address checksum field is present. The NAT

address checksum option is described in section 9.

- A: Indicates alternative checksum field is present. The alternative checksum option is described in section 10.
- Private data is described in [I.D.ietf-gue].

3. Group identifier option

A group identifier classifies packets that logically belong to the same group. Groups are arbitrarily defined for different purposes and their definition is shared between the communicating end nodes.

3.1. Extension field format

The presence of the GUE group identifier option is indicated by the G flag bit of the GUE header.

The format of the group identifier 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Group identifier                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- o Group identifier: Identifier value of a group.

3.2. Usage

The group identifier is set by an encapsulator to indicate that a packet belongs to a group. Groups may be arbitrarily defined to classify packets. Specific use cases of the group identifier may be defined in other documents ([I.D.hy-nvo3-gue-4-nvo] defines a use of this field to contain a virtual networking identifier for implementing network virtualization).

Intermediate nodes MAY apply semantics to group identifiers if group identifier information is shared and made global within a network. For instance, a firewall could block packets based on a group identifier that serves as a virtual identifier for a tenant.

4. Security option

The GUE security option provides origin authentication and integrity protection of the GUE header at tunnel end points to guarantee

isolation between tunnels and mitigate Denial of Service attacks.

4.1. Extension field format

The presence of the GUE security option is indicated by the SEC flag bits of the GUE header.

The format of the security 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 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                           Security                            ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- o Security (variable length). Contains the security information. The specific semantics and format of this field are expected to be negotiated between the two communicating nodes.

To provide security capability, the SEC flags MUST be set. Different field sizes allow different methods and extensibility. The use of the security field is expected to be negotiated out-of-band between two tunnel end points.

The values in the SEC flags are:

- o 000b - No security field
- o 001b - 64 bit security field
- o 010b - 128 bit security field
- o 011b - 256 bit security field
- o 100b - 320 bit security field (HMAC)
- o 101b, 110b, 111b - Reserved values

4.2. Usage

The GUE security option is used to provide integrity and authentication of the GUE header. Security parameters (interpretation of security field, key management, etc.) are expected to be negotiated out-of-band between two communicating hosts. Two security

algorithms are defined below.

If the GUE security option is present in a packet, the receiver MUST validate the security before processing other fields or accepting the packet. If the security option is not present, but the encapsulator and decapsulator have agreed that security is required, the receiver MUST drop the packet as failing security checks. Note that this provision covers the case where the security flags bits are corrupted such that they are reset to zero which would be interpreted as no security field being present.

4.3. Cookies

The security field may be used as a cookie. This would be similar to the cookie mechanism described in L2TP [RFC3931], and the general properties should be the same. A cookie MAY be used to validate the encapsulation. A cookie is a shared value between an encapsulator and decapsulator which SHOULD be chosen randomly and MAY be changed periodically. Different cookies MAY be used for logical flows between the encapsulator and decapsulator; for instance packets sent with different VNIs in network virtualization [I.D.hy-nvo3-gue-4-nvo] might have different cookies. Cookies can be 64, 128, or 256 bits in size.

4.4.1. Extension field format

The cookie security option is a 64, 128, or 256 bit field. The security flags are set to 001b, 010b, 011b respectively for the corresponding field size.

The format of the field 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                            Cookie                             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields are:

- o Cookie: Shared cookie value between encapsulator and decapsulator

4.3.1. Operation

4.3.1.1. Transmitter operation

The procedure for setting the GUE security cookie option on transmit is:

1) Create the GUE header including the security field with the selected length for the cookie. Set the cookie to the value that is shared with decapsulator.

4.3.1.2. Receiver operation

The procedure for verifying the security cookie is:

1) Compare the received cookie to the expected shared cookie. If both the lengths are equal and the cookie values are equal then that packet is accepted, if the lengths or values are not equal then verification failed and the packet MUST be dropped.

4.4. HMAC

Key-hashed message authentication code (HMAC) is a strong method of checking integrity and authentication of data. This sections defines a GUE security option for HMAC. Note that this is based on the HMAC TLV description in "IPv6 Segment Routing Header (SRH)" [I.D.previdi-6man-sr-header].

4.4.1. Extension field format

The HMAC option is a 320 bit field (40 octets). The security flags are set to 100b to indicate the presence of a 320 bit security field.

The format of the field 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          HMAC Key-id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Payload offset          |         Payload length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                         HMAC (256 bits)                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields are:

o HMAC Key-id: opaque field to allow multiple hash algorithms or key selection

- o Payload offset: offset in payload that indicates the first octet of payload data included in the HMAC.

- o Payload length: length of payload data starting from the payload offset to be included in the HMAC calculation. Zero indicates no payload data in used in the calculation.

- o HMAC: Output of HMAC computation

The HMAC field is the output of the HMAC computation (per [RFC2104]) using a pre-shared key identified by HMAC Key-id and of the text which consists of the concatenation of:

- o The IP addresses

- o The GUE header including all optional extensions and any private data. For the purposes of calculating the HMAC value, the HMAC value is set all zeroes.

- o Optionally some or all of GUE payload. The portion of payload covered is indicated by an offset into the payload and a length relative to the offset.

The purpose of the HMAC option is to verify the validity, the integrity, and the authentication of the GUE header itself and optionally some or all of the GUE payload.

The HMAC Key-id field allows for the simultaneous existence of several hash algorithms (SHA-256, SHA3-256 ... or future ones) as well as pre-shared keys. The HMAC Key-id field is opaque, i.e., it has neither syntax nor semantic. Having an HMAC Key-id field allows for pre-shared key roll-over when two pre-shared keys are supported for a while when GUE endpoints converge to a fresher pre-shared key.

4.4.2. Selecting a hash algorithm

The HMAC field in the HMAC option is 256 bits wide. Therefore, the HMAC MUST be based on a hash function whose output is at least 256 bits. If the output of the hash function is 256 bits, then this output is simply inserted in the HMAC field. If the output of the hash function is larger than 256 bits, then the output value is truncated to 256 bits by taking the least-significant 256 bits and inserting them in the HMAC field.

GUE implementations can support multiple hash functions but MUST implement SHA-2 [FIPS180-4] and its SHA-256 variant.

4.4.3. Pre-shared key management

The field HMAC Key-id allows for:

- Key roll-over: when there is a need to change the key (the hash pre-shared secret), then multiple pre-shared keys can be used simultaneously. A decapsulator can have a table of <HMAC Key-id, pre-shared secret> for the currently active and future keys.

- Different algorithms: by extending the previous table to <HMAC Key-id, hash function, pre-shared secret>, the decapsulator can also support simultaneously several hash algorithms

The pre-shared secret distribution can be done:

- In the configuration of the endpoints

- Dynamically using a trusted key distribution such as [RFC6407]

4.4.4. Operation

4.4.4.1. Transmitter operation

The procedure for setting the GUE HMAC option on transmit is:

1) Create the GUE header including the 320 bit security field to hold the HMAC option. Set the HMAC Key-Id, payload length, and payload offset appropriately. The 16 byte HMAC field is initialized to zero.

2) Calculate the HMAC hash over the concatenation of the IP source and destination addresses. The particular hash and keys are

agreed between the encpasulator and decapsulator out of band, and the key for input to the hash is the one indicated by the Key-Id amongst the set of shared keys.

3) Continue the the HMAC hash calculation from the start of the GUE header through the its length as indicated in GUE Hlen.

4) Continue the calculation to cover the payload portion if payload coverage is enabled (payload coverage field is non-zero). The calculation continues at the payload offset for payload length bytes.

5) Set the resultant hash value in the HMAC field.

4.4.4.2. Receiver operation

The procedure for verifying the HMAC security option is:

1) If the payload offset plus the payload coverage length is greater than the length of the encapsulated payload then drop the packet.

2) Note value in the HMAC field and set the HMAC field to zero.

3) Calculate the HMAC hash over the concatenation of the IP source and destination addresses. The particular hash and keys are agreed between the encpasulator and decapsulator out of band, and the key for input to the hash is the one indicated by the Key-Id amongst the set of shared keys.

4) Continue the HMAC hash calculation from the start of the GUE header through the its length as indicated in GUE Hlen.

5) Continue the calculation to cover the payload portion if payload coverage is enabled (payload length field is non-zero). The calculation continues at the payload offset for payload length bytes.

6) Compare the computed HMAC value with the original value of the HMAC field. If they are equal then the packet is accepted, if they are not equal then verification failed and the packet MUST

be dropped.

7) Restore the HMAC field to its original value.

4.5. Interaction with other optional extensions

If GUE fragmentation (section 5) is used in concert with the GUE security option, the security option processing is performed after fragmentation at the encapsulator and before reassembly at the decapsulator.

The GUE payload transform option (section 6) may be used in concert with the GUE security option. The payload transform option could be used to encrypt the GUE payload to provide privacy for an encapsulated packet during transit. The security option provides authentication and integrity for the GUE header (including the payload transform field in the header). The two functions are processed separately at tunnel end points. A GUE tunnel can use both functions or use one of them. Section 6.3 details handling when both are used in a packet.

5. Fragmentation option

The fragmentation option allows an encapsulator to perform fragmentation of packets being ingress to a tunnel. Procedures for fragmentation and reassembly are defined in this section. This specification adapts the procedures for IP fragmentation and reassembly described in [RFC0791] and [RFC8200]. Fragmentation can be performed on both data and control messages in GUE.

5.1. Motivation

This section describes the motivation for having a fragmentation option in GUE.

MTU and fragmentation issues with In-the-Network Tunneling are described in [RFC4459]. Considerations need to be made when a packet is received at a tunnel ingress point which may be too large to traverse the path between tunnel endpoints.

There are four suggested alternatives in [RFC4459] to deal with this:

1) Fragmentation and Reassembly by the Tunnel Endpoints

2) Signaling the Lower MTU to the Sources

3) Encapsulate Only When There is Free MTU

4) Fragmentation of the Inner Packet

Many tunneling protocol implementations have assumed that fragmentation should be avoided, and in particular alternative #3 seems preferred for deployment. In this case, it is assumed that an operator can configure the MTUs of links in the paths of tunnels to ensure that they are large enough to accommodate any packets and required encapsulation overhead. This method, however, may not be feasible in certain deployments and may be prone to misconfiguration in others.

Similarly, the other alternatives have drawbacks that are described in [RFC4459]. Alternative #2 implies use of something like Path MTU Discovery which is not known to be sufficiently reliable. Alternative #4 is not permissible with IPv6 or when the DF bit is set for IPv4, and it also introduces other known issues with IP fragmentation.

For alternative #1, fragmentation and reassembly at the tunnel endpoints, there are two possibilities: encapsulate the large packet and then perform IP fragmentation, or segment the packet and then encapsulate each segment (a non-IP fragmentation approach).

Performing IP fragmentation on an encapsulated packet has the same

issues as that of normal IP fragmentation. Most significant of these is that the Identification field is only sixteen bits in IPv4 which introduces problems with wraparound as described in [RFC4963].

The second possibility of alternative #1 follows the suggestion expressed in [RFC2764] and the fragmentation feature described in the AERO protocol [I.D.templin-aerolink]; that is for the tunneling protocol itself to incorporate a fragmentation and reassembly capability. In this method, fragmentation is part of the encapsulation and an encapsulation header contains the information for reassembly. This differs from IP fragmentation in that the IP headers of the original packet are not replicated for each fragment.

Incorporating fragmentation into the encapsulation protocol has some advantages:

- o At least a 32 bit identifier can be defined to avoid issues of the 16 bit Identification in IPv4.
- o Encapsulation mechanisms for security and identification, such as group identifiers, can be applied to each segment.
- o This allows the possibility of using alternate fragmentation and reassembly algorithms (e.g. fragmentation with Forward Error Correction).
- o Fragmentation is transparent to the underlying network so it is unlikely that fragmented packet will be unconditionally dropped as might happen with IP fragmentation.

5.2. Scope

This specification describes the mechanics of fragmentation in Generic UDP Encapsulation. The operational aspects and details for higher layer implementation must be considered for deployment, but are considered out of scope for this document. The AERO protocol [I.D.templin-aerolink] defines one use case of fragmentation with encapsulation.

5.3. Extension field format

The presence of the GUE fragmentation option is indicated by the F bit in the GUE header.

The format of the fragmentation 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Fragment offset   |Res|M|   Orig-proto    |               |
```

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               +
 |                        Identification                         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- o Fragment offset: This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

- o Res: Two bit reserved field. MUST be set to zero for transmission. If set to non-zero in a received packet then the packet MUST be dropped.

- o M: More fragments bit. Set to 1 when there are more fragments following in the datagram, set to 0 for the last fragment.

- o Orig-proto: The control type (when the C-bit in the GUE header is set) or the IP protocol (when C-bit is not set) of the fragmented packet.

- o Identification: 40 bits. Identifies fragments of a fragmented packet.

Pertinent GUE header fields to fragmentation are:

- o C-bit: This is set for each fragment based on the whether the original packet being fragmented is a control or data message.

- o Proto/ctype - For the first fragment (fragment offset is zero) this is set to that of the original packet being fragmented (either will be a control type or IP protocol). For other fragments, this is set to zero for a control message being fragmented, or to "No next header" (protocol number 59) for a data message being fragmented.

- o F bit - Set to indicate presence of the fragmentation extension field.

5.4. Fragmentation procedure

If an encapsulator determines that a packet must be fragmented (e.g. the packet's size exceeds the Path MTU of the tunnel) it should divide the packet into fragments and send each fragment as a separate

GUE packet, to be reassembled at the decapsulator (tunnel egress).

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 within the past 60 seconds (Maximum Segment Lifetime) or configured time with the same tunnel identification-- that is the same outer source and destination addresses, same UDP ports, same orig-proto, and same group identifier if present.

The initial, unfragmented, and unencapsulated packet is referred to as the "original packet". This will be a layer 2 packet, layer 3 packet, or the payload of a GUE control message:

```
+-------------------------------//-------------------------------+
|                        Original packet                         |
|             (e.g. an IPv4, IPv6, Ethernet packet)              |
+------------------------------//--------------------------------+
```

Fragmentation and encapsulation are performed on the original packet in sequence. First the packet is divided up in to fragments, and then each fragment is encapsulated. Each fragment, except possibly the last ("rightmost") one, is an integer multiple of eight octets long. Fragments MUST be non-overlapping. The number of fragments SHOULD be minimized, and all but the last fragment should be approximately equal in length.

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

```
+--------------+--------------+---------------+--//--+----------+
|    first     |    second    |     third     |      |   last   |
|   fragment   |   fragment   |   fragment    | .... | fragment |
+--------------+--------------+---------------+--//--+----------+
```

Each fragment is encapsulated as the payload of a GUE packet. This is illustrated as:

```
+------------------+----------------+-----------------------+
|  IP/UDP header   |   GUE header   |         first         |
|                  | w/ frag option |        fragment       |
+------------------+----------------+-----------------------+

+------------------+----------------+-----------------------+
|  IP/UDP header   |   GUE header   |         second        |
|                  | w/ frag option |        fragment       |
+------------------+----------------+-----------------------+
                             o
```

o


```
+------------------+----------------+-----------------------+
|  IP/UDP header   |   GUE header   |         last          |
|                  | w/ frag option |       fragment        |
+------------------+----------------+-----------------------+
```

Each fragment packet is composed of:

(1) Outer IP and UDP headers as defined for GUE encapsulation. The IP addresses and UDP ports MUST be the same for all fragments of a fragmented packet.

(2) A GUE header that indicates the fragmentation option is present. The C-bit and and proto/ctype are set appropriately as described above.

(3) The GUE fragmentation option. Orig-protocol is set to the protocol of the original packet. The M-bit is set for all fragments except the last one. Fragment offset is set as the offset of each fragment in the original packet.

(4) Other GUE extensions.

(5) The fragment itself as payload of the GUE packet.

5.5. Reassembly procedure

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

```
+------------------------------//-------------------------------+
|                          Original packet                      |
|              (e.g. an IPv4, IPv6, Ethernet packet)            |
+-----------------------------//--------------------------------+
```

The following rules govern reassembly:

The IP/UDP/GUE headers of each packet are retained until all fragments have arrived. The reassembled packet is then composed of the decapsulated payloads in the GUE packets, and the IP/UDP/GUE headers are discarded.

When a GUE packet is received with the fragment extension, the proto/ctype field in the GUE header MUST be validated. In the case that the packet is a first fragment (fragment offset is zero), the proto/ctype in the GUE header MUST equal the orig-proto value in the fragmentation option. For subsequent fragments, (fragment offset is non-zero) the proto/ctype in the GUE header MUST be 0 for a control message or 59 (no-next-hdr)

for a data message. If the proto/ctype value is invalid for a received packet it MUST be dropped.

An original packet is reassembled only from GUE fragment packets that have the same outer source address, destination address, UDP source port, UDP destination port, GUE header C-bit, group identifier if present, orig-proto value in the fragmentation option, and Fragment Identification. The protocol type or control message type (depending on the C-bit) for the reassembled packet is the value of the GUE header proto/ctype field in the first fragment.

The following error conditions can arise when reassembling fragmented packets with GUE encapsulation:

If insufficient fragments are received to complete reassembly of a packet within 60 seconds (or a configurable period) 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 payload length of a fragment is not a multiple of 8 octets and the M flag of that fragment is 1, then that fragment MUST be discarded.

If the length and offset of a fragment are such that the payload length of the packet reassembled from that fragment would exceed 65,535 octets, then that fragment MUST be discarded.

If a fragment overlaps another fragment already saved for reassembly then the new fragment that overlaps the existing fragment MUST be discarded.

If the first fragment is too small then it is possible that it does not contain the necessary headers for a stateful firewall. Sending small fragments like this has been used as an attack on IP fragmentation. To mitigate this problem, an implementation SHOULD ensure that the first fragment contains the headers of the encapsulated packet at least through the transport header.

A GUE node MUST be able to accept a fragmented packet that, after reassembly and decapsulation, is as large as 1500 octets. This means that the node must configure a reassembly buffer that is at least as large as 1500 octets plus the maximum-sized encapsulation headers that may be inserted during encapsulation. Implementations may find it more convenient and efficient to configure a reassembly buffer size of 2KB which should be large enough to accommodate even the

largest set of encapsulation headers and provides a natural memory page size boundary.

5.6. Security Considerations

Exploits that have been identified with IP fragmentation are conceptually applicable to GUE fragmentation.

Attacks on GUE fragmentation can be mitigated by:

- o Hardened implementation that applies applicable techniques from implementation of IP fragmentation.

- o Application of GUE security (section 4) or IPsec [RFC4301]. Security mechanisms can prevent spoofing of fragments from unauthorized sources.

- o Implement fragment filter techniques for GUE encapsulation as described in [RFC1858] and [RFC3128].

- o Do not accept data in overlapping segments.

- o Enforce a minimum size for the first fragment.

6. Payload transform option

The payload transform option indicates that the GUE payload has been transformed. Transforming a payload is done by running a function over the data and possibly modifying it (encrypting it for instance). The payload transform option indicates the method used to transform the data so that a decapsulator is able to validate and reverse the transformation to recover the original data. Payload transformations include encryption, authentication, CRC coverage, and compression. This specification defines a transformation for DTLS.

6.1. Extension field format

The presence of the GUE payload transform option is indicated by the T bit in the GUE header.

The format of Payload Transform Field 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |   P_C_type    |             Info              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


The fields of the option are:

Type: Payload Transform Type or Code point. Each payload transform mechanism must have one code point registered in IANA. This document specifies:

0x01: for DTLS [RFC6347]

0x80~0xFF: for private payload transform types

A private payload transform type can be used for experimental purposes or proprietary mechanisms.

P_C_type: Indicates the protocol or control type of the untransformed payload. When payload transform option is present, proto/ctype in the GUE header is set to 59 ("No next header") for a data message and zero for a control message. The IP protocol or control message type of the untransformed payload MUST be encoded in this field. The benefit of this rule is to prevent a middle box from inspecting the encrypted payload according to GUE next protocol. The assumption here is that a middle box may understand GUE base header but does not understand GUE option flag definitions.

Info: A field that can be set according to the requirements of each payload transform type. If the specification for a payload transform type does not specify how this field is to be set, then the field MUST be set to zero.

6.2. Usage

The payload transform option provides a mechanism to transform or interpret the payload of a GUE packet. The Type field provides the method used to transform the payload, and the P_C_type field provides the protocol or control message type of the payload before being transformed. The payload transformation option is generic so that it can have both security related uses (such as DTLS) as well as non security related uses (such as compression, CRC, etc.).

An encapsulator performs payload transformation before transmission, and a decapsulator performs the reverse transformation before accepting a packet. For example, if an encapsulator transforms a payload by encrypting it, the peer decaspsulator MUST decrypt the payload before accepting the packet. If a decapsulator fails to perform the reverse transformation or cannot validate the transformation it MUST discard the packet and MAY generate an alert to the management system.

6.3. Interaction with other optional extensions

If GUE fragmentation (section 5) is used in concert with the GUE transform option, the transform option processing is performed after fragmentation at the encapsulator and before reassembly at the decapsulator. If the payload transform changes the size of the data being fragmented this must be taken into account during fragmentation.

If both the security option and the payload transform are used in a GUE packet, an encapsulator MUST perform the payload transformation first, set the payload transform option in the GUE header, and then create the security option. A decapsulator does processing in reverse-- the security option is processed (GUE header is validated) and then the reverse payload transform is performed.

In order to get flow entropy from the payload, an encapsulator should derive the flow entropy before performing a payload transform.

6.4. DTLS transform

The payload of a GUE packet can be secured using Datagram Transport Layer Security (DTLS) [RFC6347]. An encapsulator would apply DTLS to the GUE payload so that the payload packets are encrypted and the GUE header remains in plaintext. The payload transform option is set to indicate that the payload is interpreted as a DTLS record.

The payload transform option for DTLS 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      1        |    P_C_type   |              0                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

DTLS [RFC6347] provides a packet fragmentation capability. To avoid packet fragmentation being performed multiple times, a GUE encapsulator SHOULD use GUE fragmentation and not DTLS fragmentation.

DTLS usage is limited to a single DTLS session for any specific tunnel encapsulator/decapsulator pair (identified by source and destination IP addresses). Both IP addresses MUST be unicast addresses - multicast traffic is not supported when DTLS is used. A GUE tunnel decapsulator implementation that supports DTLS can establish DTLS sessions with one or multiple tunnel encapsulators, and likewise a GUE tunnel encapsulator implementation can establish DTLS sessions with one or multiple decapsulators.

7. Remote checksum offload option

Remote checksum offload is mechanism that provides checksum offload of encapsulated packets using rudimentary offload capabilities found in most Network Interface Card (NIC) devices. Many NIC implementations can only offload the outer UDP checksum in UDP encapsulation. Remote checksum offload is described in [UDPENCAP].

In remote checksum offload the outer header checksum, that in the outer UDP header, is enabled in packets and, with some additional meta information, a receiver is able to deduce the checksum to be set for an inner encapsulated packet. Effectively this offloads the computation of the inner checksum. Enabling the outer checksum in encapsulation has the additional advantage that it covers more of the packet, including the encapsulation headers, than an inner checksum.

7.1. Extension field format

The presence of the GUE remote checksum offload option is indicated by the R bit in the GUE header.

The format of remote checksum offload field 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Checksum start         |       Checksum offset         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- o Checksum start: starting offset for checksum computation relative to the start of the encapsulated payload. This is typically the offset of a transport header (e.g. UDP or TCP).

- o Checksum offset: Offset relative to the start of the encapsulated packet where the derived checksum value is to be written. This typically is the offset of the checksum field in the transport header (e.g. UDP or TCP).

7.2. Usage

7.2.1. Transmitter operation

The typical actions to set remote checksum offload on transmit are:

1) Transport layer creates a packet and indicates in internal packet meta data that checksum is to be offloaded to the NIC

(normal transport layer processing for checksum offload). The checksum field is populated with the bitwise "not" of the checksum of the pseudo header or zero as appropriate.

2) Encapsulation layer adds its headers to the packet including the remote checksum offload option. The start offset and checksum offset are set accordingly.

3) Encapsulation layer arranges for checksum offload of the outer header checksum (i.e. UDP checksum).

4) Packet is sent to the NIC. The NIC will perform transmit checksum offload and set the checksum field in the outer header. The inner header and rest of the packet are transmitted without modification.

7.2.2. Receiver operation

The typical actions a host receiver does to support remote checksum offload are:

1) Receive packet and validate outer checksum following normal processing (i.e. validate non-zero UDP checksum).

2) Validate the remote checksum option. If checksum start is greater than the length of the packet, then the packet MUST be dropped. If checksum offset is greater then the length of the packet minus two, then the packet MUST be dropped.

3) Deduce full checksum for the IP packet. If a NIC is capable of receive checksum offload it will return either the full checksum of the received packet or an indication that the UDP checksum is correct. Either of these methods can be used to deduce the checksum over the IP packet [UDPENCAP].

4) From the packet checksum subtract the checksum computed from the start of the packet (outer IP header) to the offset in the packet indicted by checksum start in the meta data. The result is the deduced checksum to set in the checksum field of the encapsulated transport packet.

   In pseudo code:

```
csum: initialized to checksum computed from start (outer IP
      header) to the end of the packet
start_of_packet: address of start of packet
encap_payload_offset: relative to start_of_packet
csum_start: value from the checksum start field
```

```
checksum(start, len): function to compute checksum from start
      address for len bytes

csum -= checksum(start_of_packet, encap_payload_offset +
                 csum_start)
```

5) Write the resultant checksum value into the packet at the offset provided by checksum offset in the meta data.

In pseudo code:

```
csum_offset: value from the checksum offset field

*(start_of_packet + encap_payload_offset +
  csum_offset) = csum
```

6) Checksum is verified at the transport layer using normal processing. This should not require any checksum computation over the packet since the complete checksum has already been provided.

7.3. Security Considerations

Remote checksum offload allows a means to change the GUE payload before being received at a decapsulator. In order to prevent misuse of this mechanism, a decapsulator MUST apply security checks on the GUE payload only after checksum remote offload has been processed.

8. Checksum option

The GUE checksum option provides a checksum that covers the GUE header, a GUE pseudo header, and optionally all or part of the GUE payload. The GUE pseudo header includes the corresponding IP addresses as well as the UDP ports of the encapsulating headers. This checksum should provide protection against address corruption in IPv6 when the UDP checksum is zero. Additionally, the GUE checksum provides protection of the GUE header when the UDP checksum is set to zero with either IPv4 or IPv6. In particular, the GUE checksum can provide protection for some sensitive data, such as the virtual network identifier ([I.D.hy-nvo3-gue-4-nvo]), which when corrupted could lead to mis-delivery of a packet to the wrong virtual network.

8.1. Extension field format

The presence of the GUE checksum option is indicated by the K bit in the GUE header.

The format of the checksum extension 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |       Payload coverage        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- o Checksum: Computed checksum value. This checksum covers the GUE header (including fields and private data covered by Hlen), the GUE pseudo header, and optionally all or part of the payload (encapsulated packet).

- o Payload coverage: Number of bytes of payload to cover in the checksum. Zero indicates that the checksum only covers the GUE header and GUE pseudo header. If the value is greater than the encapsulated payload length, the packet MUST be dropped.

8.2. Requirements

The GUE header checksum SHOULD be set on transmit when using a zero UDP checksum with IPv6.

The GUE header checksum SHOULD be used when the UDP checksum is zero for IPv4 if the GUE header includes data that when corrupted can lead to misdelivery or other serious consequences, and there is no other mechanism that provides protection (no security field that checks integrity for instance).

The GUE header checksum SHOULD NOT be set when the UDP checksum is non-zero. In this case the UDP checksum provides adequate protection and this avoids convolutions when a packet traverses NAT that does address translation (in that case the UDP checksum is required).

8.3. GUE checksum pseudo header

The GUE pseudo header checksum is included in the GUE checksum to provide protection for the IP and UDP header elements which when corrupted could lead to misdelivery of the GUE packet. The GUE pseudo header checksum is similar to the standard IP pseudo header defined in [RFC0768] and [RFC0793] for IPv4, and in [RFC8200] for IPv6.

The GUE pseudo header for IPv4 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Source Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source port          |       Destination port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The GUE pseudo header for IPv6 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
```

```
+                     Destination Address                       +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Source port           |       Destination port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The GUE pseudo header does not include payload length or protocol as in the standard IP pseudo headers. The length field is deemed unnecessary for inclusion because a corrupted length field should not cause mis-delivery, the GUE checksum is applied after GUE fragmentation, and without the length field the GUE pseudo header checksum is the same for all packets of flow.

8.4. Usage

The GUE checksum is computed and verified following the standard process for computing the Internet checksum [RFC1071]. Checksum computation may be optimized per the mathematical properties

including parallel computation and incremental updates.

8.4.1. Transmitter operation

The procedure for setting the GUE checksum on transmit is:

1) Create the GUE header including the checksum and payload coverage fields. The checksum field is initially set to zero.

2) Calculate the 1's complement checksum of the GUE header from the start of the header through the its length as indicated in GUE Hlen.

3) Calculate the checksum of the GUE pseudo header for IPv4 or IPv6.

4) Calculate checksum of payload portion if payload coverage is enabled (payload coverage field is non-zero). If the length of the payload coverage is odd, logically append a single zero byte for the purposes of checksum calculation.

5) Add and fold the computed checksums for the GUE header, GUE pseudo header, and payload coverage.

6) Set the bitwise not of the resultant value in the GUE checksum field.

8.4.2. Receiver operation

If the GUE checksum option is present, the receiver MUST validate the checksum before processing any other fields or accepting the packet.

The procedure for verifying the checksum is:

1) If the payload coverage length is greater than the length of the encapsulated payload then drop the packet.

2) Calculate the checksum of the GUE header from the start of the header to the end as indicated by Hlen.

3) Calculate the checksum of the GUE pseudo header for IPv4 or IPv6.

4) Calculate the checksum of payload if payload coverage is enabled (payload coverage is non-zero). If the length of the payload coverage is odd logically append a single zero byte for the purposes of checksum calculation.

5) Sum the computed checksums for the GUE header, GUE pseudo header, and payload coverage. If the result is all 1 bits (-0 in 1's complement arithmetic), the checksum is valid and the packet is accepted; otherwise the checksum is considered invalid and the packet MUST be dropped.

8.5. Corrupted checksum flag

Note that the GUE checksum does not protect against the checksum flag (K flag) being corrupted. If an encapsulator sets the checksum flag and option but the K bit flips to be zero, then a decapsulator will incorrectly process the GUE packet as not having a checksum field.

To mitigate this issue an encapsulator and depcapsulator might agree that checksum is always required. This agreement could be established by configuration or capability negotiation.

8.6. Security Considerations

The checksum option is only a mechanism for corruption detection, it is not a security mechanism. To provide integrity checks or authentication of the GUE header, the GUE security option SHOULD be used.

9. NAT checksum address option

The NAT address checksum (NAC) option contains the checksum computed over the IP addresses of the packet. The computed value can be used by a receiver to adjust a transport layer checksum that was affected by NAT changing the IP addresses. This option is only useful when GUE encapsulates a transport layer packet that has a checksum with a pseudo header that includes the IP addresses (such as TCP or UDP).

9.1. Extension field format

The presence of the NAT address checksum option is indicated by the N bit in the GUE header.

The format of the NAT checksum address extension 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


The fields of the option are:

- o Checksum: Computed checksum value. This checksum covers the outer IP addresses.

- o Reserved: Must be zero on transmit.

9.2. Usage

The NAT address extension SHOULD be set on transmit when encapsulating a transport layer packet whose checksum might be affected by NAT being performed on the outer IP header. If this option is used then the UDP checksum MUST be used (sent with non-zero value).

The NAT address checksum is computed using the Internet checksum [RFC1071].

9.2.1. Transmitter operation

The procedure for setting the GUE checksum on transmit is:

An encapsulator computes the checksum value over the IP addresses in the IP header.

1) Compute the ones complement checksum over the source and destination IPv4 or IPv6 addresses.

2) Set the resultant value in the Checksum field.

For IPv4 the checksum is computed over:

```
 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Source Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

For IPv6 the checksum is computed over:

```
 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

9.2.2. Receiver operation

1) Validate the UDP checksum is correct

2) Compute the checksum over the IP addresses in the received packet

3) Subtract the resultant from the checksum value in the NAC option. If the difference is non-zero then NAT has changed the addresses

4) When processing a transport layer containing a checksum affected by NAT on the IP addresses, add the difference into the checksum calculation when verifying the packet.

In pseudo codes this is:

```
actual = checksum(IP addresses)
diff = actual - NAC_value
verify = checksum(transport packet) + checksum(pseudo header)
         + diff

if (verify == 0)
  packet is good
```

10. Alternative checksum option

The alternative checksum option contains a check over the GUE header and optionally all or part of the GUE payload. The algorithm used is CRC-32C which is the same as that used by iSCSI and SCTP. The alternative checksum can provide stronger protection against data corruption than that provided by the one's complement checksum used in the UDP checksum or the GUE checksum (section 8).

10.1. Extension field format

The presence of the GUE alternate checksum option is indicated by the A bit in the GUE header.

The format of alternate checksum field 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 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Reserved            |        Payload coverage       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              CRC                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- CRC: Computed CRC value. This CRC covers the GUE header (including fields and private data covered by Hlen) and optionally all or part of the payload (encapsulated packet).

- Payload coverage: Number of bytes of payload to cover in the CRC calculation. Zero indicates that the checksum only covers the GUE header. If the value is greater than the encapsulated payload length, the packet MUST be dropped.

10.2. Usage

The 32-bit alternative checksum does not include a pseudo header containing IP addresses or ports.

CRC-32C is calculated using the 0x1EDC6F41 polynomial:

x^32 + x^28 + x^27 + x^26 + x^25 + x^23 + x^22 + x^20 + x^19 +
x^18 + x^14 +x^13 + +x^11 + x^10 +x^9 +x^8 + x + 1

The procedure for setting the GUE alternative checksum on transmit is:

1) Create the GUE header including the alternative checksum field. The CRC field is initialized to zero.

2) Calculate the CRC using the CRC-32C algorithm from the start of the GUE header through the its length as indicated in GUE Hlen.

3) Continue the calculation to cover the payload portion if payload coverage is enabled (payload coverage field is non-zero). If the length of the payload coverage is not aligned to four bytes, logically append zero bytes up to the necessary alignment for the purposes of CRC calculation.

4) Set the resultant value in the CRC field.

10.2.2. Receiver operation

If the GUE alternative checksum option is present, the receiver MUST validate the checksum before processing any other fields, except the GUE checksum, or accepting the packet.

The procedure for verifying the alternate checksum is:

1) If the payload coverage length is greater than the length of the encapsulated payload then drop the packet.

2) Note value in the CRC field and set the CRC field to zero.

3) Calculate the CRC using the CRC-32C algorithm from the start of the GUE header through the its length as indicated in GUE Hlen.

4) Continue the calculation to cover the payload portion if payload coverage is enabled (payload coverage field is non-zero). If the length of the payload coverage is not aligned to four bytes, logically append zero bytes up to the necessary alignment for the purposes of CRC calculation.

5) Compare the computed value with the original value of the CRC field. If they are equal then that packet is accepted, if they are not equal then verification failed and the packet MUST be

dropped.

6) Restore the CRC field to its original value.

10.3. Corrupted alternate checksum flag

Similar to the GUE checksum, the GUE alternate checksum does not protect against the alternative checksum flag (A flag) being corrupted. If an encapsulator sets the alternative checksum flags and option but the A bit flips to be zero, then a decapsulator will incorrectly process that packet as not having an alternate checksum field.

To mitigate this issue an encapsulator and depcapsulator might agree that an alternate checksum is always required. This agreement could be established by configuration or GUE capability negotiation.

10.4. Security Considerations

The alternate checksum option is only a mechanism for corruption detection, it is not a security mechanism. To provide integrity checks or authentication of the GUE header, the GUE security option SHOULD be used.

11. Processing order of options

Options MUST be processed in a specific order for both transmission and reception. Note that some options, such as the checksum option, depend on other fields in the GUE header to be initialized.

11.1. Processing order when sending

When setting the security option (HMAC option in particular), the checksum option, or the alternate checksum option-- all the GUE fields being used must be present and properly set in the header. The checksum value in the checksum option or alternate checksum option MUST be initialized to zero to ensure consistent HMAC and checksum calculation.

The order of processing options to send a GUE packet are:

1) Fragment if necessary and set fragmentation option. If the

group identifier is present it is copied into each fragment. If payload transformation will increase the size of the payload that MUST be accounted for when deciding how to fragment. Apply processing below for each fragment

2) Set group identifier option (to the same value for each fragment)

3) Perform payload transform (potentially on a fragment) and set payload transform option.

4) Set Remote checksum offload.

5) Set NAT address checksum option.

6) Set security option per cookie or HMAC calculation.

7) Calculate GUE alternate checksum and set the alternate checksum option.

8) Calculate GUE checksum and set checksum option.

11.2. Processing order when receiving

On reception the order of actions is:

1) Verify GUE checksum.

2) Verify alternate checksum. If the GUE checksum option is present, set its checksum fields to zero for computing the alternate checksum. After computation, restore the checksum value in the GUE checksum field.

3) Verify security option. If the GUE checksum option or alternate checksum option are also present and HMAC computation is being done over the GUE header, then set the checksum fields to zero for computing the HMAC. After computation, restore the checksum values.

4) Save the NAT address checksum value. It will be applied when processing the encapsulated packet.

5) Adjust packet for remote checksum offload.

6) Perform payload transformation (i.e. decrypt payload).

7) Perform reassembly.

8) Process packet (take group identifier into account if present).

The relative processing order between GUE extensions and private fields is unspecified in this specification. Any processing order requirements regarding private data must be agreed upon between an ecapsulator and decapsulator.

12. Security Considerations

Encapsulation of a network protocol in GUE should not increase security risk, nor provide additional security in itself. GUE requires that the source port for UDP packets SHOULD be randomly seeded to mitigate some possible denial service attacks.

If the integrity and privacy of data packets being transported through GUE is a concern, the GUE security option and payload encryption using the the transform option SHOULD be used to remove the concern. If the integrity is the only concern, the tunnel may consider use of GUE security only for optimization. Likewise, if privacy is the only concern, the tunnel may use a GUE transform for

encryption only.

If a GUE payload already provides secure mechanism, e.g., the payload is an IPsec packet, it is still valuable to consider use of GUE security.

GUE may rely on other secure tunnel mechanisms such as DTLS [RFC6347] over the whole UDP payload for securing the whole GUE packet or IPsec [RFC4301] to achieve the secure transport over an IP network or Internet.

IPsec [RFC4301] was designed as a network security mechanism, and therefore resides at the network layer. As such, if the tunnel is secured with IPsec, the UDP header would not be visible to intermediate routers in either IPsec tunnel or transport mode. This is a drawback since it prohibits intermediate routers to perform load balancing based on the flow entropy in UDP header. In addition, this method prohibits some middle box functions on the path.

By comparison, DTLS [RFC6347] was designed for application level security and can better preserve network and transport layer protocol information than IPsec [RFC4301]. Using DTLS over UDP to secure the GUE tunnel, both GUE header and payload will be encrypted. In order to differentiate plaintext GUE header from the encrypted GUE header, the destination port of the UDP header between two must be different, which essentially requires another standard UDP port for GUE with DTLS. The drawback on this method is to prevent a middle box operation to GUE tunnel on the path.

Use of two independent tunnel mechanisms such as GUE and DTLS over UDP to carry a network protocol over an IP network adds some overlap and complexity. For example, fragmentation will be done twice.

As the result, a GUE tunnel SHOULD use the security mechanisms specified in this document to provide secure transport over an IP network or Internet when it is needed. GUE encapsulation can be used as a secure transport mechanism over an IP network and Internet.

13. IANA Consideration

IANA is requested to create a "GUE flag-fields" registry to allocate flags and extension fields used with GUE. This shall be a registry of bit assignments for flags, length of extension fields for corresponding flags, and descriptive strings. There are sixteen bits for primary GUE header flags (bit number 0-15). New values are assigned in accordance with RFC Required policy [RFC5226]. New flags should be allocated from high to low order bit contiguously without holes. This document requests an initial assignment of flags in the

registry.

IANA is requested to assign flags for the extensions defined in this specification. Specifically, an assignment is requested for the Group Identfier, Security, Fragmentation, Payload Transform, Remote Checksum Offload, Checksum, NAT Checksum, and Alternate Checksum extensions in the "GUE flag-fields" registry.

Flags bits	Field size	Description	Reference
Bit 0	4 bytes	Group identifier	This document
Bit 1..3	001->8 bytes 010->16 bytes 011->32 bytes 100->40 bytes	Security	This document
Bit 4	8 bytes	Fragmen- tation	This document
Bit 5	4 bytes	Payload transform	This document
Bit 6	4 bytes	Remote checksum offload	This document
Bit 7	4 bytes	Checksum	This document
Bit 8	4 bytes	NAT checksum address	This document
Bit 9	4 bytes	Alternate checksum	This document
Bit 10..15		Unassigned	

IANA is requested to set up a registry for the GUE payload transform types. Payload transform types are 8 bit values. New values for control types 1-127 are assigned via Standards Action [RFC5226].

+----------------+-----------------+---------------+

Transform type	Description	Reference
0	Reserved	This document
1	DTLS	This document
2..127	Unassigned	
128..255	User defined	This document

14. References

14.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC6347] Rescoria, E., Modadugu, N., "Datagram Transport Layer Security Version 1.2", RFC6347, 2012.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[I.D.ietf-gue] T. Herbert, L. Yong, and O. Zia, "Generic UDP Encapsulation" draft-ietf-intarea-gue-01

[FIPS180-4] Secure Hash Standard (SHS), Nation Institute of Standards and Technology, 8/2015

14.2. Informative References

[RFC3931] Lau, J., Townsley, W., et al, "Layer Two Tunneling Protocol

- Version 3 (L2TPv3)", RFC3931, 1999

[RFC2104] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol" , RFC 2401, DOI 10.17487/RFC2401, November 1998, <http://www.rfc-editor.org/info/rfc2401>.

[RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation" , RFC 6407, DOI 10.17487/RFC6407, October 2011, <http://www.rfc-editor.org/info/rfc6407>.

[RFC4459] Savola, ., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, DOI 10.17487/RFC4459, April 2006, <http://www.rfc-editor.org/info/rfc4459>.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <http://www.rfc-editor.org/info/rfc4963>.

[RFC2764] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, A. Malis, "A Framework for IP Based Virtual Private Networks", RFC2764, February 2000.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, October 1995.

[RFC3128] Miller, I., "Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)", RFC 3128, June 2001.

[RFC1071] Braden, R., Borman, D., and C. Partridge, "Computing the Internet checksum", RFC1071, September 1988.

[I.D.hy-nvo3-gue-4-nvo] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Network Virtualization Overlay" draft-hy-nvo3-gue-4-nvo-03

[I.D.previdi-6man-sr-header] Previdi S. et al, "IPv6 Segment Routing Header (SRH) draft-ietf-6man-segment-routing-header-02

[I.D.templin-aerolink] F. Templin, "Transmission of IP Packets over AERO Links" draft-templin-aerolink-62

[UDPENCAP] T. Herbert, "UDP Encapsulation in Linux",

http://people.netfilter.org/pablo/netdev0.1/papers/UDP-Encapsulation-in-Linux.pdf

Authors' Addresses

Tom Herbert
Quantonium
4701 Patrick Henry Dr.
Santa Clara, CA
USA

EMail: tom@herbertland.com

Lucy Yong
Austin, TX
USA

Email: lucy.yong@huawei.com

Fred L. Templin
Boeing Research & Technology
P.O. Box 3707
Seattle, WA 98124
USA

Email: fltemplin@acm.org