Internet-Draft
Intended status: Standard track
Expires March 31, 2019

T. Herbert
Quantonium

September 27, 2018

Generic TCP Encapsulation
draft-herbert-tsvwg-gte-00

Status of this Memo

Copyright Notice

Abstract

This specification describes Generic TCP Encapsulation (GTE) which is a method to encapsulate packets of different IP protocols within TCP data streams. Encapsulating packets in TCP facilitates communications across networks that block or sub-optimally handle non-TCP traffic. GTE is an adaptation of Generic UDP Encapsulation (GUE) to work in the context of TCP. GTE employs the GUE encapsulation format and optional extensions.

Table of Contents

1 Introduction

This specification describes Generic TCP Encapsulation (GTE) which is a general method for encapsulating packets of arbitrary IP protocols within Transport Control Protocol (TCP) [RFC0793] streams. The motivation and basic requirements for encapsulating packets with TCP are described in [TCPENCAP]. The primary motivation is to tunnel packets over networks that either block or treat non-TCP traffic sub-optimally. For instance, a real-time gaming application that uses UDP may chose to encapsulate packets in TCP in order to traverse a stateful firewall that only allows TCP.

GTE is an adaptation of Generic UDP Encapsulation [GUE] to encapsulate packets in a TCP stream. Packets are encapsulated in GTE messages. Each GTE message is composed of a message length, a GUE header variant, and a GUE payload. The message length is needed to demarcate messages in the TCP stream. The TCP data stream is then composed of a sequence of GTE messages. The GUE header and protocol in GTE messages are the same as that defined in GUE. Sender and receiver processing is modified accordingly to take into account the differences between TCP and UDP encapsulation. GTE defines an optimization to compress out the GUE header in certain circumstances.

1.1 Requirements Language

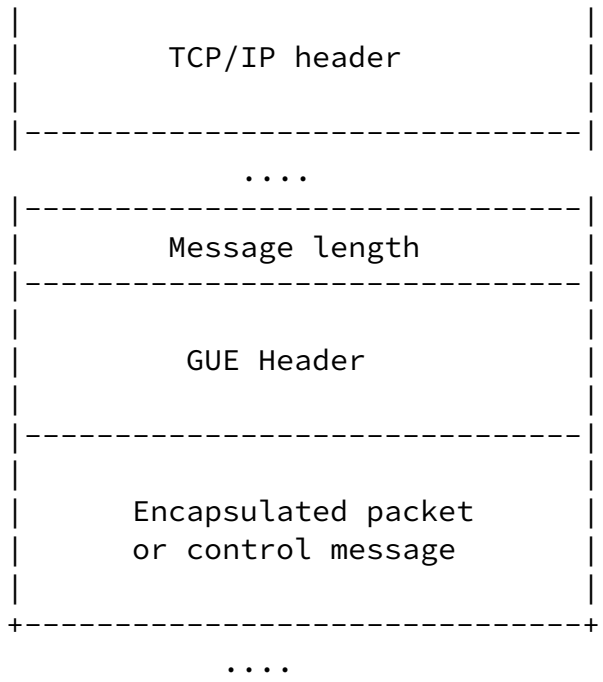
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2 Encapsulation format

This section describes the encapsulation formats of Generic TCP Encapsulation.

2.1 GTE messages in a TCP stream

A GTE message is encapsulated in a TCP stream. A message is comprised of a message length field, a GUE header of one of the GUE variants, and a payload which is either an encapsulated packet of some IP protocol or a control message such as an OAM (Operations, Administration, and Management) message. Encapsulation of GTE message in TCP has the general format:

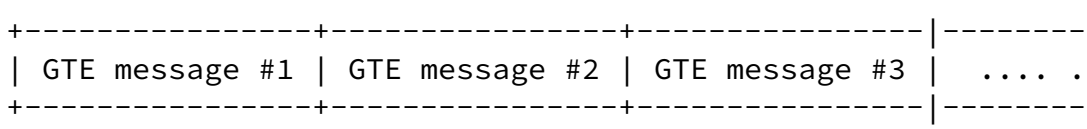


A TCP stream is composed of a sequence of GUE messages. Each message is prefixed by the length of the message for demarcation. Note that there is no on-to-one correspondence between a GTE message and a TCP segment. Multiple GTE messages may be in a single TCP segment, and a single GTE message may span multiple TCP segments. Additionally, there is no assumed alignment of GTE messages within a stream

The GUE header and encapsulation are defined in [GUE]. A GTE message may carry variant 0 or variant 1 GUE packets. The formats are detailed below.

2.2 TCP connections

A GTE stream is composed of GTE message. The message are sequential and per the semantics of TCP they are processed in order.



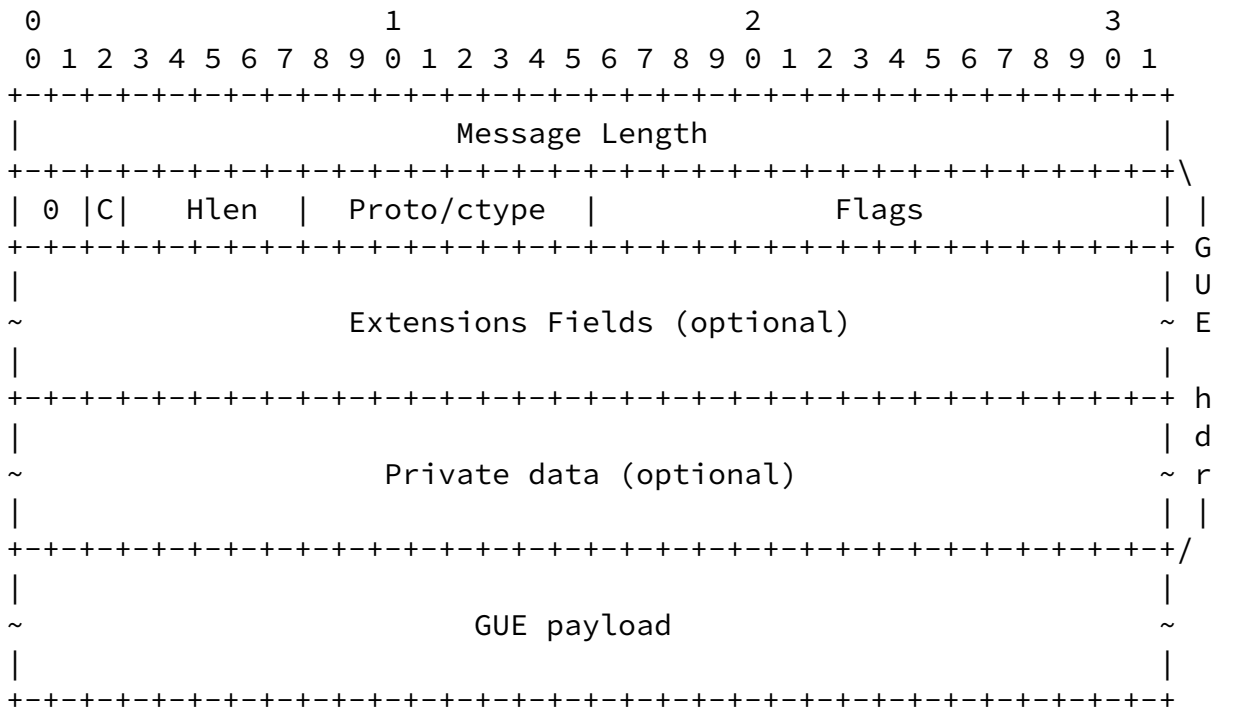
A default TCP port number (suggesting 6080) will be assigned to GTE. A server may listen on this port number for incoming connections. When a connection is established, the data stream is interpreted as a sequence of GTE messages, so the first four bytes in the stream are the length of the first GTE message.

GTE could also use an HTTP connection for the transport. This would be done using the HTTP Upgrade header to specify use of GTE. Specification of this is outside the scope of this document.

Transport Layer Security (TLS) can be used on the TCP stream. In this case, the GTE messages are the plain text before TLS is applied. GTE messages are processed on receive after TLS processing.

2.3 Encapsulation with GUE variant 0

Variant 0 of GUE defines a generic extensible format to encapsulate packets by Internet protocol number. The format of a GTE message with GUE variant 0 is:



Fields:

- Message Length: Length of the GTE message not including the four bytes of this field. The length is the sum of the GUE header length and the length of the encapsulated GUE payload. The size of the message length field may be changed using the Message Length Size control message (section 4.1).

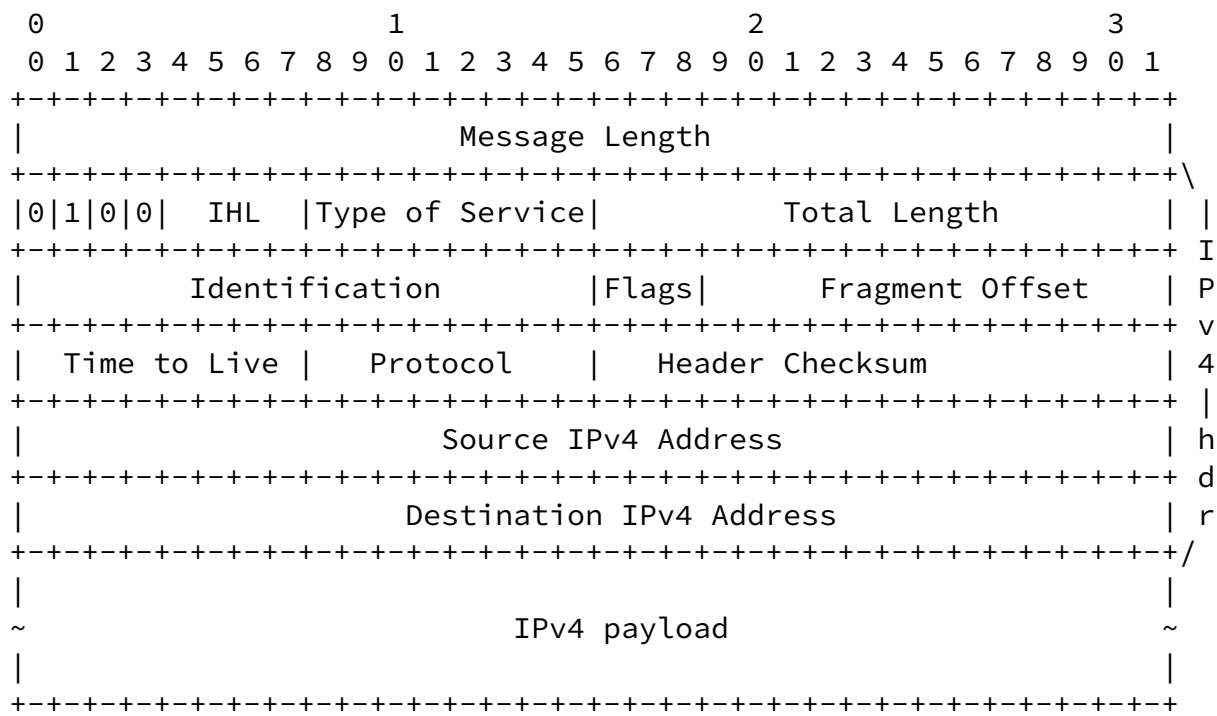
The GUE header fields are specified in [GUE] and GUE extensions are describe in [GUEEXTEN].

2.4 Encapsulation with GUE variant 1

Variant 1 of GUE allows direct encapsulation of IPv4 and IPv6 packets in GUE.

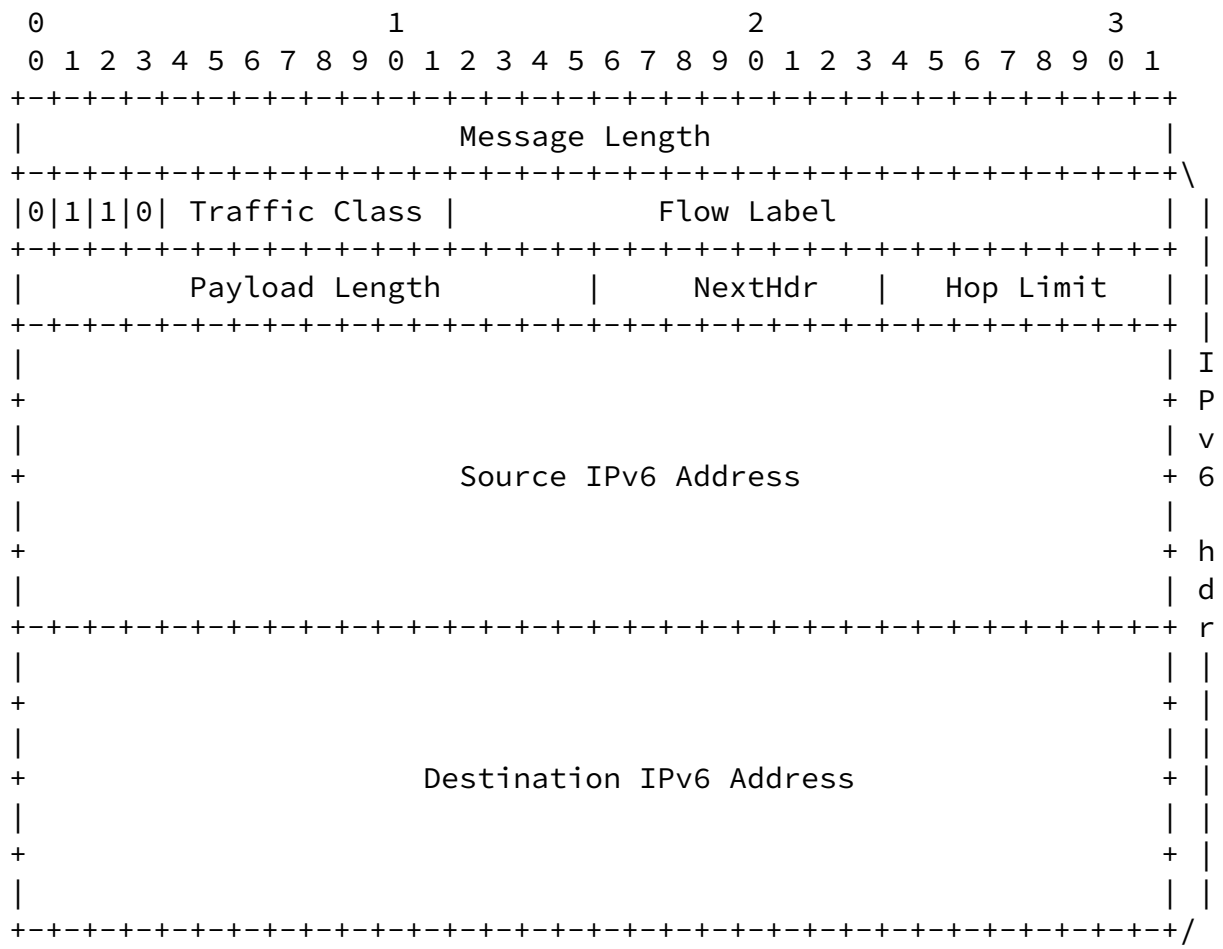
2.4.1 IPv4 direct encapsulation

A GTE message with direct encapsulation of an IPv4 packet has the format:



2.4.2 IPv6 direct encapsulation

A GTE message with direct encapsulation of an IPv6 packet has the format:



3 Operation

This section describes the operation of GTE.

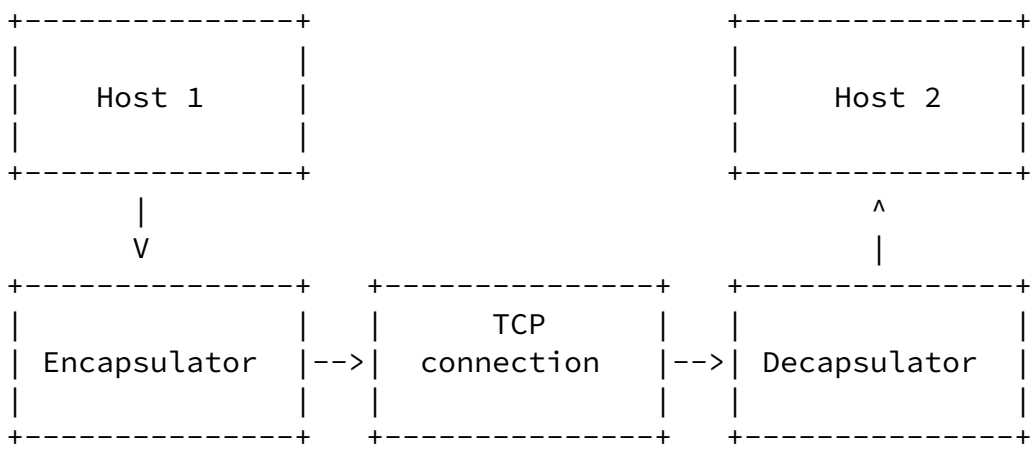
3.1 Encapsulation flavors

GTE can do either network layer of transport layer encapsulation.

3.1.1 Network layer encapsulation

Network tunneling can be achieved by encapsulating layer 2 or layer 3 packets. In this case the encapsulator and decapsulator nodes are the tunnel endpoints as well as the endpoints of the GTE TCP connection. These could be routers providing tunnels on behalf of communicating hosts.

The figure below illustrates the use of GTE network layer encapsulation between two hosts.



Host 1 is sending packets to Host 2. An encapsulator performs encapsulation of packets from Host 1. These encapsulated packets traverse the network in a TCP stream. At the decapsulator, packets are decapsulated and sent on to Host 2. GTE encapsulation is not required the reverse direction.

3.1.2 Transport layer encapsulation

GTE can encapsulate transport layer packets such as UDP or TCP. The encapsulated packets do not include their own IP header, instead the IP header is inferred from the TCP connection.

The figure below illustrates the use of GTE transport layer encapsulation between two hosts.

When encapsulating transport layer packets, the encapsulator and decapsulator should be co-resident with the hosts. The encapsulated

transport layer packets are in a TCP stream. The corresponding IP addresses of the endpoints of the TCP connection are taken to be the endpoints of the encapsulated transport packet. Note that the encapsulated transport layer packet is independent of the encapsulating TCP headers, in particular, port numbers of the outer TCP headers and encapsulated transport headers are independent.

3.2 Encapsulator operation

Encapsulators create GTE data messages, set flags and optional extension fields in the GUE header, and forward packets to a decapsulator by writing messages on a TCP data stream.

An encapsulator can be an end host originating the packets of a flow, or can be a network device performing encapsulation on behalf of hosts (routers implementing tunnels for instance). In either case, the intended target (decapsulator) is the peer of the TCP connection.

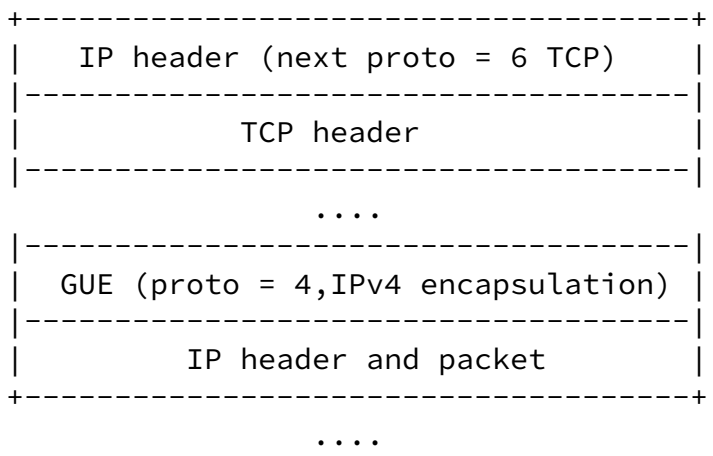
If an encapsulator is tunneling packets -- that is encapsulating packets of layer 2 or layer 3 protocols (e.g. EtherIP, IPIP, or ESP tunnel mode) -- the propagation of network layer information, such as ECN [RFC6040] and diff-serv [RFC2983] should be considered. Since there is no one-to-one mapping between encapsulated packets and the outer packet, the best way to map information may not be obvious. For instance, two IP packets encapsulated in the same TCP segment may have different diff-serv bits. An implementation MAY apply its own configurable heuristics for tunneling of one protocol over another.

3.3 Decapsulator operation

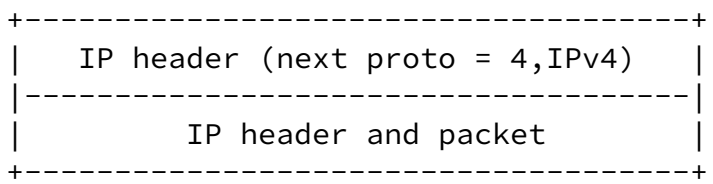
If a complete GTE data message is received on a TCP stream, the payload of the GTE message is logically extracted. An IP packet is created with an IP header that is fabricated from the TCP connection parameters. The next protocol in the IP header is set to the protocol from the proto field in the GUE header. The resulting packet is then resubmitted into the protocol stack to process as though it was received with the protocol in the GUE header.

3.3.1 Receiving network layer encapsulation

Consider that a data message is received where GTE encapsulated an IP packet. In this case, the proto field in the GUE header is set to 4.



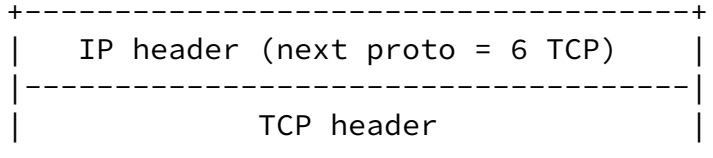
The receiver extracts the IP header and payload, and encapsulates that in an IP packet using IPIP encapsulation. The IP header is derived from the TCP connection or IP header that was present in a received TCP segment. The next protocol field is set to 4. The resultant packet would have the format:

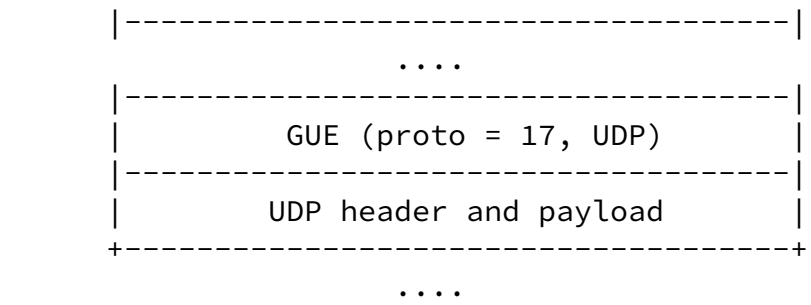


This packet is then resubmitted into the protocol stack to be processed as an IPIP packet.

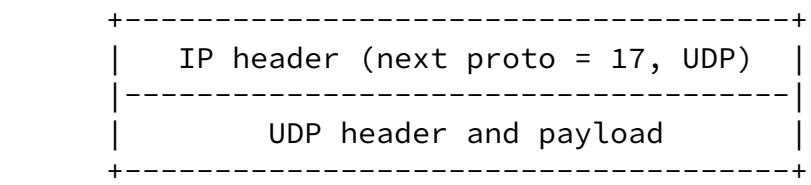
3.3.2 Receiving transport layer encapsulation

Consider that a data message is received where GTE encapsulates a UDP packet. In this case, the proto field in the GUE header is set to 17 for UDP:





The receiver extracts the UDP header and payload, and formats those in a corresponding UDP/IP packet. The IP header is derived from the TCP connection or IP header that was present in a received TCP segment. The next protocol field is set to 17. The resultant packet would have the format:



This packet is then resubmitted into the protocol stack to be processed as a UDP packet.

3.3.3 Error handling

If a receiver encounters an error in processing the GUE header it SHOULD close the GTE connection and MAY log an error. The possible errors in processing a GUE header are described in [GUE].

3.4 Processing a received control message

If a valid GUE control message is received, the packet MUST be processed as a control message. The specific processing to be performed depends on the ctype in the GUE header. See [GUE].

3.5 NAT interaction

Transport layer encapsulation (section 3.1.2) allows TCP and UDP packets (without IP headers) to be directly encapsulated in GTE. The IP headers for these are derived from those used in the TCP connection. In particular, the pseudo header for an encapsulated

transport checksum might cover the IP addresses used in outer TCP packets. When a packet traverses a NAT and the addresses of the TCP packet changes, the checksum for an encapsulated TCP or UDP packet becomes incorrect.

To resolve this problem, the NAT Address Checksum option can be used [GUEEXEN]. A sender computes the one's complement checksum over the IP addresses for the outer TCP connection and sets the value in the Checksum field of the GUE NAT Address Checksum option. When a receiver processes the GTE message, it can compute the required checksum adjustment by taking the difference between checksum over the IP addresses for the connection that it sees and the value it received in the option. If the adjustment is non-zero then that can be added to the computed packet checksum in verification. The exact algorithm is described in [GUEEXTEN].

3.6 Checksum offload of encapsulated transport checksum

Checksum offload is a common technique in Network Interface Cards (NICS) to improve performance [UDPENCAP]. Checksum offload is done on a per packet basis and typically can offload only one checksum in a packet. Several techniques have been implemented to make general use of the capability to effectively offload checksum computation for any number of encapsulated transport checksums in a packet. These techniques include "checksum-unnecessary conversion", Local Checksum Offload (LCO), and Remote Checksum Offload (RCO).

In the case of GTE, packets are encapsulated in a stream so the aforementioned techniques for checksum offload don't directly apply and must be adapted.

3.6.1 Transmit checksum offload

To offload an encapsulated transport layer checksum for transmission, the Remote Checksum Offload (RCO) Option is used [GUEXTEN]. The Checksum Start and Checksum Offset fields in the option are set to point to the starting byte for checksum calculation and the offset where the checksum is to be written. When a receiver processes this field, it will write the derived checksum value at the indicated offset (i.e. the checksum field of an encapsulated transport protocol).

3.6.2 Receive checksum offload

To offload a transport checksum calculation in a received GTE packet, the fact that TCP packets must always include a checksum can be leveraged. Given the TCP payload (or payloads) containing a GTE message, the checksum over the GTE message can be derived by subtracting out the checksum for portions of the payload that aren't part of the GTE message. Once the checksum over the GTE message is determined, that can be used to verify any encapsulated transport layer checksums in the message. Note that this is most efficient when a TCP segment contains only one (or part of one) GTE message.

3.7 GUE extensions

There is no prohibition to using any of the GUE extensions [GUEEXTENS] in GTE. Their usefulness in the context of a TCP stream may vary. The use of NAT Address Checksum and Remote Checksum Offload options are described above. The Fragmentation option is not very useful since TCP provides segmentation and reassembly. Similarly, the GUE checksum option is not useful since the whole message is already covered by the TCP checksum. The Security, Payload Transform, and Alternate Checksum options may be useful to provide security or strong data integrity checks. The Group Identifier might be useful in some cases.

3.8 Surplus area

A an encapsulated packet in GTE may have its own length field such that the encapsulated packet does not take up all the space of the GTE message as indicate by GTE message length. Any bytes beyond the encapsulated packet to the end of message are called "surplus area". Surplus area, for those protocols that have a separate length field, is considered to be reserved. [UDPOPTIONS] is a specification for placing UDP options in surplus area.

4 GTE control messages

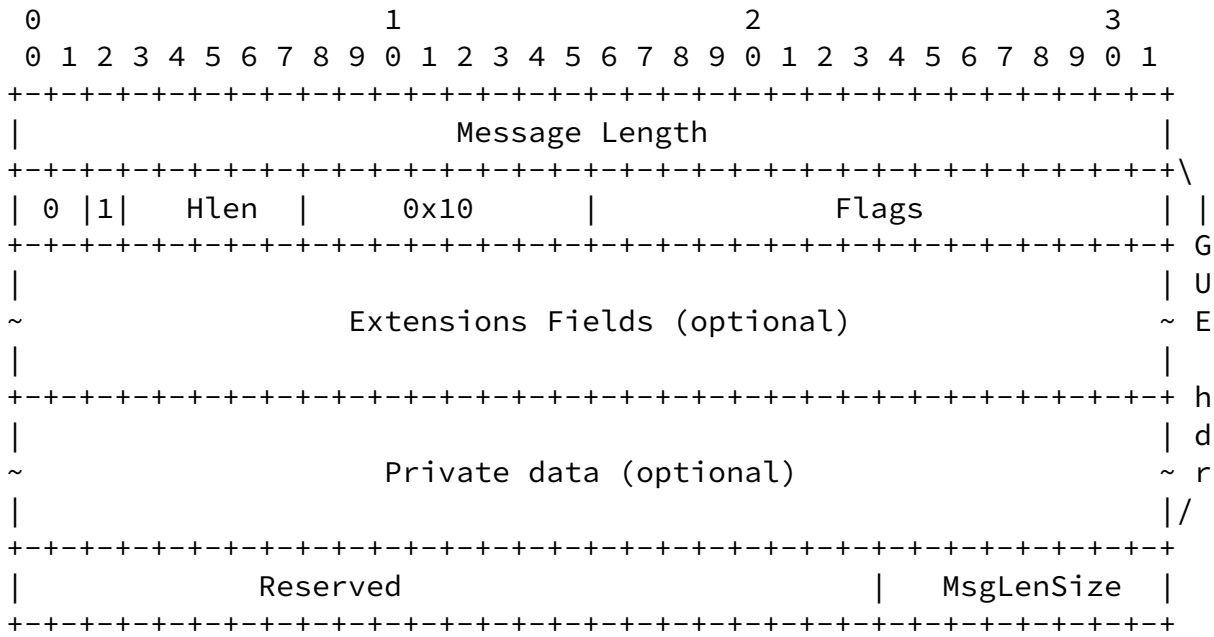
Two GUE control messages are defined to set parameters on a GTE connection. One control message sets the size of the message length

field, and the other sets a GUE header template for compressing out the GUE header in GTE messages.

4.1 Message length size

The default message length field in GTE is four bytes. The GUE "Message length size" control message allows the size to be changed for a GTE connection.

The format of the message length size control message is:



Pertinent fields are:

- GUE C bit: Set to 1 to indicate control message
- GUE Proto/ctype field: Set to 0x10 to indicate a message length size control message
- Reserved: Set to zero on transmit, ignored on receive
- MsgLenSize: Number of byte for message length size. Valid values are 1,2,3, or 4

Note that the Reserved and MsgLenSize fields are in the payload of the GUE message. These fields are not included in the GUE header length.

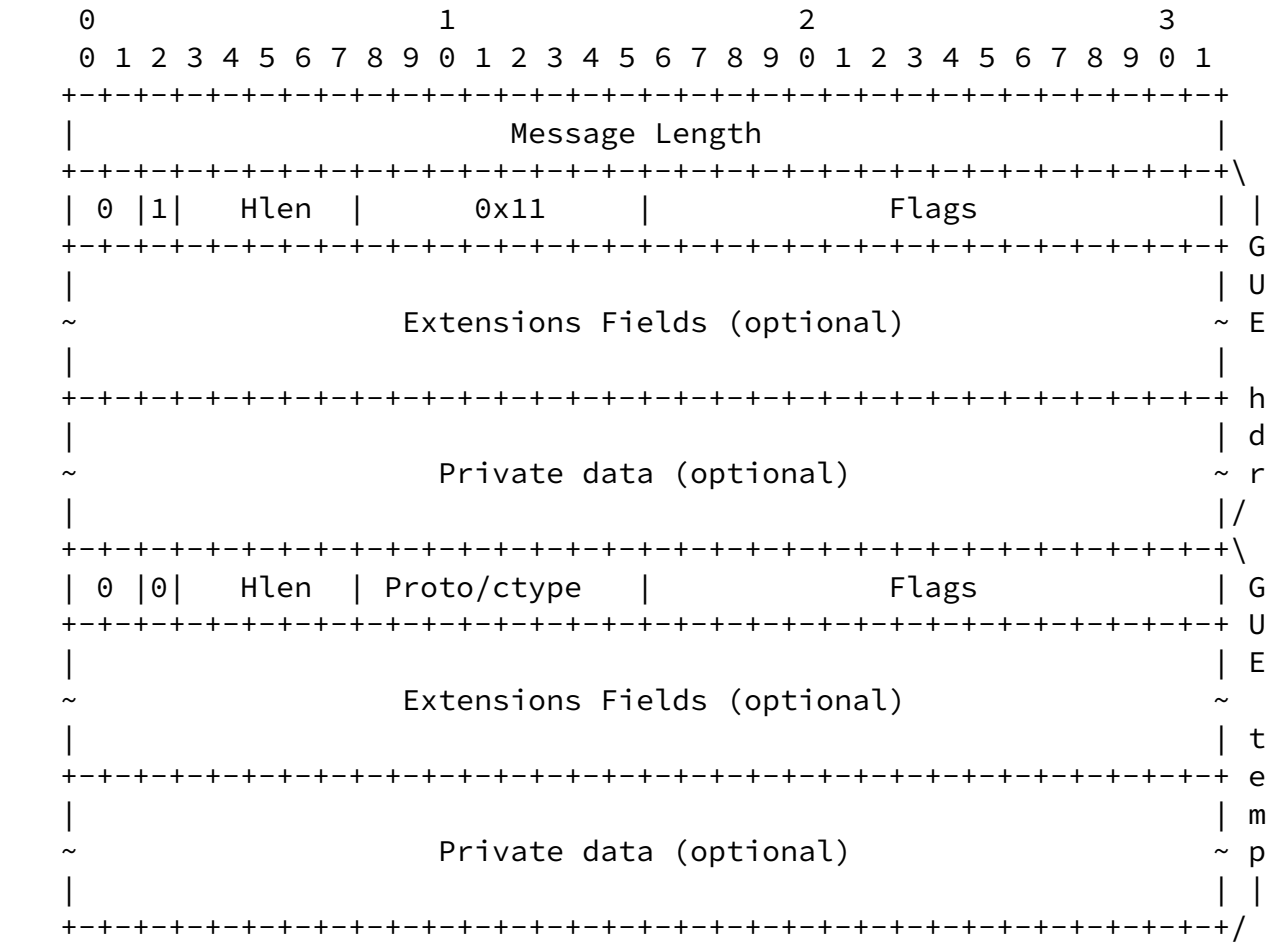
If a received message is too short to include the MsgLenSize field or the MsgLenSize is zero or a value greater then 4, then the connection MUST be closed and an error MAY be logged.

After a message length size control message is received, the size of the message length field in subsequent messages is taken to be the provided value. The message length size control message MAY be sent multiple times to use different sizes for lifetime of the connection.

4.2 GUE header template

An application may use GTE to always tunnel one specific protocol over GTE where the GUE header is identical in all GTE messages. The "GUE header template" control message is defined to compress out the GUE header in such a case. The payload of this control message indicates the GUE header to be applied to all subsequent messages.

The format of the GUE header template control message is:



Pertinent fields in the (outer) GUE header are:

- o Variant: Set to 0
- o C bit: Set to 1 to indicate control message
- o Proto/ctype field: Set to 0x11 to indicate a header template control message

Pertinent fields in the GUE header template GUE are:

- o Variant: Must be set to 0
- o C bit: Should be zero to indicate a data message
- o Proto/ctype: Set to the common IP protocol number for all the messages in the GTE stream

The GUE header template may have flags set, extension fields present, as well as private data.

4.2.1 Header template validation

The format of received GUE template header should be validated like a normal GUE header as described in [GUE]. If the header template is determined to be malformed, then the connection SHOULD be dropped and an error MAY be logged.

The GUE header template must be variant 0. If a GUE header template is received with a variant that is another value the connection MUST be closed and an error MAY be logged.

4.2.2 Optional extensions in header templates

Any GUE extension that may be set in a header template will need to be applicable to all messages.

The Checksum and Alternate Checksum options are not useful in a GUE header template since their input includes payload data. If a proposed header template contains such options the connection SHOULD

be dropped and an error MAY be logged.

The Fragmentation option is nonsensical to be in a GUE header template. If a proposed header template contains the Fragmentation option then the connection SHOULD be dropped and an error MAY be logged.

The Security option may be useful in a GUE header template if it does

not include any payload data as input. If the Security option in a header template requires payload data as input, then the connection SHOULD be dropped and an error MAY be logged. If the Security option in a header template doesn't require payload data as input, then it SHOULD be verified (in this case all the input should be contained within the GUE header template). If the Security option fails verification, then the connection SHOULD be dropped and an error MAY be logged.

The Group Identifier, Remote Checksum Offload, NAT Address Checksum and Payload Transform options are valid optional extensions to use in a GUE header template.

4.2.3 Processing received messages

After a GUE header template control message is received, the header template is applied to all subsequent GTE messages. GTE messages will contain a message length followed by the GUE payload. For each received message, the GUE header from the saved template for the connection is logically inserted into the message, and the message is processed as though the GUE header was received.

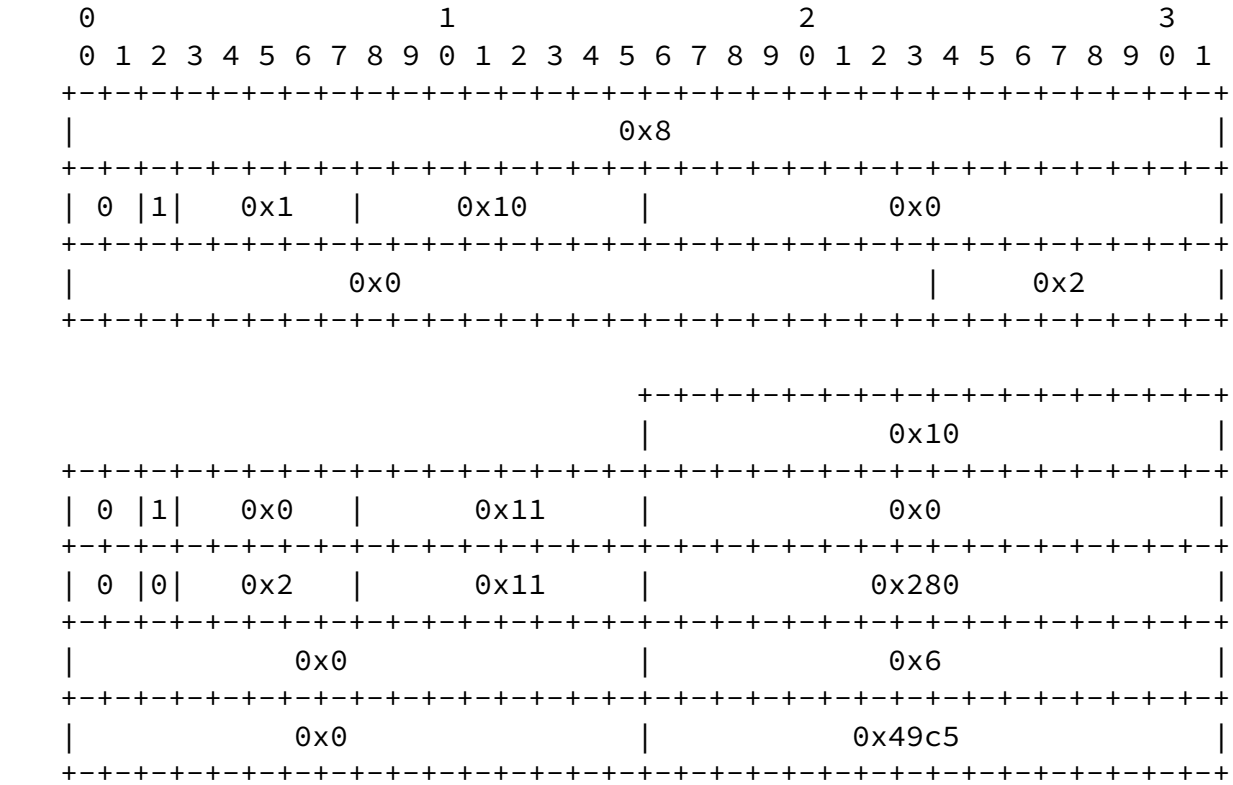
The GUE header template control message can only be sent once on a connection. All messages following the control message are considered to have the same GUE header and there is no way to send any more GUE control messages on the stream. If both the message length size and the GUE header template are to be set for a GTE connection, then the message length size control message MUST be sent first.

4.3 Example of UDP over GTE

This section provides an example use of the GTE control message to encapsulate a stream of UDP packets over a GTE connection. In this

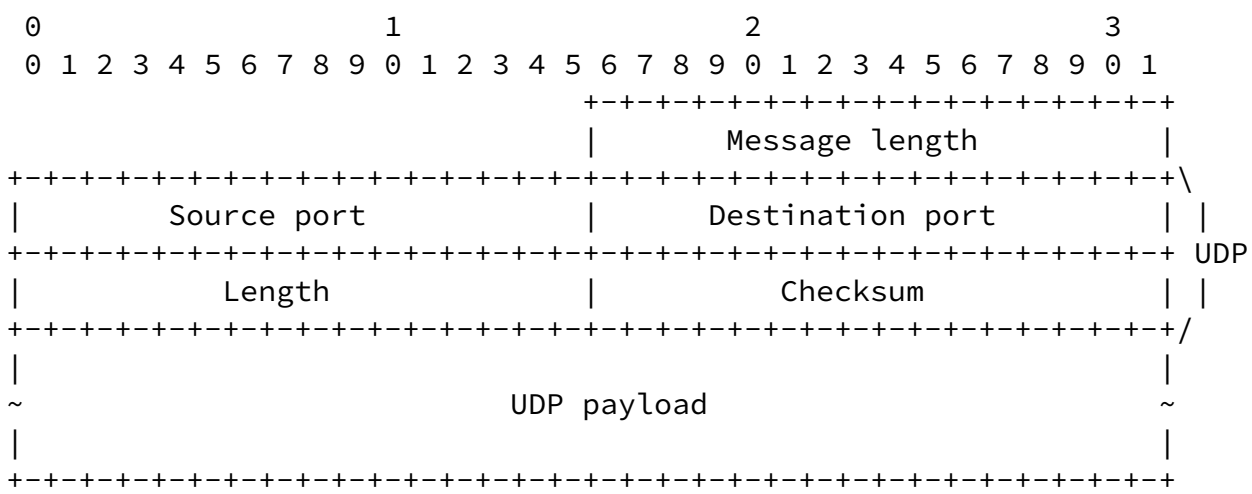
example, the sender uses both the message length size and GUE header template control messages. The message length size is set to two bytes to accommodate the largest UDP packet (65535 bytes). The GUE header template sets the protocol for all messages to be UDP, enables Remote Checksum Offload, and enables the NAT Address Checksum option. In this example, the checksum start and checksum offset field in the Remote Checksum Offload option take values of 0 and 6. The addresses of the connection in this example are 2001:DB8::c099:1:2:0 and 2001:DB8::a92a:7:8:1-- the one's complement sum over those addresses is 0x49c5.

The message length size and the GUE header template control messages would be as follows:



All the messages following these control messages are encapsulated

UDP packets in the following format:



Note that for each message, a full UDP/IP packet can be created by deriving an IP header from the TCP connection (the address endpoints of the TCP connection are the same the addresses of the UDP packet). The NAT Address Checksum option can be used to insure that the UDP checksum remains correct if the TCP connection goes through a NAT.

5 Security Considerations

GUE security mechanisms are applicable in GTE. Security considerations for TCP encapsulation are discussed in [TCPENCAP].

6 IANA Considerations

6.1 TCP port number

IANA is requested to assign one TCP port number for Generic TCP Encapsulation. The suggested number is 6080 which is the same as the UDP port number assigned for Generic UDP Encapsulation.

6.2 GUE control types

IANA is requested to assign two values in the registry for the GUE control types:

Control type	Description	Reference
0x10	GTE message length size	This document
0x11	GUE header template	This document

6 Acknowledgements

7 References

7.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

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

7.2. Informative References

[RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <http://www.rfc-editor.org/info/rfc2983>.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <http://www.rfc-editor.org/info/rfc6040>.

[TCPENCAP] T. Pauly and E. Kinnear, "TCP Encapsulation Considerations", draft-pauly-tcp-encapsulation-00

[GUEEXTEN] Herbert, T., Yong, L., and Templin, F., "Extensions for Generic UDP Encapsulation" draft-ietf-intarea-gue-extensions-05

[UDPENCAP] T. Herbert, "UDP Encapsulation in Linux", http://people.netfilter.org/pablo/netdev0.1/papers/UDP-Encapsulation-in-Linux.pdf

[LCO] Cree, E., https://www.kernel.org/doc/Documentation/networking/checksum-offloads.txt

Author's Address

Tom Herbert
Quantonium
4701 Patrick Henry
Santa Clara, CA 95054
US

Email: tom@herbertland.com