

Internet Area WG
Internet-Draft
Intended status: Standard track
Expires September 8, 2019

T. Herbert
Quantonium
L. Yong
Independent
O. Zia
Microsoft
March 7, 2019

Generic UDP Encapsulation
draft-ietf-intarea-gue-07

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

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

This Internet-Draft will expire on September 8, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

Abstract

This specification describes Generic UDP Encapsulation (GUE), which is a scheme for using UDP to encapsulate packets of different IP protocols for transport across layer 3 networks. By encapsulating packets in UDP, specialized capabilities in networking hardware for efficient handling of UDP packets can be leveraged. GUE specifies basic encapsulation methods upon which higher level constructs, such as tunnels and overlay networks for network virtualization, can be constructed. GUE is extensible by allowing optional data fields as part of the encapsulation, and is generic in that it can encapsulate packets of various IP protocols.

Table of Contents

1. Introduction	5
1.1. Applicability	5
1.2. Terminology and acronyms	6
1.3. Requirements Language	7
2. Base packet format	8
2.1. GUE variant	8
3. Variant 0	9
3.1. Header format	9
3.2. Proto/ctype field	10
3.2.1. Proto field	10
3.2.2. Ctype field	11
3.3. Flags and extension fields	11
3.3.1. Requirements	11
3.3.2. Example GUE header with extension fields	12
3.4. Private data	13
3.5. Message types	13
3.5.1. Control messages	13
3.5.2. Data messages	14
4. Variant 1	14
4.1. Direct encapsulation of IPv4	15
4.2. Direct encapsulation of IPv6	16
5. Operation	17
5.1. Network tunnel encapsulation	17
5.2. Transport layer encapsulation	17
5.3. Encapsulator operation	18
5.4. Decapsulator operation	18
5.4.1. Processing a received data message	18
5.4.2. Processing a received control message	19
5.5. Middlebox inspection	19
5.6. Router and switch operation	20
5.6.1. Connection semantics	20
5.6.2. NAT	21
5.7. Checksum Handling	21

5.7.1. Requirements	21
5.7.2. UDP Checksum with IPv4	21
5.7.3. UDP Checksum with IPv6	22
5.8. MTU and fragmentation	22
5.9. Congestion control	23
5.10. Multicast	23
5.11. Flow entropy for ECMP	23
5.11.1. Flow classification	24
5.11.2. Flow entropy properties	24
5.12. Negotiation of acceptable flags and extension fields	25
6. Motivation for GUE	26
6.1. Benefits of GUE	26
6.2. Comparison of GUE to other encapsulations	26
7. Security Considerations	28
8. IANA Considerations	28
8.1. UDP source port	28
8.2. GUE variant number	29
8.3. Control types	29
9. Acknowledgements	29
10. References	30
10.1. Normative References	30
10.2. Informative References	31
Appendix A: NIC processing for GUE	34
A.1. Receive multi-queue	34
A.2. Checksum offload	34
A.2.1. Transmit checksum offload	35
A.2.2. Receive checksum offload	35
A.3. Transmit Segmentation Offload	36
A.4. Large Receive Offload	37
Appendix B: Implementation considerations	37
B.1. Privileged ports	37
B.2. Setting flow entropy as a route selector	38
B.3. Hardware protocol implementation considerations	38
Authors' Addresses	39

1. Introduction

This specification describes Generic UDP Encapsulation (GUE) which is a general method for encapsulating packets of arbitrary IP protocols within User Datagram Protocol (UDP) [RFC0768] packets. Encapsulating packets in UDP facilitates efficient transport across networks. Networking devices widely provide protocol specific processing and optimizations for UDP (as well as TCP) packets. Packets for atypical IP protocols (those not usually parsed by networking hardware) can be encapsulated in UDP packets to maximize deliverability and to leverage flow specific mechanisms for routing and packet steering.

GUE provides an extensible header format for including optional data in the encapsulation header. This data potentially covers items such as a virtual networking identifier, security data for validating or authenticating the GUE header, congestion control data, etc. GUE also allows private optional data in the encapsulation header. This feature can be used by a site or implementation to define local custom optional data, and allows experimentation of options that may eventually become standard.

This document does not define any specific GUE extensions. [GUEEXTEN] specifies a set of initial extensions.

1.1. Applicability

GUE is a network encapsulation protocol that encapsulates packets for various IP protocols. Potential use cases include network tunneling, multi-tenant network virtualization, tunneling for mobility, and transport layer encapsulation. GUE is intended for deploying overlay networks in public or private data center environments, as well as providing a general tunneling mechanism usable in the Internet.

GUE is a UDP based encapsulation protocol transported over existing IPv4 and IPv6 networks. Hence, as a UDP based protocol, GUE adheres to the UDP usage guidelines as specified in [RFC8085]. Applicability of these guidelines are dependent on the underlay IP network and the nature of GUE payload protocol (for example TCP/IP or IP/Ethernet).

[RFC8085] outlines two applicability scenarios for UDP applications, 1) general Internet and 2) controlled environment. GUE is intended to allow deployment in both controlled environments and in the uncontrolled Internet. The requirements of [RFC8085] pertaining to deployment of a UDP encapsulation protocol in these environments are applicable. Section 5 provides the specifics for satisfying requirements of [RFC8085]. It is the responsibility of the operator deploying GUE to ensure that the necessary operational requirements are met for the environment in which GUE is being deployed.

GUE has much of the same applicability and benefits as GRE-in-UDP [RFC8086] that are afforded by UDP encapsulation protocols. GUE offers the possibility of good performance for load-balancing encapsulated IP traffic in transit networks using existing Equal-Cost Multipath (ECMP) mechanisms that use a hash of the five-tuple of source IP address, destination IP address, UDP/TCP source port, UDP/TCP destination port, and protocol number. Encapsulating packets in UDP enables use of the UDP source port to provide entropy to ECMP hashing.

In addition, GUE enables extending the use of atypical IP protocols (those other than TCP and UDP) across networks that might otherwise filter packets carrying those protocols. GUE may also be used with connection oriented UDP semantics in order to facilitate traversal through stateful firewalls and stateful NAT.

Additional motivation for the GUE protocol is provided in section 6.

1.2. Terminology and acronyms

GUE	Generic UDP Encapsulation
GUE Header	A variable length protocol header that is composed of a primary four byte header and zero or more four byte words of optional header data
GUE packet	A UDP/IP packet that contains a GUE header and GUE payload within the UDP payload
GUE variant	A version of the GUE protocol or an alternate form of a version
Encapsulator	A network node that encapsulates packets in GUE
Decapsulator	A network node that decapsulates and processes packets encapsulated in GUE
Data message	An encapsulated packet in a GUE payload that is addressed to the protocol stack for an associated protocol
Control message	A formatted message in the GUE payload that is implicitly addressed to the decapsulator to monitor or control the state or behavior of a tunnel
Flags	A set of bit flags in the primary GUE header

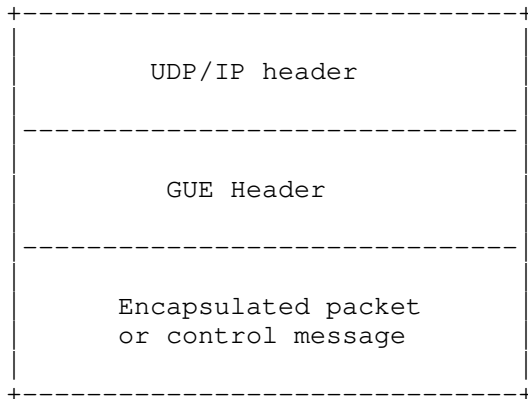
Extension field	An optional field in a GUE header whose presence is indicated by corresponding flag(s)
C-bit	A single bit flag in the primary GUE header that indicates whether the GUE packet contains a control message or data message
Hlen	A field in the primary GUE header that gives the length of the GUE header
Proto/ctype	A field in the GUE header that holds either the IP protocol number for a data message or a type for a control message
Private data	Optional data in the GUE header that can be used for private purposes
Outer IP header	Refers to the outer most IP header or packet when encapsulating a packet over IP
Inner IP header	Refers to an encapsulated IP header when an IP packet is encapsulated
Outer packet	Refers to an encapsulating packet
Inner packet	Refers to a packet that is encapsulated

1.3. 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. Base packet format

A GUE packet is comprised of a UDP packet whose payload is a GUE header followed by 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. A GUE packet has the general format:



The GUE header is variable length as determined by the presence of optional extension fields and private data.

2.1. GUE variant

The first two bits of the GUE header contain the GUE protocol variant number. The variant number can indicate the version of the GUE protocol as well as alternate forms of a version.

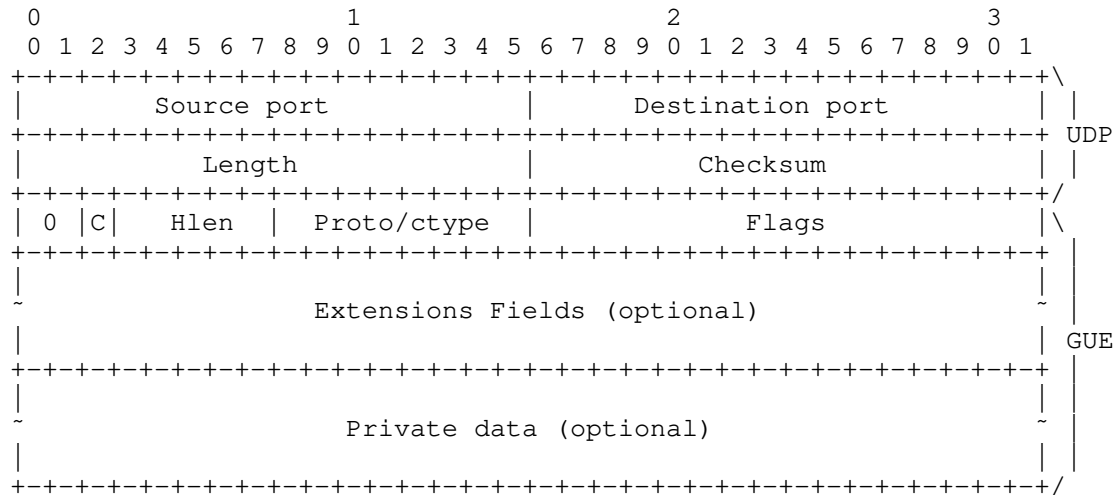
Variants 0 and 1 are described in this specification; variants 2 and 3 are reserved.

3. Variant 0

Variant 0 indicates version 0 of GUE. This variant defines a generic extensible format to encapsulate packets by Internet protocol number.

3.1. Header format

The header format for variant 0 of GUE in UDP is:



The contents of the UDP header are:

- o Source port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the local source port for the connection. When connection semantics are not applied, the source port is either set to a flow entropy value, as described in section 5.11, or is set to the GUE assigned port number, 6080.
- o Destination port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the destination port for the tuple. If connection semantics are not applied then the destination port is set to the GUE assigned port number, 6080.
- o Length: Canonical length of the UDP packet (length of UDP header and payload).
- o Checksum: Standard UDP checksum (handling is described in section 5.7).

The GUE header consists of:

- o Variant: 0 indicates GUE protocol version 0 with a header.
- o C: C-bit: When set indicates a control message. When not set indicates a data message.
- o Hlen: Length in 32-bit words of the GUE header, including optional extension fields but not the first four bytes of the header. Computed as $(\text{header_len} - 4) / 4$, where `header_len` is the total header length in bytes. All GUE headers are a multiple of four bytes in length. Maximum header length is 128 bytes.
- o Proto/ctype: When the C-bit is set, this field contains a control message type for the payload (section 3.2.2). When the C-bit is not set, the field holds the Internet protocol number for the encapsulated packet in the payload (section 3.2.1). The control message or encapsulated packet begins at the offset provided by Hlen.
- o Flags: Header flags that may be allocated for various purposes and may indicate the presence of extension fields. Undefined header flag bits MUST be set to zero on transmission.
- o Extension Fields: Optional fields whose presence is indicated by corresponding flags.
- o Private data: Optional private data block (see section 3.4). If the private block is present, it immediately follows that last extension field present in the header. The private block is considered to be part of the GUE header. The length of this data is determined by subtracting the starting offset of the private data from the header length.

3.2. Proto/ctype field

The proto/ctype fields either contains an Internet protocol number (when the C-bit is not set) or GUE control message type (when the C-bit is set).

3.2.1. Proto field

When the C-bit is not set, the proto/ctype field MUST contain an IANA Internet Protocol Number [IANA-PN]. The protocol number is interpreted relative to the IP protocol that encapsulates the UDP packet (i.e. protocol of the outer IP header). The protocol number serves as an indication of the type of the next protocol header which is contained in the GUE payload at the offset indicated in Hlen.

IP protocol number 59 ("No next header") can be set to indicate that the GUE payload does not begin with the header of an IP protocol. This would be the case, for instance, if the GUE payload were a fragment when performing GUE level fragmentation. The interpretation of the payload is performed through other means such as flags and extension fields, and nodes MUST NOT parse packets based on the IP protocol number in this case.

3.2.2. Ctype field

When the C-bit is set, the proto/ctype field MUST be set to a valid control message type. A value of zero indicates that the GUE payload requires further interpretation to deduce the control type. This might be the case when the payload is a fragment of a control message, where only the reassembled packet can be interpreted as a control message.

Control messages will be defined in an IANA registry. Control message types 1 through 127 may be defined in standards. Types 128 through 255 are reserved to be user defined for experimentation or private control messages.

This document does not specify any standard control message types other than type 0. Type 0 does not define a format of the control message. Instead, it indicates that the GUE payload is a control message, or part of a control message (as might be the case in GUE fragmentation), that cannot be correctly parsed or interpreted without additional context.

3.3. Flags and extension fields

Flags and associated extension fields are the primary mechanism of extensibility in GUE. As mentioned in section 3.1, GUE header flags indicate the presence of optional extension fields in the GUE header. [GUEEXTEN] defines an initial set of GUE extensions.

3.3.1. Requirements

There are sixteen flag bits in the GUE header. Flags may indicate presence of extension fields. The size of an extension field indicated by a flag MUST be fixed in the specification of the flag.

Flags can be paired together to allow different lengths for an extension field. For example, if two flag bits are paired, a field can possibly be three different lengths-- that is bit value of 00 indicates no field present; 01, 10, and 11 indicate three possible lengths for the field. Regardless of how flag bits are paired, the lengths and offsets of extension fields corresponding to a set of

flags MUST be well defined and deterministic.

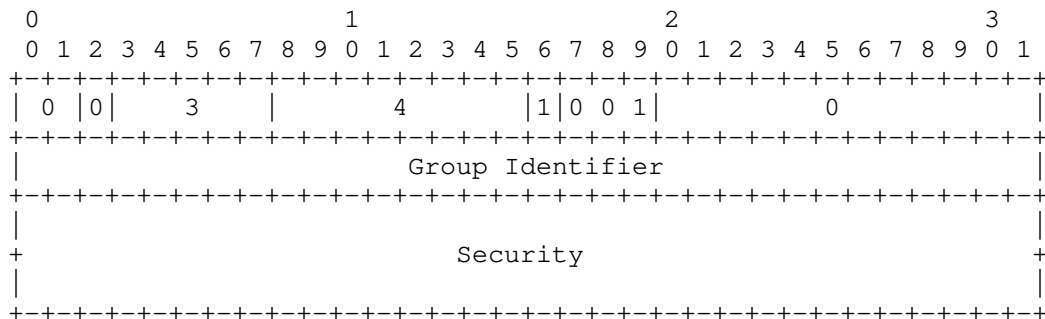
Extension fields are placed in order of the flags. New flags are to be allocated from high to low order bit contiguously without holes. Flags allow random access, for instance to inspect the field corresponding to the Nth flag bit, an implementation only considers the previous N-1 flags to determine the offset. Flags after the Nth flag are not pertinent in calculating the offset of the field for the Nth flag. Random access of flags and fields permits processing of optional extensions in an order that is independent of their position in the packet.

Flags (or paired flags) are idempotent such that new flags MUST NOT cause reinterpretation of old flags. Also, new flags MUST NOT alter interpretation of other elements in the GUE header nor how the message is parsed (for instance, in a data message the proto/ctype field always holds an IP protocol number as an invariant).

The set of available flags can be extended in the future by defining a "flag extensions bit" that refers to a field containing a new set of flags.

3.3.2. Example GUE header with extension fields

An example GUE header for a data message encapsulating an IPv4 packet and containing the Group Identifier and Security extension fields (both defined in [GUEEXTEN]) is shown below:



In the above example, the first flag bit is set which indicates that the Group Identifier extension is present which is a 32 bit field. The second through fourth bits of the flags are paired flags that indicate the presence of a Security field with seven possible sizes. In this example 001 indicates a sixty-four bit security field.

3.4. Private data

An implementation MAY use private data for its own use. The private data immediately follows the last extension field in the GUE header and is not a fixed length. This data is considered part of the GUE header and MUST be accounted for in header length (Hlen). The length of the private data MUST be a multiple of four bytes and is determined by subtracting the offset of private data in the GUE header from the header length. Specifically:

$$\text{Private_length} = (\text{Hlen} * 4) - \text{Length}(\text{flags})$$

where "Length(flags)" returns the sum of lengths of all the extension fields present in the GUE header. When there is no private data present, the length of the private data is zero.

The semantics and interpretation of private data are implementation specific. The private data may be structured as necessary, for instance it might contain its own set of flags and extension fields.

An encapsulator and decapsulator MUST agree on the meaning of private data before using it. The mechanism to achieve this agreement is outside the scope of this document but could include implementation-defined behavior, coordinated configuration, in-band communication using GUE control messages, or out-of-band messages.

If a decapsulator receives a GUE packet with private data, it MUST validate the private data appropriately. If a decapsulator does not expect private data from an encapsulator, the packet MUST be dropped. If a decapsulator cannot validate the contents of private data per the provided semantics, the packet MUST also be dropped. An implementation MAY place security data in GUE private data which if present MUST be verified for packet acceptance.

3.5. Message types

There are two message types in GUE variant 0: control messages and data messages.

3.5.1. Control messages

Control messages carry formatted data that are implicitly addressed to the decapsulator to monitor or control the state or behavior of a tunnel (OAM). For instance, an echo request and corresponding echo reply message can be defined to test for liveness.

Control messages are indicated in the GUE header when the C-bit is set. The payload is interpreted as a control message with type

specified in the proto/ctype field. The format and contents of the control message are indicated by the type and can be variable length.

Other than interpreting the proto/ctype field as a control message type, the meaning and semantics of the rest of the elements in the GUE header are the same as that of data messages. Forwarding and routing of control messages should be the same as that of a data message with the same outer IP and UDP header; this ensures that control messages can be created that follow the same path through the network as data messages.

3.5.2. Data messages

Data messages carry encapsulated packets that are addressed to the protocol stack for the associated protocol. Data messages are a primary means of encapsulation and can be used to create tunnels for overlay networks.

Data messages are indicated in GUE header when the C-bit is not set. The payload of a data message is interpreted as an encapsulated packet of an Internet protocol indicated in the proto/ctype field. The encapsulated packet immediately follows the GUE header.

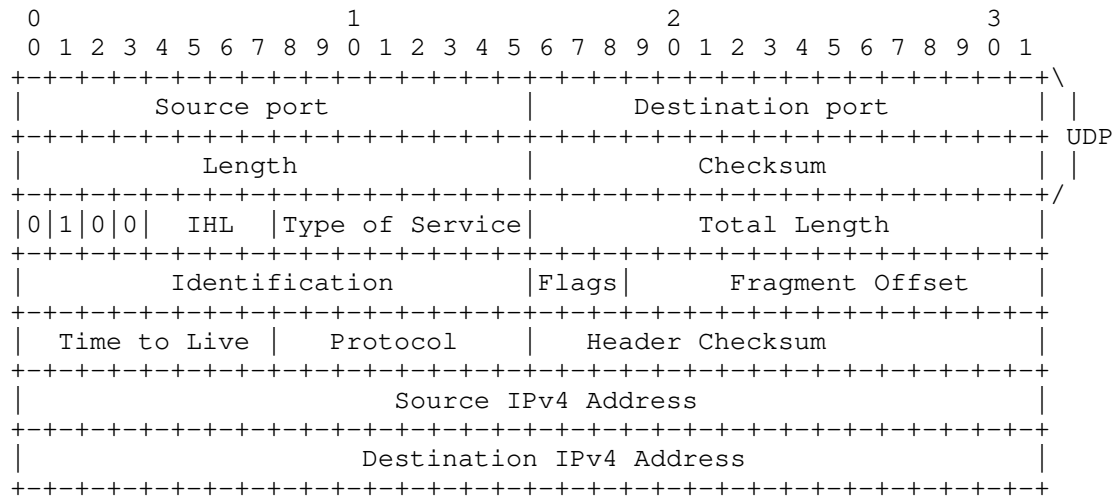
4. Variant 1

Variant 1 of GUE allows direct encapsulation of IPv4 and IPv6 in UDP. In this variant there is no GUE header, a UDP packet carries an IP packet. The first two bits of the UDP payload are the GUE variant field and coincide with the first two bits of the version number in the IP header. The first two version bits of IPv4 and IPv6 are 01, so we use GUE variant 1 for direct IP encapsulation which makes the two bits of GUE variant to also be 01.

This technique is effectively a means to compress out the GUE version 0 header when encapsulating IPv4 or IPv6 packets and there are no flags, extension fields, or private data present. This method is compatible to use on the same port number as packets with the GUE header (GUE variant 0 packets). This technique saves encapsulation overhead on costly links for the common use of IP encapsulation, and also obviates the need to allocate a separate UDP port number for IP-over-UDP encapsulation.

4.1. Direct encapsulation of IPv4

The format for encapsulating IPv4 directly in UDP is:

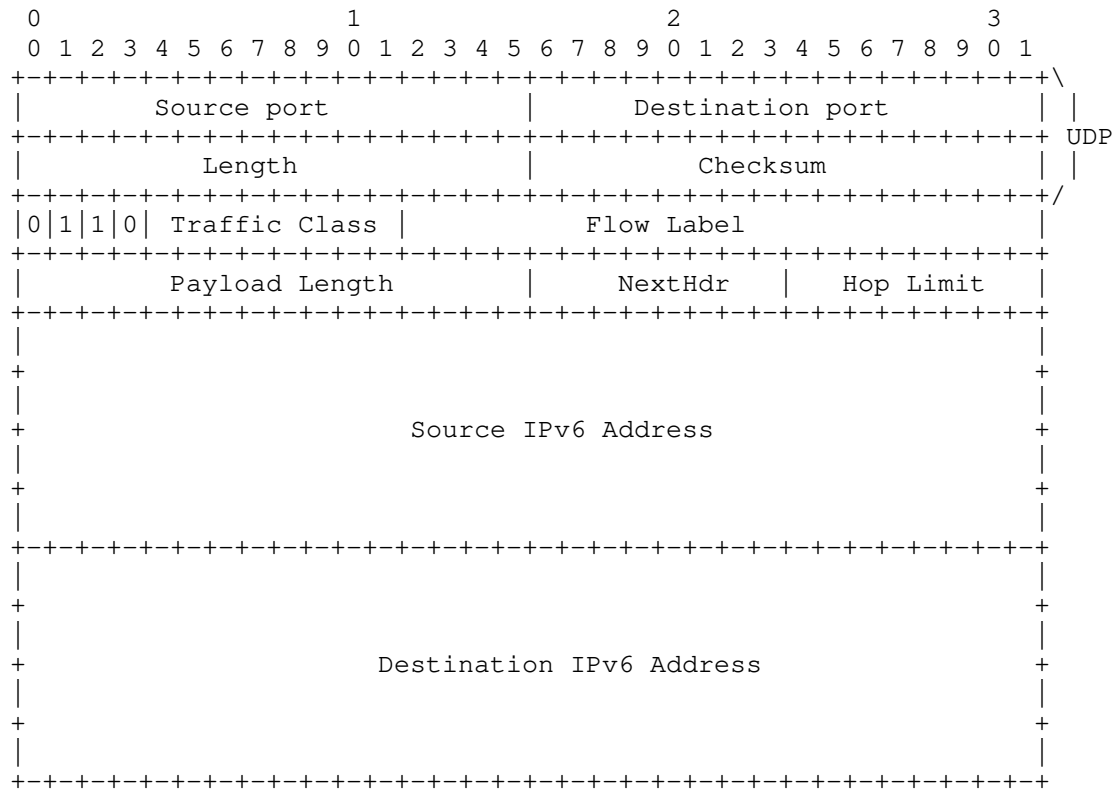


The UDP fields are set in a similar manner as described in section 3.1.

Note that the 0100 value in the first four bits of the UDP payload expresses the GUE variant as 1 (bits 01) and IP version as 4 (bits 0100).

4.2. Direct encapsulation of IPv6

The format for encapsulating IPv6 directly in UDP is demonstrated below:

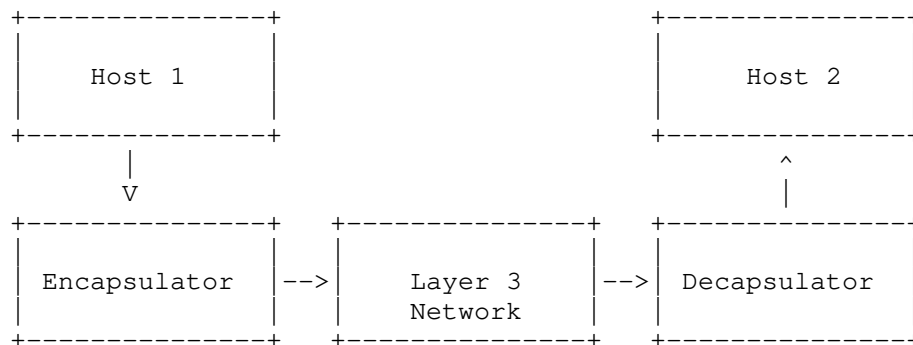


The UDP fields are set in a similar manner as described in section 3.1.

Note that the 0110 value in the first four bits of the the UDP payload expresses the GUE variant as 1 (bits 01) and IP version as 6 (bits 0110).

5. Operation

The figure below illustrates the use of GUE 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 as UDP packets. At the decapsulator, packets are decapsulated and sent on to Host 2. Packet flow in the reverse direction need not be symmetric; for example, the reverse path might not use GUE or any other form of encapsulation.



The encapsulator and decapsulator may be co-resident with the corresponding hosts, or may be on separate nodes in the network.

5.1. Network tunnel 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. These could be routers that provide network tunnels on behalf of communicating hosts.

5.2. Transport layer encapsulation

When encapsulating layer 4 packets, the encapsulator and decapsulator should be co-resident with the hosts. In this case, the encapsulation headers are inserted between the IP header and the transport packet. The addresses in the IP header refer to both the endpoints of the encapsulation and the endpoints for terminating the encapsulated transport protocol. Note that the transport layer ports in the encapsulated packet are independent of the UDP ports in the outer packet.

5.3. Encapsulator operation

Encapsulators create GUE data messages, set the fields of the UDP header, set flags and optional extension fields in the GUE header, and forward packets to a decapsulator.

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 indicated by the outer destination IP address and destination port in the UDP header.

If an encapsulator is tunneling packets -- that is encapsulating packets of layer 2 or layer 3 protocols (e.g. EtherIP, IPIP, ESP tunnel mode) -- it SHOULD follow standard conventions for tunneling one protocol over another. For instance, if an IP packet is being encapsulated in GUE then diffserv interaction [RFC2983] and ECN propagation for tunnels [RFC6040] SHOULD be followed.

5.4. Decapsulator operation

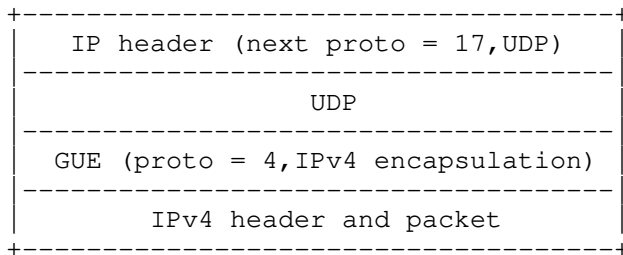
A decapsulator performs decapsulation of GUE packets. A decapsulator is addressed by the outer destination IP address and UDP destination port of a GUE packet. The decapsulator validates packets, including fields of the GUE header.

If a decapsulator receives a GUE packet with an unsupported variant, unknown flag, bad header length (too small for included extension fields), unknown control message type, bad protocol number, an unsupported payload type, or an otherwise malformed header, it MUST drop the packet. Such events MAY be logged subject to configuration and rate limiting of logging messages. Note that set flags in a GUE header that are unknown to a decapsulator MUST NOT be ignored. If a GUE packet is received by a decapsulator with unknown flags, the packet MUST be dropped.

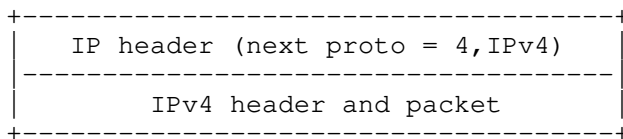
5.4.1. Processing a received data message

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

As an example, consider that a data message is received where GUE encapsulates an IPv4 packet using GUE variant 0. In this case proto field in the GUE header is set to 4 for IPv4 encapsulation:



The receiver removes the UDP and GUE headers and sets the next protocol field in the IP packet to 4, which is derived from the GUE proto field. The resultant packet would have the format:



This packet is then resubmitted into the protocol stack to be processed as an IPv4 encapsulated packet.

5.4.2. Processing a received control message

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

5.5. Middlebox inspection

A middlebox MAY inspect a GUE header. A middlebox MUST NOT modify a GUE header or UDP payload.

To inspect a GUE header, a middlebox needs to identify GUE packets. The obvious method is to match the destination UDP port number to be the GUE port number (i.e. 6080). Per [RFC7605], transport port numbers only have meaning at the endpoints of communications, so inferring the type of a UDP payload based on port number may be incorrect. Middleboxes MUST NOT take any action that would have harmful side effects if a UDP packet were misinterpreted as being a GUE packet. In particular, a middlebox MUST NOT modify a UDP payload based on inferring the payload type from the port number lest the middlebox could cause silent data corruption.

A middlebox MAY interpret some flags and extension fields of the GUE header for classification purposes, but is not required to understand any of the flags or extension fields in GUE packets. A middlebox MUST

NOT drop a GUE packet merely because there are flags unknown to it. Similarly, a middlebox MUST NOT arbitrarily filter packets based on GUE flags or extension fields that are present or not present. The header length in the GUE header allows a middlebox to inspect the payload packet without needing to parse the flags or extension fields.

5.6. Router and switch operation

Routers and switches SHOULD forward GUE packets as standard UDP/IP packets. The outer five-tuple should contain sufficient information to perform flow classification corresponding to the flow of the inner packet. A router does not normally need to parse a GUE header, and none of the flags or extension fields in the GUE header are expected to affect routing. In cases where the outer five-tuple does not provide sufficient entropy for flow classification, for instance UDP ports are fixed to provide connection semantics (section 5.6.1), then the encapsulated packet MAY be parsed to determine flow entropy.

A router MUST NOT modify a GUE header or payload when forwarding a packet. It MAY encapsulate a GUE packet in another GUE packet, for instance to implement a network tunnel (i.e. by encapsulating an IP packet with a GUE payload in another IP packet as a GUE payload). In this case, the router takes the role of an encapsulator, and the corresponding decapsulator is the logical endpoint of the tunnel. When encapsulating a GUE packet within another GUE packet, there are no provisions to automatically copy flags or fields to the outer GUE header. Each layer of encapsulation is considered independent.

5.6.1. Connection semantics

A middlebox might infer bidirectional connection semantics for a UDP flow. For instance, a stateful firewall might create a five-tuple rule to match flows on egress, and a corresponding five-tuple rule for matching ingress packets where the roles of source and destination are reversed for the IP addresses and UDP port numbers. To operate in this environment, a GUE tunnel should be configured to assume connected semantics defined by the UDP five tuple and the use of GUE encapsulation needs to be symmetric between both endpoints. The source port set in the UDP header MUST be the destination port the peer would set for replies. In this case, the UDP source port for a tunnel would be a fixed value and not set to be flow entropy as described in section 5.11.

The selection of whether to make the UDP source port fixed or set to a flow entropy value for each packet sent SHOULD be configurable for a tunnel. The default MUST be to set the flow entropy value in the UDP source port.

5.6.2. NAT

IP address and port translation can be performed on the UDP/IP headers adhering to the requirements for NAT (Network Address Translation) with UDP [RFC4787]. In the case of stateful NAT, connection semantics **MUST** be applied to a GUE tunnel as described in section 5.6.1. GUE endpoints **MAY** also invoke STUN [RFC5389] or ICE [RFC5245] to manage NAT port mappings for encapsulations.

5.7. Checksum Handling

The potential for mis-delivery of packets due to corruption of IP, UDP, or GUE headers needs to be considered. Historically, the UDP checksum would be considered sufficient as a check against corruption of either the UDP header and payload or the IP addresses. Encapsulation protocols, such as GUE, can be originated or terminated on devices incapable of computing the UDP checksum for packet. This section discusses the requirements around checksum and alternatives that might be used when an endpoint does not support UDP checksum.

5.7.1. Requirements

One of the following requirements **MUST** be met:

- o UDP checksums are enabled (for IPv4 or IPv6).
- o The GUE header checksum is used (defined in [GUEEXTEN]).
- o Use zero UDP checksums. This is always permissible with IPv4; in IPv6, they can only be used in accordance with applicable requirements in [RFC8086], [RFC6935], and [RFC6936].

5.7.2. UDP Checksum with IPv4

For UDP in IPv4, the UDP checksum **MUST** be processed as specified in [RFC0768] and [RFC1122] for both transmit and receive. An encapsulator **MAY** set the UDP checksum to zero for performance or implementation considerations. The IPv4 header includes a checksum that protects against mis-delivery of the packet due to corruption of IP addresses. The UDP checksum potentially provides protection against corruption of the UDP header, GUE header, and GUE payload. Enabling or disabling the use of checksums is a deployment consideration that should take into account the risk and effects of packet corruption, and whether the packets in the network are already adequately protected by other, possibly stronger mechanisms, such as the Ethernet CRC. If an encapsulator sets a zero UDP checksum for IPv4, it **SHOULD** use the GUE header checksum as described in [GUEEXTEN] if there are no other mechanisms used that would detect

corruption of GUE packets.

When a decapsulator receives a packet, the UDP checksum field MUST be processed. If the UDP checksum is non-zero, the decapsulator MUST verify the checksum before accepting the packet. By default, a decapsulator SHOULD accept UDP packets with a zero checksum. A node MAY be configured to disallow zero checksums per [RFC1122]. Configuration of zero checksums can be selective. For instance, zero checksums might be disallowed from certain hosts that are known to be traversing paths subject to packet corruption. If verification of a non-zero checksum fails, a decapsulator lacks the capability to verify a non-zero checksum, or a packet with a zero-checksum was received and the decapsulator is configured to disallow that, then the packet MUST be dropped.

5.7.3. UDP Checksum with IPv6

In IPv6, there is no checksum in the IPv6 header that protects against mis-delivery due to address corruption. Therefore, when GUE is used over IPv6, either the UDP checksum or the GUE header checksum SHOULD be used unless there are alternative mechanisms in use that protect against misdelivery. The UDP checksum and GUE header checksum SHOULD NOT be used at the same time since that would be mostly redundant.

If neither the UDP checksum nor the GUE header checksum is used, then the requirements for using zero IPv6 UDP checksums in [RFC6935] and [RFC6936] MUST be met.

When a decapsulator receives a packet, the UDP checksum field MUST be processed. If the UDP checksum is non-zero, the decapsulator MUST verify the checksum before accepting the packet. By default a decapsulator MUST only accept UDP packets with a zero checksum if the GUE header checksum is used and is verified. If verification of a non-zero checksum fails or a decapsulator lacks the capability to verify a non-zero checksum then the packet MUST be dropped. If a packet is received with a zero UDP checksum, no GUE header checksum, and zero UDP checksums are disallowed then the packet MUST be dropped.

5.8. MTU and fragmentation

Standard conventions for handling of MTU (Maximum Transmission Unit) and fragmentation in conjunction with networking tunnels (encapsulation of layer 2 or layer 3 packets) SHOULD be followed. Details are described in MTU and Fragmentation Issues with In-the-Network Tunneling [RFC4459].

If a packet is fragmented before encapsulation in GUE, all the related fragments MUST be encapsulated using the same UDP source port. An operator SHOULD set MTU to account for encapsulation overhead and reduce the likelihood of fragmentation.

Alternative to IP fragmentation, the GUE fragmentation extension can be used. GUE fragmentation is described in [GUEEXTEN].

5.9. Congestion control

Per requirements of [RFC8085], if the IP traffic encapsulated with GUE implements proper congestion control then no additional mechanisms should be required.

In the case that the encapsulated traffic does not implement any or sufficient control, or it is not known whether a transmitter will consistently implement proper congestion control, then congestion control at the encapsulation layer MUST be provided per [RFC8085]. Note that this case applies to a significant use case in network virtualization in which guests run third party networking stacks that cannot be implicitly trusted to implement conformant congestion control.

Out of band mechanisms such as rate limiting, Managed Circuit Breaker [RFC8084], or traffic isolation MAY be used to provide rudimentary congestion control. For finer-grained congestion control that allows alternate congestion control algorithms, reaction time within an RTT, and interaction with ECN, in-band mechanisms might be warranted.

5.10. Multicast

GUE packets can be multicast to decapsulators using a multicast destination address in the outer IP header. Each receiving host will decapsulate the packet independently following normal decapsulator operations. The receiving decapsulators need to agree on the same set of GUE parameters and properties; how such an agreement is reached is outside the scope of this document.

GUE allows encapsulation of unicast, broadcast, or multicast traffic. Flow entropy (the value in the UDP source port) can be generated from the header of encapsulated unicast or broadcast/multicast packets at an encapsulator. The mapping mechanism between the encapsulated multicast traffic and the multicast capability in the IP network is transparent and independent of the encapsulation and is otherwise outside the scope of this document.

5.11. Flow entropy for ECMP

A major objective of using GUE is that a network device can perform flow classification corresponding to the flow of the inner encapsulated packet based on the contents of the outer headers.

5.11.1. Flow classification

When a packet is encapsulated with GUE and connection semantics are not applied, the source port in the outer UDP packet is set to a flow entropy value that corresponds to the flow of the inner packet. When a device computes a five-tuple hash on the outer UDP/IP header of a GUE packet, the resultant value classifies the packet per its inner flow.

Examples of deriving flow entropy for encapsulation are:

- o If the encapsulated packet is a layer 4 packet, TCP/IPv4 for instance, the flow entropy could be based on the canonical five-tuple hash of the inner packet.
- o If the encapsulated packet is an AH transport mode packet with TCP as next header, the flow entropy could be a hash over a three-tuple: TCP protocol and TCP ports of the encapsulated packet.
- o If a node is encrypting a packet using ESP tunnel mode and GUE encapsulation, the flow entropy could be based on the contents of the clear-text packet. For instance, a canonical five-tuple hash for a TCP/IP packet could be used.

[RFC6438] discusses methods to compute and set flow entropy value for IPv6 flow labels, such methods can also be used to create flow entropy values for GUE.

5.11.2. Flow entropy properties

The flow entropy is the value set in the UDP source port of a GUE packet. Flow entropy in the UDP source port SHOULD adhere to the following properties:

- o The value set in the source port is within the ephemeral port range (49152 to 65535 [RFC6335]). Since the high order two bits of the port are set to one, this provides fourteen bits of entropy for the value.
- o The flow entropy has a uniform distribution across encapsulated flows.
- o An encapsulator MAY occasionally change the flow entropy used

for an inner flow per its discretion (for security, route selection, etc). To avoid thrashing or flapping the value, the flow entropy used for a flow SHOULD NOT change more than once every thirty seconds (or a configurable value).

- o Decapsulators, or any networking devices, SHOULD NOT attempt to interpret flow entropy as anything more than an opaque value. Neither should they attempt to reproduce the hash calculation used by an encapsulator in creating a flow entropy value. They MAY use the value to match further receive packets for steering decisions, but MUST NOT assume that the hash uniquely or permanently identifies a flow.
- o Input to the flow entropy calculation is not restricted to ports and addresses; input could include the flow label from an IPv6 packet, SPI from an ESP packet, or other flow related state in the encapsulator that is not necessarily conveyed in the packet.
- o The assignment function for flow entropy SHOULD be randomly seeded to mitigate denial of service attacks. The seed SHOULD be changed periodically.

5.12. Negotiation of acceptable flags and extension fields

An encapsulator and decapsulator need to achieve agreement about GUE parameters that will be used in communications. Parameters include supported GUE variants, flags and extension fields that can be used, security algorithms and keys, supported protocols and control messages, etc. This document proposes different general methods to accomplish this, however the details of implementing these are considered out of scope.

General methods for this are:

- o Configuration. The parameters used for a tunnel are configured at each endpoint.
- o Negotiation. A tunnel negotiation can be performed. This could be accomplished in-band of GUE using control messages.
- o Via a control plane. Parameters for communicating with a tunnel endpoint can be set in a control plane protocol (such as that needed for network virtualization).
- o Via security negotiation. Use of security typically implies a key exchange between endpoints. Other GUE parameters may be conveyed as part of that process.

6. Motivation for GUE

This section provides the motivation for GUE with respect to other encapsulation methods.

6.1. Benefits of GUE

- * GUE is a generic encapsulation protocol. GUE can encapsulate protocols that are represented by an IP protocol number. This includes layer 2, layer 3, and layer 4 protocols.
- * GUE is an extensible encapsulation protocol. Standardized optional data such as security, virtual networking identifiers, fragmentation are defined.
- * For extensibility, GUE uses flag fields as opposed to TLVs as some other encapsulation protocols do. Flag fields are strictly ordered, allow random access, and are efficient in use of header space.
- * GUE allows private data to be sent as part of the encapsulation. This permits experimentation or customization in deployment.
- * GUE allows sending of control messages such as OAM using the same GUE header format (for routing purposes) as normal data messages.
- * GUE maximizes deliverability of non-UDP and non-TCP protocols.
- * GUE provides a means for exposing per flow entropy for ECMP for atypical protocols such as SCTP, DCCP, ESP, etc.

6.2. Comparison of GUE to other encapsulations

A number of different encapsulation techniques have been proposed for the encapsulation of one protocol over another. EtherIP [RFC3378] provides layer 2 tunneling of Ethernet frames over IP. GRE [RFC2784], MPLS [RFC4023], and L2TP [RFC2661] provide methods for tunneling layer 2 and layer 3 packets over IP. NVGRE [RFC7637] and VXLAN [RFC7348] are proposals for encapsulation of layer 2 packets for network virtualization. IPIP [RFC2003] and Generic packet tunneling in IPv6 [RFC2473] provide methods for tunneling IP packets over IP.

Several proposals exist for encapsulating packets over UDP including ESP over UDP [RFC3948], TCP directly over UDP [TCPUDP], VXLAN [RFC7348], LISP [RFC6830] which encapsulates layer 3 packets, MPLS/UDP [RFC7510], GENEVE [GENEVE], and GRE-in-UDP Encapsulation [RFC8086].

GUE has the following discriminating features:

- o UDP encapsulation leverages specialized network device processing for efficient transport. The semantics for using the UDP source port for flow entropy as input to ECMP are defined in section 5.11.
- o GUE permits encapsulation of arbitrary IP protocols, which includes layer 2, 3, and 4 protocols.
- o Multiple protocols can be multiplexed over a single UDP port number. This is in contrast to techniques to encapsulate protocols over UDP using a protocol specific port number (such as ESP/UDP, GRE/UDP, SCTP/UDP). GUE provides a uniform and extensible mechanism for encapsulating all IP protocols in UDP with minimal overhead (four bytes of additional header).
- o GUE is extensible. New flags and extension fields can be defined.
- o The GUE header includes a header length field. This allows a network node to inspect an encapsulated packet without needing to parse the full encapsulation header.
- o Private data in the encapsulation header allows local customization and experimentation while being compatible with processing in network nodes (routers and middleboxes).
- o GUE includes both data messages (encapsulation of packets) and control messages (such as OAM).
- o The flags-field model facilitates efficient implementation of extensibility in hardware. For instance, a TCAM can be used to parse a known set of N flags where the number of entries in the TCAM is 2^N . By comparison, the number of TCAM entries needed to parse a set of N arbitrarily ordered TLVs is approximately e^N .
- o GUE includes a variant that encapsulates IPv4 and IPv6 packets directly within UDP.

7. Security Considerations

There are two important considerations of security with respect to GUE.

- o Authentication and integrity of the GUE header.
- o Authentication, integrity, and confidentiality of the GUE payload.

GUE security is provided by extensions for security defined in [GUEEXTEN]. These extensions include methods to authenticate the GUE header and encrypt the GUE payload.

The GUE header can be authenticated using a security extension for an HMAC (Hashed Message Authentication Code). Securing the GUE payload can be accomplished use of the GUE Payload Transform extension. This extension allows the use of DTLS (Datagram Transport Layer Security) to encrypt and authenticate the GUE payload.

A hash function for computing flow entropy (section 5.11) SHOULD be randomly seeded to mitigate some possible denial service attacks.

8. IANA Considerations

8.1. UDP source port

A user UDP port number assignment for GUE has been assigned:

```
Service Name: gue
Transport Protocol(s): UDP
Assignee: Tom Herbert <tom@herbertland.com>
Contact: Tom Herbert <tom@herbertland.com>
Description: Generic UDP Encapsulation
Reference: draft-herbert-gue
Port Number: 6080
Service Code: N/A
Known Unauthorized Uses: N/A
Assignment Notes: N/A
```

8.2. GUE variant number

IANA is requested to set up a registry for the GUE variant number. The GUE variant number is two bits containing four possible values. This document defines variants 0 and 1. New values are assigned in accordance with RFC Required policy [RFC5226].

Variant number	Description	Reference
0	GUE Version 0 with header	This document
1	GUE Version 0 with direct IP encapsulation	This document
2..3	Unassigned	

8.3. Control types

IANA is requested to set up a registry for the GUE control types. Control types are 8 bit values. New values for control types 1-127 are assigned in accordance with RFC Required policy [RFC5226].

Control type	Description	Reference
0	Control payload needs more context for interpretation	This document
1..127	Unassigned	
128..255	User defined	This document

9. Acknowledgements

The authors would like to thank David Liu, Erik Nordmark, Fred Templin, Adrian Farrel, Bob Briscoe, and Murray Kucherawy for valuable input on this draft. Special thanks to Fred Templin who is serving as document shepherd.

10. References

10.1. Normative References

- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<http://www.rfc-editor.org/info/rfc768>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [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>>.
- [RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <<http://www.rfc-editor.org/info/rfc6935>>.
- [RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <<http://www.rfc-editor.org/info/rfc6936>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<http://www.rfc-editor.org/info/rfc1122>>.
- [RFC4459] Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, DOI 10.17487/RFC4459, April 2006, <<http://www.rfc-editor.org/info/rfc4459>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and

Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.

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

[GUEEXTEN] Herbert, T., Yong, L., and Templin, F., "Extensions for Generic UDP Encapsulation", draft-herbert-gue-extensions-06

10.2. Informative References

[RFC8086] Yong, L., Ed., Crabbe, E., Xu, X., and T. Herbert, "GRE-in-UDP Encapsulation", RFC 8086, DOI 10.17487/RFC8086, March 2017, <<http://www.rfc-editor.org/info/rfc8086>>.

[RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <<https://www.rfc-editor.org/info/rfc7605>>.

[RFC4787] Audet, F., Ed., and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<http://www.rfc-editor.org/info/rfc4787>>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <<http://www.rfc-editor.org/info/rfc5389>>.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.

[RFC8084] Fairhurst, G., "Network Transport Circuit Breakers", BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017, <<https://www.rfc-editor.org/info/rfc8084>>.

[RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011, <<http://www.rfc-editor.org/info/rfc6438>>.

- [RFC3378] Housley, R. and S. Hollenbeck, "EtherIP: Tunneling Ethernet Frames in IP Datagrams", RFC 3378, DOI 10.17487/RFC3378, September 2002, <<http://www.rfc-editor.org/info/rfc3378>>.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<http://www.rfc-editor.org/info/rfc2784>>.
- [RFC4023] Worster, T., Rekhter, Y., and E. Rosen, Ed., "Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)", RFC 4023, DOI 10.17487/RFC4023, March 2005, <<http://www.rfc-editor.org/info/rfc4023>>.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<http://www.rfc-editor.org/info/rfc2661>>.
- [RFC7637] Garg, P., Ed., and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <<https://www.rfc-editor.org/info/rfc7637>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, August 2014, <<http://www.rfc-editor.org/info/rfc7348>>.
- [RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <<http://www.rfc-editor.org/info/rfc2003>>.
- [RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473, December 1998, <<https://www.rfc-editor.org/info/rfc2473>>.
- [RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, DOI 10.17487/RFC3948, January 2005, <<http://www.rfc-editor.org/info/rfc3948>>.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <<http://www.rfc-editor.org/info/rfc6830>>.

editor.org/info/rfc6830>.

- [RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black,
"Encapsulating MPLS in UDP", RFC 7510, DOI
10.17487/RFC7510, April 2015, <<http://www.rfc-editor.org/info/rfc7510>>.
- [IANA-PN] IANA, "Protocol Numbers",
<<https://www.iana.org/assignments/protocol-numbers>>.
- [TCPUDP] Chesire, S., Graessley, J., and McGuire, R.,
"Encapsulation of TCP and other Transport Protocols over
UDP", draft-cheshire-tcp-over-udp-00
- [GENEVE] Gross, J., Ed., Ganga, I. Ed., and Sridhar, T., "Geneve:
Generic Network Virtualization Encapsulation", draft-ietf-
nvo3-geneve-10
- [UDPENCAP] Herbert, T., "UDP Encapsulation in Linux",
<[http://people.netfilter.org/pablo/netdev0.1/papers/UDP-
Encapsulation-in-Linux.pdf](http://people.netfilter.org/pablo/netdev0.1/papers/UDP-Encapsulation-in-Linux.pdf)>
- [MULTIQ] Herbert, T. and de Bruijn, W., "Scaling in the Linux
Networking Stack", <[https://www.kernel.org/doc/
Documentation/networking/scaling.txt](https://www.kernel.org/doc/Documentation/networking/scaling.txt)>
- [CSUMOFF] Cree, E., "Checksum Offloads in the Linux Networking
Stack", <[https://www.kernel.org/doc/Documentation/
networking/checksum-offloads.txt](https://www.kernel.org/doc/Documentation/networking/checksum-offloads.txt)>
- [SEGOFF] Duyck, A., "Segmentation Offloads in the Linux Networking
Stack", <[https://www.kernel.org/doc/
Documentation/networking/segmentation-offloads.txt](https://www.kernel.org/doc/Documentation/networking/segmentation-offloads.txt)>

Appendix A: NIC processing for GUE

This appendix is informational and does not constitute a normative part of this document.

This appendix provides some guidelines for Network Interface Cards (NICs) to implement common offloads and accelerations to support GUE. Note that most of this discussion is generally applicable to other methods of UDP based encapsulation. An overview of UDP based encapsulation and acceleration is in [UDPENCAP]

A.1. Receive multi-queue

Contemporary NICs support multiple receive descriptor queues (multi-queue) [MUTLIQ]. Multi-queue enables load balancing of network processing for a NIC across multiple CPUs. On packet reception, a NIC selects an appropriate queue for host processing. Receive Side Scaling (RSS) is a common method which uses the flow hash for a packet to index an indirection table where each entry stores a queue number. Flow Director and Accelerated Receive Flow Steering (aRFS) allow a host to program the queue that is used for a given flow which is identified either by an explicit five-tuple or by the flow's hash.

GUE encapsulation is compatible with multi-queue NICs that support five-tuple hash calculation for UDP/IP packets as input to RSS. The flow entropy in the UDP source port ensures classification of the encapsulated flow even in the case that the outer source and destination addresses are the same for all flows (e.g. all flows are going over a single tunnel).

By default, UDP RSS support is often disabled in NICs to avoid out-of-order reception that can occur when UDP packets are fragmented. As discussed in section 5.8, fragmentation of GUE packets is mostly avoided by fragmenting packets before entering a tunnel, GUE fragmentation, path MTU discovery in higher layer protocols, or operator adjusting MTUs. Other UDP traffic might not implement such procedures to avoid fragmentation, so enabling UDP RSS support in the NIC might be a considered tradeoff during configuration.

A.2. Checksum offload

Many NICs provide capabilities to calculate the standard ones complement checksum for packets in transmit or receive [CSUMOFF]. When using GUE encapsulation, there are at least two checksums that are of interest: the encapsulated packet's transport checksum, and the UDP checksum in the outer header.

A.2.1. Transmit checksum offload

NICs can provide a protocol agnostic method to offload the transmit checksum (NETIF_F_HW_CSUM in Linux parlance) that can be used with GUE. In this method, the host provides checksum related parameters in a transmit descriptor for a packet. These parameters include the starting offset of data to checksum, the length of data to checksum, and the offset in the packet where the computed checksum is to be written. The host initializes the checksum field to a pseudo header checksum.

In the case of GUE, the checksum for an encapsulated transport layer packet, a TCP packet for instance, can be offloaded by setting the appropriate checksum parameters.

NICs typically can offload only one transmit checksum per packet, so simultaneously offloading both an inner transport packet's checksum and the outer UDP checksum is likely not possible.

If an encapsulator is co-resident with a host, then checksum offload may be performed using remote checksum offload (RCO) [GUEEXTEN]. Remote checksum offload relies on NIC offload of the simple UDP/IP checksum which is commonly supported even in legacy devices. In remote checksum offload, the outer UDP checksum is set and the GUE header includes an option indicating the start and offset of the inner "offloaded" checksum. The inner checksum is initialized to the pseudo header checksum. When a decapsulator receives a GUE packet with the remote checksum offload option, it completes the offload operation by determining the packet checksum from the indicated start point to the end of the packet, and then adds this into the checksum field at the offset given in the option. Computing the checksum from the start to end of packet is efficient if checksum-complete is provided on the receiver.

Another alternative when an encapsulator is co-resident with a host is to perform Local Checksum Offload (LCO) [CSUMOFF]. In this method, the inner transport layer checksum is offloaded and the outer UDP checksum can be deduced based on the fact that the portion of the packet covered by the inner transport checksum will sum to zero or at least the bitwise "not" of the inner pseudo header.

A.2.2. Receive checksum offload

GUE is compatible with NICs that perform a protocol agnostic receive checksum (CHECKSUM_COMPLETE in Linux parlance). In this technique, a NIC computes a ones complement checksum over all (or some predefined portion) of a packet. The computed value is provided to the host stack in the packet's receive descriptor. The host driver can use

this checksum to "patch up" and validate any inner packet transport checksums, as well as the outer UDP checksum if it is non-zero.

Many legacy NICs don't provide checksum-complete but instead provide an indication that a checksum has been verified (CHECKSUM_UNNECESSARY in Linux). Usually, such validation is only done for simple TCP/IP or UDP/IP packets. If a NIC indicates that a UDP checksum is valid, the checksum-complete value for the UDP packet is the bitwise "not" of the pseudo header checksum. In this way, checksum-unnecessary can be converted to checksum-complete. So, if the NIC provides checksum-unnecessary for the outer UDP header in an encapsulation, checksum conversion can be done so that the checksum-complete value is derived and can be used by the stack to validate checksums in the encapsulated packet.

A.3. Transmit Segmentation Offload

Transmit Segmentation Offload (TSO) [SEGOFF] is a NIC feature where a host provides a large (>MTU size) TCP packet to the NIC, which in turn splits the packet into separate segments and transmits each one. This is useful to reduce CPU load on the host.

The process of TSO can be generalized as:

- Split the TCP payload into segments of size less than or equal to MTU.
- For each created segment:
 1. Replicate the TCP header and all preceding headers of the original packet.
 2. Set payload length fields in any headers to reflect the length of the segment.
 3. Set TCP sequence number to correctly reflect the offset of the TCP data in the stream.
 4. Recompute and set any checksums that either cover the payload of the packet or cover header which was changed by setting a payload length.

Following this general process, TSO can be extended to support TCP encapsulation in GUE. For each segment the Ethernet, outer IP, UDP header, GUE header, inner IP header (if tunneling), and TCP headers are replicated. Any packet length header fields need to be set properly (including the length in the outer UDP header), and checksums need to be set correctly (including the outer UDP checksum

if being used).

To facilitate TSO with GUE, it is recommended that extension fields do not contain values that need to be updated on a per segment basis. For example, extension fields should not include checksums, lengths, or sequence numbers that refer to the payload. If the GUE header does not contain such fields then the TSO engine only needs to copy the bits in the GUE header when creating each segment and does not need to parse the GUE header.

A.4. Large Receive Offload

Large Receive Offload (LRO) [SEGOFF] is a NIC feature where packets of a TCP connection are reassembled, or coalesced, in the NIC and delivered to the host as one large packet. This feature can reduce CPU utilization in the host.

LRO requires significant protocol awareness to be implemented correctly and is difficult to generalize. Packets in the same flow need to be unambiguously identified. In the presence of tunnels or network virtualization, this may require more than a five-tuple match (for instance packets for flows in two different virtual networks may have identical five-tuples). Additionally, a NIC needs to perform validation over packets that are being coalesced, and needs to fabricate a single meaningful header from all the coalesced packets.

The conservative approach to supporting LRO for GUE would be to assign packets to the same flow only if they have identical five-tuple and were encapsulated the same way. That is the outer IP addresses, the outer UDP ports, GUE protocol, GUE flags and fields, and inner five tuple are all identical.

Appendix B: Implementation considerations

This appendix is informational and does not constitute a normative part of this document.

B.1. Priveleged ports

Using the source port to contain a flow entropy value disallows the security method of a receiver enforcing that the source port be a privileged port. Privileged ports are defined by some operating systems to restrict source port binding. Unix, for instance, considered port number less than 1024 to be privileged.

Enforcing that packets are sent from a privileged port is widely considered an inadequate security mechanism and has been mostly deprecated. To approximate this behavior, an implementation could

restrict a user from sending a packet destined to the GUE port without proper credentials.

B.2. Setting flow entropy as a route selector

An encapsulator generating flow entropy in the UDP source port could modulate the value to perform a type of multipath source routing. Assuming that networking switches perform ECMP based on the flow hash, a sender can affect the path by altering the flow entropy. For instance, a host can store a flow hash in its protocol control block (PCB) for an inner flow, and might alter the value upon detecting that packets are traversing a lossy path. Changing the flow entropy for a flow SHOULD be subject to hysteresis (at most once every thirty seconds) to limit the number of out of order packets.

B.3. Hardware protocol implementation considerations

Low level data path protocols, such as GUE, are often supported in high speed network device hardware. Variable length header (VLH) protocols like GUE are sometimes considered difficult to efficiently implement in hardware. In order to retain the important characteristics of an extensible and robust protocol, hardware vendors may practice "constrained flexibility". In this model, only certain combinations or protocol header parameterizations are implemented in the hardware fast path. Each such parameterization is fixed length so that the particular instance can be optimized as a fixed length protocol. In the case of GUE, this constitutes specific combinations of GUE flags, fields, and next protocol. The selected combinations would naturally be the most common cases which form the "fast path", and other combinations are assumed to take the "slow path".

In time, the needs and requirements of a protocol may change which may manifest themselves as new parameterizations to be supported in the fast path. To allow this extensibility, a device practicing constrained flexibility should allow fast path parameterizations to be programmable.

Authors' Addresses

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

Email: tom@herbertland.com

Lucy Yong
Independent
Austin, TX
US

Osama Zia
Microsoft
1 Microsoft Way
Redmond, WA 98029
US

Email: osamaz@microsoft.com

INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: September 9, 2019

T. Herbert
Quantonium
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

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

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

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

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

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

Copyright and License Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. GUE header format with optional extensions	4
3. Group identifier option	6
3.1. Extension field format	6
3.2. Usage	6
4. Security option	6
4.1. Extension field format	7
4.2. Usage	7
4.3. Cookies	8
4.4.1. Extension field format	9
4.3.1. Operation	8
4.3.1.1. Transmitter operation	8
4.3.1.2. Receiver operation	9
4.4. HMAC	9
4.4.1. Extension field format	9
4.4.2. Selecting a hash algorithm	10
4.4.3. Pre-shared key management	11
4.4.4. Operation	11
4.4.4.1. Transmitter operation	11
4.4.4.2. Receiver operation	11
4.5. Interaction with other optional extensions	12
5. Fragmentation option	12
5.1. Motivation	13
5.2. Scope	14
5.3. Extension field format	14
5.4. Fragmentation procedure	15
5.5. Reassembly procedure	17
5.6. Security Considerations	19
6. Payload transform option	19
6.1. Extension field format	19
6.2. Usage	20
6.3. Interaction with other optional extensions	21
6.4. DTLS transform	21
7. Remote checksum offload option	22
7.1. Extension field format	22
7.2. Usage	22
7.2.1. Transmitter operation	22
7.2.2. Receiver operation	23

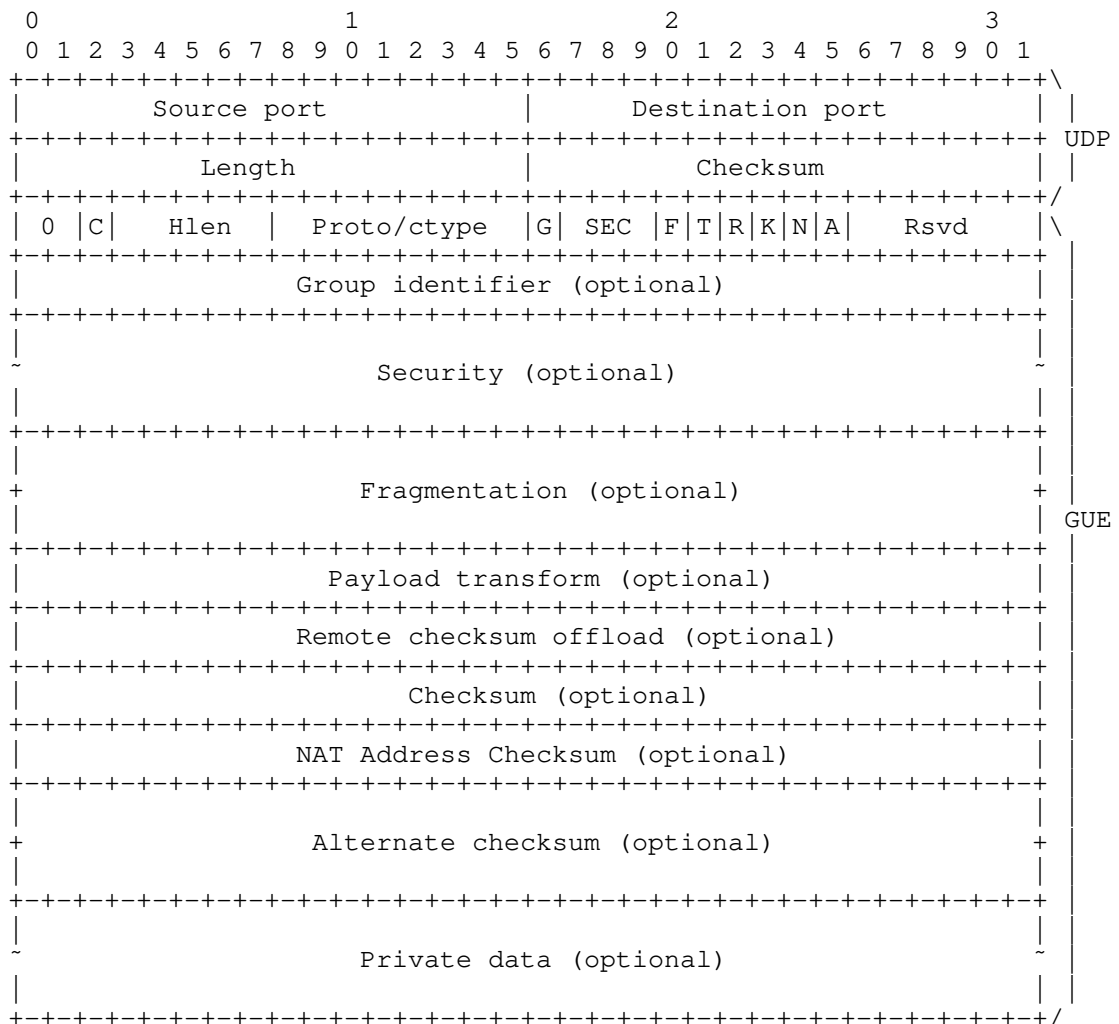
7.3. Security Considerations	24
8. Checksum option	24
8.1. Extension field format	24
8.2. Requirements	25
8.3. GUE checksum pseudo header	25
8.4. Usage	26
8.4.1. Transmitter operation	27
8.4.2. Receiver operation	27
8.5. Corrupted checksum flag	28
8.6. Security Considerations	28
9. NAT checksum address option	28
9.1. Extension field format	28
9.2. Usage	29
9.2.1. Transmitter operation	29
9.2.2. Receiver operation	30
10. Alternative checksum option	30
10.1. Extension field format	31
10.2. Usage	31
10.2.2. Receiver operation	32
10.3. Corrupted alternate checksum flag	32
10.4. Security Considerations	33
11. Processing order of options	33
11.1. Processing order when sending	33
11.2. Processing order when receiving	34
12. Security Considerations	34
13. IANA Consideration	35
14. References	37
14.1. Normative References	37
14.2. Informative References	37
Authors' Addresses	39

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:



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 $(\text{header_len} - 4) / 4$. The length of the encapsulated packet is determined from the UDP length and the Hlen: $\text{encapsulated_packet_length} = \text{UDP_Length} - 12 - 4 * \text{Hlen}$.
- 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.
- o G: Indicates the the group identifier extension field is present. The group identifier option is described in section 3.
- o SEC: Indicates security extension field is present. The security option is described in section 4.
- o F: Indicates fragmentation extension field is present. The fragmentation option is described in section 5.
- o T: Indicates payload transform extension field is present. The payload transform option is described in section 6.
- o R: Indicates the remote checksum extension field is present. The remote checksum offload option is described in section 7.
- o K: Indicates checksum extension field is present. The checksum option is described in section 8.
- o N: Indicates NAT address checksum field is present. The NAT

address checksum option is described in section 9.

- o A: Indicates alternative checksum field is present. The alternative checksum option is described in section 10.
- o 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

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.

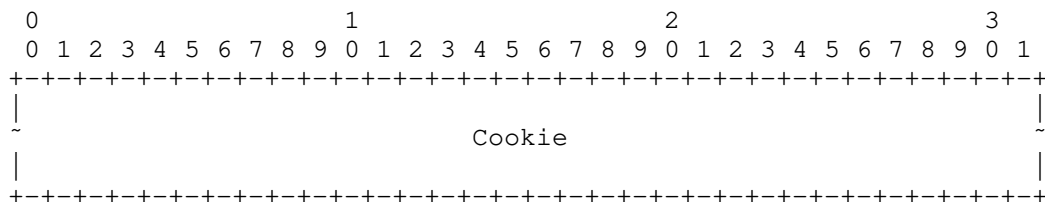
4.3. Cookies

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:



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.

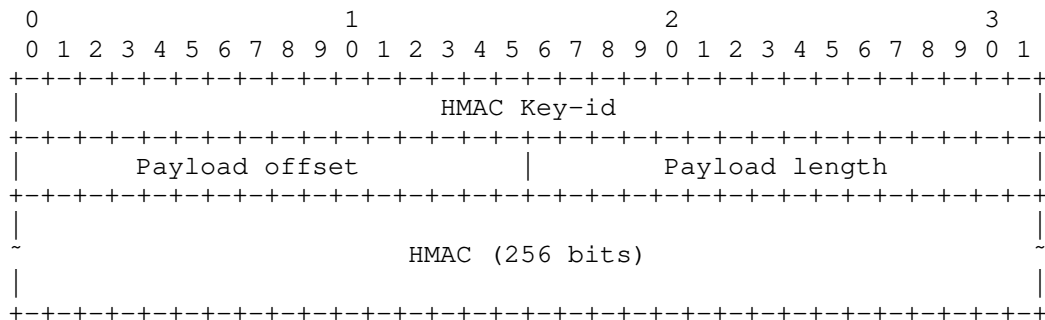
4.4. HMAC

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:



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 is 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:

- o 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.
- o 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:

- o In the configuration of the endpoints
- o 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 encapsulator 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 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 encapsulator 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.

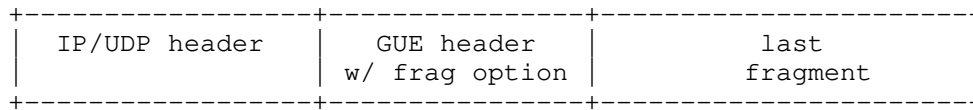
5.2. Scope

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:

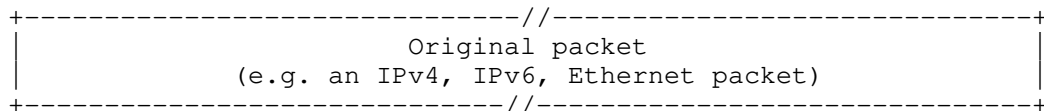


Each fragment packet is composed of:

- (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 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:



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 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 decapsulator 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.

(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 than 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.

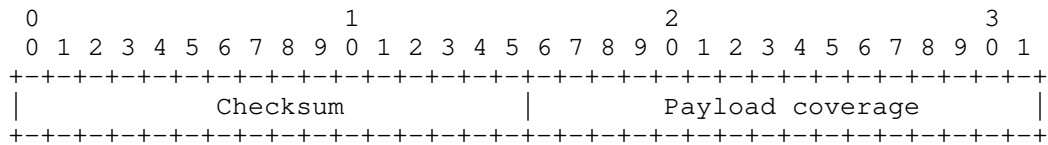
8. Checksum option

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:



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.

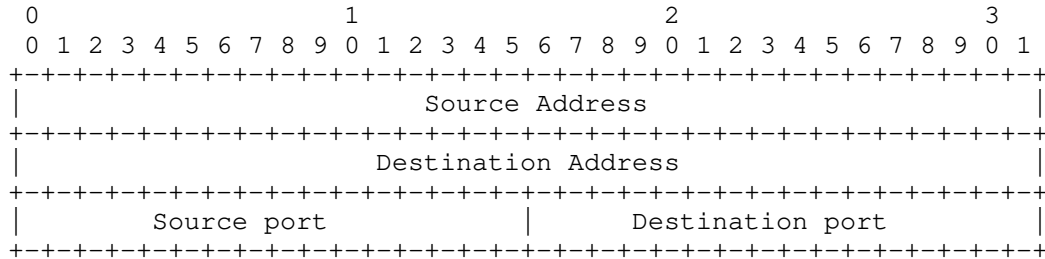
The GUE header checksum SHOULD be used when the UDP checksum is zero for IPv4 if the GUE header includes data that when corrupted can lead to misdelivery or other serious consequences, and there is no other mechanism that provides protection (no security field that checks integrity for instance).

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).

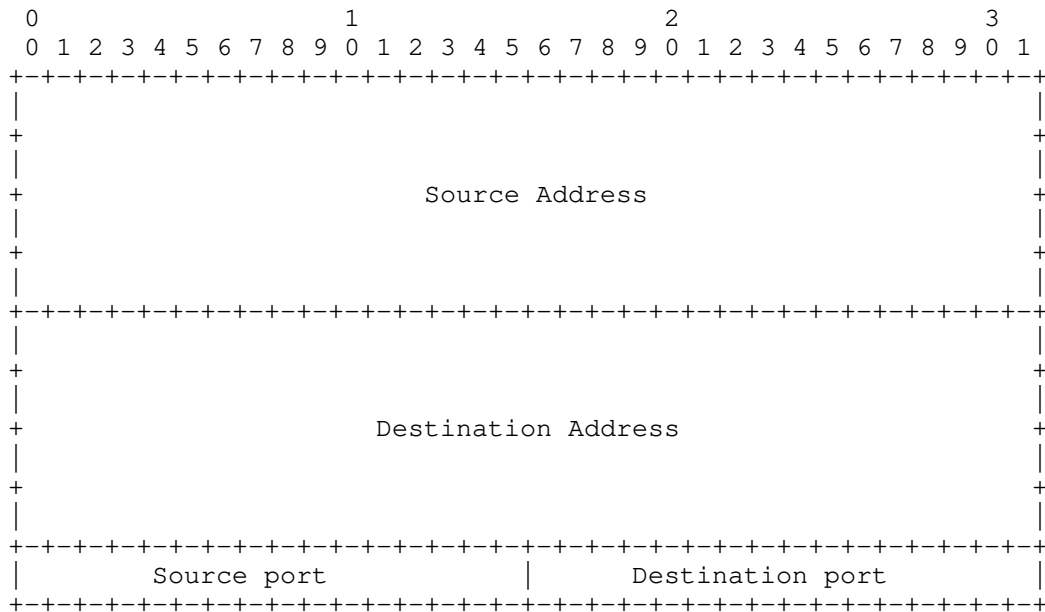
8.3. GUE checksum pseudo header

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:



The GUE pseudo header for IPv6 is:



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.

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.

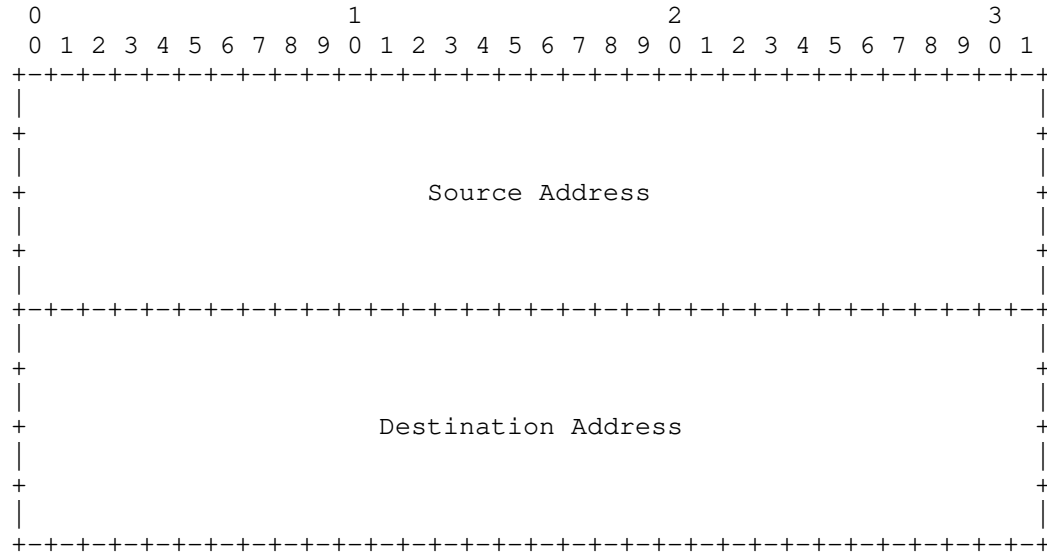
For IPv4 the checksum is computed over:

```

      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:



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

```

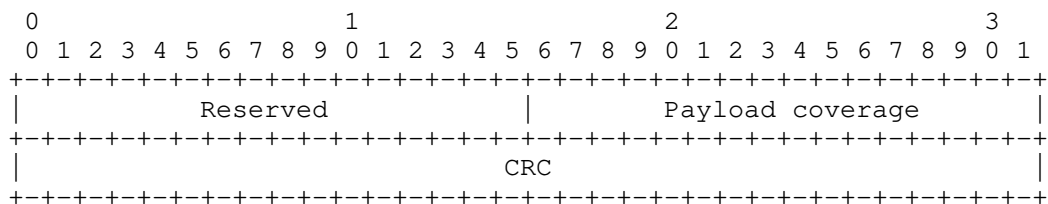
10. Alternative checksum option

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:



The fields of the option are:

- o 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).
- o 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 decapsulator 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 encapsulator 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 Identifier, 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	Fragmentation	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

intarea
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2019

P. Pfister
E. Vyncke, Ed.
Cisco
T. Pauly
Apple
D. Schinazi
Google LLC
W. Shao
Cisco
March 8, 2019

Discovering Provisioning Domain Names and Data
draft-ietf-intarea-provisioning-domains-04

Abstract

An increasing number of hosts access the Internet via multiple interfaces or, in IPv6 multi-homed networks, via multiple IPv6 prefix configurations context.

This document describes a way for hosts to identify such contexts, called Provisioning Domains (PvDs), where Fully Qualified Domain Names (FQDNs) act as PvD identifiers. Those identifiers are advertised in a new Router Advertisement (RA) option and, when present, are associated with the set of information included within the RA.

Based on this FQDN, hosts can retrieve additional information about their network access characteristics via an HTTP over TLS query. This allows applications to select which Provisioning Domains to use as well as to provide configuration parameters to the transport layer and above.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on September 9, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Terminology	4
3. Provisioning Domain Identification using Router Advertisements	4
3.1. PvD ID Option for Router Advertisements	4
3.2. Router Behavior	7
3.3. Non-PvD-aware Host Behavior	8
3.4. PvD-aware Host Behavior	8
3.4.1. DHCPv6 configuration association	9
3.4.2. DHCPv4 configuration association	9
3.4.3. Connection Sharing by the Host	9
3.4.4. Usage of DNS Servers	10
4. Provisioning Domain Additional Information	11
4.1. Retrieving the PvD Additional Information	11
4.2. Operational Consideration to Providing the PvD Additional Information	13
4.3. PvD Additional Information Format	13
4.3.1. Private Extensions	14
4.3.2. Example	14
4.4. Detecting misconfiguration and misuse	15
5. Operational Considerations	15
6. Security Considerations	17
7. Privacy Considerations	17
8. IANA Considerations	18
8.1. Additional Information PvD Keys Registry	18
8.2. PvD Option Flags Registry	18
9. Acknowledgements	18
10. References	19
10.1. Normative references	19

10.2. Informative references	20
Appendix A. Changelog	22
A.1. Version 00	22
A.2. Version 01	22
A.3. Version 02	23
A.4. WG Document version 00	23
A.5. WG Document version 01	24
A.6. WG Document version 02	24
A.7. WG Document version 04	25
Authors' Addresses	25

1. Introduction

It has become very common in modern networks for hosts to access the internet through different network interfaces, tunnels, or next-hop routers. To describe the set of network configurations associated with each access method, the concept of Provisioning Domain (PvD) was defined in [RFC7556].

This document specifies a way to identify PvDs with Fully Qualified Domain Names (FQDN), called PvD IDs. Those identifiers are advertised in a new Router Advertisement (RA) [RFC4861] option called the PvD ID Router Advertisement option which, when present, associates the PvD ID with all the information present in the Router Advertisement as well as any configuration object, such as addresses, deriving from it. The PVD ID Router Advertisement option may also contain a set of other RA options. Since such options are only considered by hosts implementing this specification, network operators may configure hosts that are 'PvD-aware' with PvDs that are ignored by other hosts.

Since PvD IDs are used to identify different ways to access the internet, multiple PvDs (with different PvD IDs) could be provisioned on a single host interface. Similarly, the same PvD ID could be used on different interfaces of a host in order to inform that those PvDs ultimately provide identical services.

This document also introduces a way for hosts to retrieve additional information related to a specific PvD by means of an HTTP over TLS query using an URI derived from the PvD ID. The retrieved JSON object contains additional information that would typically be considered unfit, or too large, to be directly included in the Router Advertisement, but might be considered useful to the applications, or even sometimes users, when choosing which PvD should be used.

2. Terminology

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

In addition, this document uses the following terminology:

Provisioning Domain (PvD): A set of network configuration information; for more information, see [RFC7556].

PvD ID: A Fully Qualified Domain Name (FQDN) used to identify a PvD.

Explicit PvD: A PvD uniquely identified with a PvD ID. For more information, see [RFC7556].

Implicit PvD: A PvD that, in the absence of a PvD ID, is identified by the host interface to which it is attached and the address of the advertising router. See also [RFC7556].

PvD-aware host A host that supports the association of network configuration information into PvDs and the use of these PvDs. Also named PvD-aware node in [RFC7556].

3. Provisioning Domain Identification using Router Advertisements

Explicit PvDs are identified by a PvD ID. The PvD ID is a Fully Qualified Domain Name (FQDN) which MUST belong to the network operator in order to avoid naming collisions. The same PvD ID MAY be used in several access networks when they ultimately provide identical services (e.g., in all home networks subscribed to the same service); else, the PvD ID MUST be different to follow section 2.4 of [RFC7556].

3.1. PvD ID Option for Router Advertisements

This document introduces a Router Advertisement (RA) option called PvD option. It is used to convey the FQDN identifying a given PvD (see Figure 1), bind the PvD ID with configuration information received over DHCPv4 (see Section 3.4.2), enable the use of HTTP over TLS to retrieve the PvD Additional Information JSON object (see Section 4), as well as contain any other RA options which would otherwise be valid in the RA.

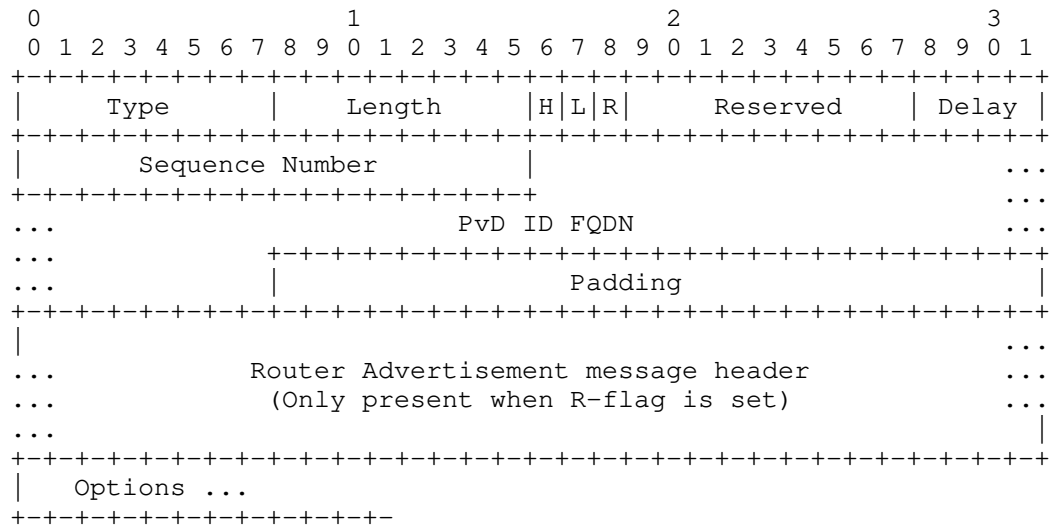


Figure 1: PvD ID Router Advertisements Option format

- Type : (8 bits) Set to 21.
- Length : (8 bits) The length of the option in units of 8 octets, including the Type and Length fields, the Router Advertisement message header, if any, as well as the RA options that are included within the PvD Option.
- H-flag : (1 bit) 'HTTP' flag stating whether some PvD Additional Information is made available through HTTP over TLS, as described in Section 4.
- L-flag : (1 bit) 'Legacy' flag stating whether the router is also providing IPv4 information using DHCPv4 (see Section 3.4.2).
- R-flag : (1 bit) 'Router Advertisement' flag stating whether the PvD Option is followed (right after padding to the next 64 bits boundary) by a Router Advertisement message header (See section 4.2 of [RFC4861]).
- Delay : (4 bits) Unsigned integer used to delay HTTP GET queries from hosts by a randomized backoff (see Section 4.1).
- Reserved : (13 bits) Reserved for later use. It MUST be set to zero by the sender and ignored by the receiver.
- Sequence Number: (16 bits) Sequence number for the PvD Additional Information, as described in Section 4.

PvD ID FQDN : The FQDN used as PvD ID encoded in DNS format, as described in Section 3.1 of [RFC1035]. Domain names compression described in Section 4.1.4 of [RFC1035] MUST NOT be used.

Padding : Zero or more padding octets to the next 8 octets boundary. It MUST be set to zero by the sender, and ignored by the receiver.

RA message header : (16 octets) When the R-flag is set, a full Router Advertisement message header as specified in [RFC4861]. The 'Type', 'Code' and 'Checksum' fields (i.e. the first 32 bits), MUST be set to zero by the sender and ignored by the receiver. The other fields are to be set and parsed as specified in [RFC4861] or any updating documents.

Options : Zero or more RA options that would otherwise be valid as part of the Router Advertisement main body, but are instead included in the PvD Option such as to be ignored by hosts that are not 'PvD-aware'.

Here is an example of a PvD option with example.org as the PvD ID FQDN and including a RDNSS and prefix information options (it also have the sequence number 123, presence of additional information to be fetched with a delay indicated as 5):

'Options' field) MUST be repeated in all the transmitted RAs. The options within the 'Options' field, MAY be transmitted only once, included in one of the transmitted PvD options.

3.3. Non-PvD-aware Host Behavior

As the PvD Option has a new option code, non-PvD-aware hosts will simply ignore the PvD Option and all the options it contains. This ensure the backward compatibility required in section 3.3 of [RFC7556]. This behavior allows for a mixed-mode network with a mix of PvD-aware and non-PvD-aware hosts coexist.

3.4. PvD-aware Host Behavior

Hosts MUST associate received RAs and included configuration information (e.g., Router Valid Lifetime, Prefix Information [RFC4861], Recursive DNS Server [RFC8106], Routing Information [RFC4191] options) with the explicit PvD identified by the first PvD Option present in the received RA, if any, or with the implicit PvD identified by the host interface and the source address of the received RA otherwise.

In case multiple PvD options are found in a given RA, hosts MUST ignore all but the first PvD option.

If a host receives PvD options flags that it does not recognize (currently in the Reserved field), it MUST ignore these flags.

Similarly, hosts MUST associate all network configuration objects (e.g., default routers, addresses, more specific routes, DNS Recursive Resolvers) with the PvD associated with the RA which last updated the object. For example, addresses that are generated using a received Prefix Information option (PIO) are associated with the PvD of the last received RA which included the given PIO.

PvD IDs MUST be compared in a case-insensitive manner (i.e., A=a), assuming ASCII with zero parity while non-alphabetic codes must match exactly (see also Section 3.1 of [RFC1035]). For example, "pvd.example.com." or "PvD.Example.coM." would refer to the same PvD.

While resolving names, executing the default address selection algorithm [RFC6724] or executing the default router selection algorithm when forwarding packets ([RFC2461], [RFC4191] and [RFC8028]), hosts MAY consider only the configuration associated with an arbitrary set of PvDs.

For example, a host MAY associate a given process with a specific PvD, or a specific set of PvDs, while associating another process

with another PvD. A PvD-aware application might also be able to select, on a per-connection basis, which PvDs should be used. In particular, constrained devices such as small battery operated devices (e.g. IoT), or devices with limited CPU or memory resources may purposefully use a single PvD while ignoring some received RAs containing different PvD IDs.

The way an application expresses its desire to use a given PvD, or a set of PvDs, or the way this selection is enforced, is out of the scope of this document. Useful insights about these considerations can be found in [I-D.kline-mif-mpvd-api-reqs].

3.4.1. DHCPv6 configuration association

When a host retrieves configuration elements using DHCPv6 (e.g., addresses or DNS recursive resolvers), they MUST be associated with the explicit or implicit PvD of the RA received on the same interface, sent from the same LLA, and with the O-flag or M-flag set [RFC4861]. If no such PvD is found, or whenever multiple different PvDs are found, the host behavior is unspecified.

This process requires hosts to keep track of received RAs, associated PvD IDs, and routers LLA; it also assumes that the router either acts as a DHCPv6 server or relay and uses the same LLA for DHCPv6 and RA traffic (which may not be the case when the router uses VRRP to send its RA).

3.4.2. DHCPv4 configuration association

When a host retrieves configuration elements from DHCPv4, they MUST be associated with the explicit PvD received on the same interface, whose PVD Options L-flag is set and, in the case of a non point-to-point link, using the same datalink address. If no such PvD is found, or whenever multiple different PvDs are found, the configuration elements coming from DHCPv4 MUST be associated with the implicit PvD identified by the interface on which the DHCPv4 transaction happened. The case of multiple explicit PvD for an IPv4 interface is undefined.

3.4.3. Connection Sharing by the Host

The situation when a host shares connectivity from an upstream interface (e.g. cellular) to a downstream interface (e.g. WiFi) is known as 'tethering'. Techniques such as ND-proxy [RFC4389], 64share [RFC7278] or prefix delegation (e.g. using DHCPv6-PD [RFC8415]) may be used for that purpose.

Whenever the RAs received from the upstream interface contain a PVD RA option, hosts that are sharing connectivity SHOULD include a PVD Option within the RAs sent downstream with:

The same PVD-ID FQDN.

The same H-bit, Delay and Sequence Number values.

The L bit set whenever the host is sharing IPv4 connectivity received from the same upstream interface.

The bits from the Reserved field set to 0.

The values of the R-bit, Router Advertisement message header and Options field depend on whether the connectivity should be shared only with PVD-aware hosts or not (see Section 3.2). In particular, all options received within the upstream PVD option and included in the downstream RA SHOULD be included in the downstream PVD option.

3.4.4. Usage of DNS Servers

PvD-aware hosts can be provisioned with recursive DNS servers via RA options passed within an explicit PVD, via RA options associated with an implicit PVD, via DHCPv6 or DHCPv4, or from some other provisioning mechanism that creates an implicit PVD (such as a VPN). In all of these cases, the DNS server addresses SHOULD be strongly associated with the corresponding PVD. Specifically, queries sent to a configured recursive DNS server SHOULD be sent from a local IP address that belongs to the matching PVD. Answers received from the DNS server SHOULD only be used on the same PVD.

Maintaining the correct usage of DNS within PVDs avoids various practical errors, such as:

A PVD associated with a VPN or otherwise private network may provide DNS answers that contain addresses inaccessible over another PVD.

A PVD that uses a NAT64 [RFC6146] and DNS64 [RFC6147] will synthesize IPv6 addresses in DNS answers that are not globally routable, and cannot be used on other PVDs. Conversely, an IPv4 address resolved via DNS on another PVD cannot be directly used on a NAT64 network without the host synthesizing an IPv6 address.

4. Provisioning Domain Additional Information

Additional information about the network characteristics can be retrieved based on the PvD ID. This set of information is called PvD Additional Information, and is encoded as a JSON object [RFC7159].

The purpose of this additional set of information is to securely provide additional information to applications about the connectivity that is provided using a given interface and source address pair. It typically includes data that would be considered too large, or not critical enough, to be provided within an RA option. The information contained in this object MAY be used by the operating system, network libraries, applications, or users, in order to decide which set of PvDs should be used for which connection, as described in Section 3.4.

4.1. Retrieving the PvD Additional Information

When the H-flag of the PvD Option is set, hosts MAY attempt to retrieve the PvD Additional Information associated with a given PvD by performing an HTTP over TLS [RFC2818] GET query to `https://<PvD-ID>/.well-known/pvd` [RFC5785]. Inversely, hosts MUST NOT do so whenever the H-flag is not set.

Note that the DNS name resolution of the PvD ID, the PKI checks as well as the actual query MUST be performed using the considered PvD. In other words, the name resolution, PKI checks, source address selection, as well as the next-hop router selection MUST be performed while using exclusively the set of configuration information attached with the PvD, as defined in Section 3.4. In some cases, it may therefore be necessary to wait for an address to be available for use (e.g., once the Duplicate Address Detection or DHCPv6 processes are complete) before initiating the HTTP over TLS query. If the host has a temporary address per [RFC4941] in this PvD, then hosts SHOULD use a temporary address to fetch the PvD Additional Information and SHOULD deprecate the used temporary address and generate a new temporary address afterward.

If the HTTP status of the answer is greater than or equal to 400 the host MUST abandon and consider that there is no additional PvD information. If the HTTP status of the answer is between 300 and 399, inclusive, it MUST follow the redirection(s). If the HTTP status of the answer is between 200 and 299, inclusive, the host MAY get a file containing a single JSON object. When a JSON object could not be retrieved, an error message SHOULD be logged and/or displayed in a rate-limited fashion.

After retrieval of the PvD Additional Information, hosts MUST keep track of the Sequence Number value received in subsequent RAs including the same PvD ID. In case the new value is greater than the value that was observed when the PvD Additional Information object was retrieved (using serial number arithmetic comparisons [RFC1982]), or whenever the validity time included in the PVD Additional Information JSON object is expired, hosts MUST either perform a new query and retrieve a new version of the object, or, failing that, deprecate the object and stop using the additional information provided in the JSON object.

Hosts retrieving a new PvD Additional Information object MUST check for the presence and validity of the mandatory fields specified in Section 4.3. A retrieved object including an expiration time that is already past or missing a mandatory element MUST be ignored.

In order to avoid synchronized queries toward the server hosting the PvD Additional Information when an object expires, object updates are delayed by a randomized backoff time.

When a host performs an object update after it detected a change in the PvD Option Sequence number, it MUST delay the query by a random time between zero and $2^{**}(\text{Delay} * 2)$ milliseconds, where 'Delay' corresponds to the 4 bits long unsigned integer in the last received PvD Option.

When a host last retrieved an object at time A including a validity time B, and is configured to keep the object up to date, it MUST perform the update at a uniformly random time in the interval $[(B-A)/2, B]$.

In the example Figure 2, the delay field value is 5, this means that host MUST delay the query by a random number between 0 and $2^{**}(5 * 2)$ milliseconds, i.e., between 0 and 1024 milliseconds.

Since the 'Delay' value is directly within the PvD Option rather than the object itself, an operator may perform a push-based update by incrementing the Sequence value while changing the Delay value depending on the criticality of the update and its PvD Additional Information servers capacity.

The PvD Additional Information object includes a set of IPv6 prefixes (under the key "prefixes") which MUST be checked against all the Prefix Information Options advertised in the RA. If any of the prefixes included in the PIO is not covered by at least one of the listed prefixes, the PvD associated with the tested prefix MUST be considered unsafe and MUST NOT be used. While this does not prevent

a malicious network provider, it does complicate some attack scenarios, and may help detecting misconfiguration.

4.2. Operational Consideration to Providing the PvD Additional Information

Whenever the H-flag is set in the PvD Option, a valid PvD Additional Information object MUST be made available to all hosts receiving the RA by the network operator. In particular, when a captive portal is present, hosts MUST still be allowed to perform DNS, PKI and HTTP over TLS operations related to the retrieval of the object, even before logging into the captive portal.

Routers MAY increment the PVD Option Sequence number in order to inform host that a new PvD Additional Information object is available and should be retrieved.

The server providing the JSON files SHOULD also check whether the client address is part of the prefixes listed into the additional information and SHOULD return a 403 responsecode if there is no match.

4.3. PvD Additional Information Format

The PvD Additional Information is a JSON object.

The following table presents the mandatory keys which MUST be included in the object:

JSON key	Description	Type	Example
name	Human-readable service name	UTF-8 string [RFC3629]	"Awesome Wifi"
expires	Date after which this object is not valid	[RFC3339]	"2017-07-23T06:00:00Z"
prefixes	Array of IPv6 prefixes valid for this PVD	Array of strings	["2001:db8:1::/48", "2001:db8:4::/48"]

A retrieved object which does not include a valid string associated with the "name" key at the root of the object, or a valid date associated with the "expires" key, also at the root of the object, MUST be ignored. In such cases, an error message SHOULD be logged

and/or displayed in a rate-limited fashion. If the PIO of the received RA is not covered by at least one of the "prefixes" key, the retrieved object SHOULD be ignored.

The following table presents some optional keys which MAY be included in the object.

JSON key	Description	Type	Example
localizedName	Localized user-visible service name, language can be selected based on the HTTP Accept-Language header in the request.	UTF-8 string	"Wifi Genial"
dnsZones	DNS zones searchable and accessible	array of DNS zones	["example.com", "sub.example.org"]
noInternet	No Internet, set when the PvD only provides restricted access to a set of services	boolean	true

It is worth noting that the JSON format allows for extensions. Whenever an unknown key is encountered, it MUST be ignored along with its associated elements.

4.3.1. Private Extensions

JSON keys starting with "x-" are reserved for private use and can be utilized to provide information that is specific to vendor, user or enterprise. It is RECOMMENDED to use one of the patterns "x-FQDN-KEY" or "x-PEN-KEY" where FQDN is a fully qualified domain name or PEN is a private enterprise number [PEN] under control of the author of the extension to avoid collisions.

4.3.2. Example

Here are two examples based on the keys defined in this section.

```
{
  "name": "Foo Wireless",
  "localizedName": "Foo-France Wifi",
  "expires": "2017-07-23T06:00:00Z",
  "prefixes" : ["2001:db8:1::/48", "2001:db8:4::/48"],
}

{
  "name": "Bar 4G",
  "localizedName": "Bar US 4G",
  "expires": "2017-07-23T06:00:00Z",
  "prefixes": ["2001:db8:1::/48", "2001:db8:4::/48"],
}
```

4.4. Detecting misconfiguration and misuse

When a host retrieves the PvD Additional Information, it MUST verify that the TLS server certificate is valid for the performed request (e.g., that the Subject Name is equal to the PvD ID expressed as an FQDN). This authentication creates a secure binding between the information provided by the trusted Router Advertisement, and the HTTPS server. However, this does not mean the Advertising Router and the PvD server belong to the same entity.

Hosts MUST verify that all prefixes in the RA PIO are covered by a prefix from the PvD Additional Information. An adversarial router willing to fake the use of a given explicit PvD, without any access to the actual PvD Additional Information, would need to perform NAT66 in order to circumvent this check.

It is also RECOMMENDED that the HTTPS server checks the IPv6 source addresses of incoming connections (see Section 4.1). This check give reasonable assurance that neither NPTv6 [RFC6296] nor NAT66 were used and restricts the information to the valid network users.

Note that this check cannot be performed when the HTTPS query is performed over IPv4. Therefore, the PvD ID FQDN SHOULD NOT have a DNS A record whenever all hosts using the given PvD have IPv6 connectivity.

5. Operational Considerations

This section describes some use cases of PvD. For the sake of simplicity, the RA messages will not be described in the usual ASCII art but rather in an indented list. For example, a RA message containing some options and a PvD option that also contains other options will be described as:

- o RA Header: router lifetime = 6000
- o Prefix Information Option: length = 4, prefix = 2001:db8:cafe::/64
- o PvD Option header: length = 3 + 5 + 4 , PvD ID FQDN = example.org., R-flag = 0 (actual length of the header with padding 24 bytes = 3 * 8 bytes)
 - * Recursive DNS Server: length = 5, addresses= [2001:db8:cafe::53, 2001:db8:f00d::53]
 - * Prefix Information Option: length = 4, prefix = 2001:db8:f00d::/64

It is expected that for some years, networks will have a mixed environment of PvD-aware hosts and non-PvD-aware hosts. If there is a need to give specific information to PvD-aware hosts only, then it is recommended to send TWO RA messages: one for each class of hosts. For example, here is the RA for non-PvD-aware hosts:

- o RA Header: router lifetime = 6000 (non-PvD-aware hosts will use this router as a default router)
- o Prefix Information Option: length = 4, prefix = 2001:db8:cafe::/64
- o Recursive DNS Server Option: length = 3, addresses= [2001:db8:cafe::53]
- o PvD Option header: length = 3 + 2, PvD ID FQDN = foo.example.org., R-flag = 1 (actual length of the header 24 bytes = 3 * 8 bytes)
 - * RA Header: router lifetime = 0 (PvD-aware hosts will not use this router as a default router), implicit length = 2

And here is a RA example for PvD-aware hosts:

- o RA Header: router lifetime = 0 (non-PvD-aware hosts will not use this router as a default router)
- o PvD Option header: length = 3 + 2 + 4 + 3, PvD ID FQDN = example.org., R-flag = 1 (actual length of the header 24 bytes = 3 * 8 bytes)
 - * RA Header: router lifetime = 1600 (PvD-aware hosts will use this router as a default router), implicit length = 2
 - * Prefix Information Option: length = 4, prefix = 2001:db8:f00d::/64

- * Recursive DNS Server Option: length = 3, addresses=[2001:db8:f00d::53]

In the above example, non-PvD-aware hosts will only use the first RA sent from their default router and using the 2001:db8:cafe::/64 prefix. PvD-aware hosts will autonomously configure addresses from both PIOs, but will only use the source address in 2001:db8:f00d::/64 to communicate past the first hop router since only the router sending the second RA will be used as default router; similarly, they will use the DNS server 2001:db8:f00d::53 when communicating with this address.

6. Security Considerations

Although some solutions such as IPsec or SeND [RFC3971] can be used in order to secure the IPv6 Neighbor Discovery Protocol, in practice actual deployments largely rely on link layer or physical layer security mechanisms (e.g. 802.1x [IEEE8021X]) in conjunction with RA Guard [RFC6105].

This specification does not improve the Neighbor Discovery Protocol security model, but extends the purely link-local trust relationship between the host and the default routers with HTTP over TLS communications which servers are authenticated as rightful owners of the FQDN received within the trusted PvD ID RA option.

It must be noted that Section 4.4 of this document only provides reasonable assurance against misconfiguration but does not prevent an hostile network access provider to advertize wrong information that could lead applications or hosts to select an hostile PvD. Users should always apply caution when connecting to an unknown network.

7. Privacy Considerations

Retrieval of the PvD Additional Information over HTTPS requires early communications between the connecting host and a server which may be located further than the first hop router. Although this server is likely to be located within the same administrative domain as the default router, this property can't be ensured. Therefore, hosts willing to retrieve the PvD Additional Information before using it without leaking identity information, SHOULD make use of an IPv6 Privacy Address and SHOULD NOT include any privacy sensitive data, such as User Agent header or HTTP cookie, while performing the HTTP over TLS query.

From a privacy perspective, retrieving the PvD Additional Information is not different from establishing a first connection to a remote server, or even performing a single DNS lookup. For example, most

operating systems already perform early queries to well known web sites, such as `http://captive.example.com/hotspot-detect.html`, in order to detect the presence of a captive portal.

There may be some cases where hosts, for privacy reasons, should refrain from accessing servers that are located outside a certain network boundary. In practice, this could be implemented as a whitelist of 'trusted' FQDNs and/or IP prefixes that the host is allowed to communicate with. In such scenarios, the host SHOULD check that the provided PvD ID, as well as the IP address that it resolves into, are part of the allowed whitelist.

8. IANA Considerations

Upon publication of this document, IANA is asked to remove the 'reclaimable' tag off the value 21 for the PvD option (from the IPv6 Neighbor Discovery Option Formats registry).

IANA is asked to assign the value "pvd" from the Well-Known URIs registry.

8.1. Additional Information PvD Keys Registry

IANA is asked to create and maintain a new registry called "Additional Information PvD Keys", which will reserve JSON keys for use in PvD additional information. The initial contents of this registry are given in Section 4.3.

New assignments for Additional Information PvD Keys Registry will be administered by IANA through Expert Review RFC8126 [RFC8126].

8.2. PvD Option Flags Registry

IANA is also asked to create and maintain a new registry entitled "PvD Option Flags" reserving bit positions from 0 to 15 to be used in the PvD option bitmask. Bit position 0, 1 and 2 are reserved by this document (as specified in Figure 1). Future assignments require Standards Action RFC8126 [RFC8126], via a Standards Track RFC document.

9. Acknowledgements

Many thanks to M. Stenberg and S. Barth for their earlier work: [I-D.stenberg-mif-mpvd-dns], as well as to Basile Bruneau who was author of an early version of this document.

Thanks also to Marcus Keane, Mikael Abrahamsson, Ray Bellis, Zhen Cao, Tim Chow, Lorenzo Colitti, Michael Di Bartolomeo, Ian Farrer,

Bob Hinden, Tatuya Jinmei, Erik Kline, Ted Lemon, Jen Lenkova, Veronika McKillop, Mark Townsley and James Woodyatt for useful and interesting discussions.

Finally, special thanks to Thierry Danis and Wenqin Shao for their valuable inputs and implementation efforts ([github]), Tom Jones for his integration effort into the NEAT project and Rigil Salim for his implementation work.

10. References

10.1. Normative references

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <<https://www.rfc-editor.org/info/rfc1982>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2461] Narten, T., Nordmark, E., and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, DOI 10.17487/RFC2461, December 1998, <<https://www.rfc-editor.org/info/rfc2461>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <<https://www.rfc-editor.org/info/rfc4861>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

10.2. Informative references

- [github] Cisco, "IPv6-mPvD github repository", <<https://github.com/IPv6-mPvD>>.
- [I-D.kline-mif-mpvd-api-reqs] Kline, E., "Multiple Provisioning Domains API Requirements", draft-kline-mif-mpvd-api-reqs-00 (work in progress), November 2015.
- [I-D.stenberg-mif-mpvd-dns] Stenberg, M. and S. Barth, "Multiple Provisioning Domains using Domain Name System", draft-stenberg-mif-mpvd-dns-00 (work in progress), October 2015.
- [IEEE8021X] IEEE, "IEEE Standards for Local and Metropolitan Area Networks: Port based Network Access Control, IEEE Std".
- [PEN] IANA, "Private Enterprise Numbers", <<https://www.iana.org/assignments/enterprise-numbers>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <<https://www.rfc-editor.org/info/rfc3971>>.
- [RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", RFC 4191, DOI 10.17487/RFC4191, November 2005, <<https://www.rfc-editor.org/info/rfc4191>>.
- [RFC4389] Thaler, D., Talwar, M., and C. Patel, "Neighbor Discovery Proxies (ND Proxy)", RFC 4389, DOI 10.17487/RFC4389, April 2006, <<https://www.rfc-editor.org/info/rfc4389>>.
- [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <<https://www.rfc-editor.org/info/rfc4941>>.

- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC6105] Levy-Abegnoli, E., Van de Velde, G., Popoviciu, C., and J. Mohacsi, "IPv6 Router Advertisement Guard", RFC 6105, DOI 10.17487/RFC6105, February 2011, <<https://www.rfc-editor.org/info/rfc6105>>.
- [RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <<https://www.rfc-editor.org/info/rfc6146>>.
- [RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <<https://www.rfc-editor.org/info/rfc6147>>.
- [RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011, <<https://www.rfc-editor.org/info/rfc6296>>.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <<https://www.rfc-editor.org/info/rfc6724>>.
- [RFC7278] Byrne, C., Drown, D., and A. Vizdal, "Extending an IPv6 /64 Prefix from a Third Generation Partnership Project (3GPP) Mobile Interface to a LAN Link", RFC 7278, DOI 10.17487/RFC7278, June 2014, <<https://www.rfc-editor.org/info/rfc7278>>.
- [RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015, <<https://www.rfc-editor.org/info/rfc7556>>.
- [RFC8028] Baker, F. and B. Carpenter, "First-Hop Router Selection by Hosts in a Multi-Prefix Network", RFC 8028, DOI 10.17487/RFC8028, November 2016, <<https://www.rfc-editor.org/info/rfc8028>>.

- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.
- [RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <<https://www.rfc-editor.org/info/rfc8415>>.

Appendix A. Changelog

Note to RFC Editors: Remove this section before publication.

A.1. Version 00

Initial version of the draft. Edited by Basile Bruneau + Eric Vyncke and based on Basile's work.

A.2. Version 01

Major rewrite intended to focus on the the retained solution based on corridors, online, and WG discussions. Edited by Pierre Pfister. The following list only includes major changes.

PvD ID is an FQDN retrieved using a single RA option. This option contains a sequence number for push-based updates, a new H-flag, and a L-flag in order to link the PvD with the IPv4 DHCP server.

A lifetime is included in the PvD ID option.

Detailed Hosts and Routers specifications.

Additional Information is retrieved using HTTP-over-TLS when the PvD ID Option H-flag is set. Retrieving the object is optional.

The PvD Additional Information object includes a validity date.

DNS-based approach is removed as well as the DNS-based encoding of the PvD Additional Information.

Major cut in the list of proposed JSON keys. This document may be extended later if need be.

Monetary discussion is moved to the appendix.

Clarification about the 'prefixes' contained in the additional information.

Clarification about the processing of DHCPv6.

A.3. Version 02

The FQDN is now encoded with ASCII format (instead of DNS binary) in the RA option.

The PvD ID option lifetime is removed from the object.

Use well known URI "https://<PvD-ID>/well-known/pvd"

Reference RFC3339 for JSON timestamp format.

The PvD ID Sequence field has been extended to 16 bits.

Modified host behavior for DHCPv4 and DHCPv6.

Removed IKEv2 section.

Removed mention of RFC7710 Captive Portal option. A new I.D. will be proposed to address the captive portal use case.

A.4. WG Document version 00

Document has been accepted as INTAREA working group document

IANA considerations follow RFC8126 [RFC8126]

PvD ID FQDN is encoded as per RFC 1035 [RFC1035]

PvD ID FQDN is prepended by a one-byte length field

Marcus Keane added as co-author

dnsZones key is added back

draft of a privacy consideration section and added that a temporary address should be used to retrieve the PvD additional information

per Bob Hinden's request: the document is now aiming at standard track and security considerations have been moved to the main section

A.5. WG Document version 01

Removing references to 'metered' and 'characteristics' keys. Those may be in scope of the PvD work, but this document will focus on essential parts only.

Removing appendix section regarding link quality and billing information.

The PvD RA Option may now contain other RA options such that PvD-aware hosts may receive configuration information otherwise invisible to non-PvD-aware hosts.

Clarify that the additional PvD Additional Information is not intended to modify host's networking stack behavior, but rather provide information to the Application, used to select which PvDs must be used and provide configuration parameters to the transport layer.

The RA option padding is used to increase the option size to the next 64 (was 32) bits boundary.

Better detail the Security model and Privacy considerations.

A.6. WG Document version 02

Use the IANA value of 21 in the text and update the IANA considerations section accordingly

add the Delay field to avoid the thundering herd effect

add Wenqin Shao as author

keep the 1 PvD per RA model

changed the intro (per Zhen Cao) "when choosing which PvD and transport should be used" => "when choosing which PvD should be used"

rename A-flag in R-flag to avoid A-flag of PIO

use the wording "PvD Option", removing the ID token as it is now a container with more than just an ID, removing 'RA' in the option name to be consistent with other IANA NDP option

use "non-PvD-aware" rather than "PvD-ignorant"

added more reference to RFC 7556 (notably for PvD being globally unique, introducing PvD-aware host vs. PvD-aware node)

Section 3.4.3 renamed from "interconnection shared by node" to 'connection shared by node"

Section 3.4 renamed into "PvD-aware Host Behavior"

Added a section "Non-PvD-aware Host Behavior"

A.7. WG Document version 04

Updated reference for DHCPv6-PD from RFC 3633 to RFC 8415.

Enhanced IANA considerations to clarify review process and new registries.

Added a section on considerations for handling DNS on a PvD-aware host.

Authors' Addresses

Pierre Pfister
Cisco
11 Rue Camille Desmoulins
Issy-les-Moulineaux 92130
France

Email: ppfister@cisco.com

Eric Vyncke (editor)
Cisco
De Kleetlaan, 6
Diegem 1831
Belgium

Email: evyncke@cisco.com

Tommy Pauly
Apple

Email: tpaully@apple.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, California 94043
USA

Email: dschinazi.ietf@gmail.com

Wenqin Shao
Cisco
11 Rue Camille Desmoulins
Issy-les-Moulineaux 92130
France

Email: wenshao@cisco.com

INTAREA WG
Internet-Draft
Updates: 8335 (if approved)
Intended status: Standards Track
Expires: August 5, 2019

M. Nayak
R. Bonica
R. Puttur
Juniper Networks
February 1, 2019

Probing IP Interfaces By Virtual Function Index
draft-nayak-intarea-probe-by-vfi-00

Abstract

This document enhances the PROBE diagnostic tool so that it can identify the probed interface by Virtual Function Index. In order to achieve that goal, this document also extends the Interface Identification Object. The Interface Identification Object is an ICMP Extension Object class.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on August 5, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Problem Statement	2
2. Requirements Language	2
3. ICMP Extended Echo Request Message	3
4. Interface Identification Object	3
5. ICMP Extended Echo Reply Message	4
6. ICMP Message Processing	4
7. Updates To RFC 8335	4
8. IANA Considerations	5
9. Security Considerations	5
10. Acknowledgements	5
11. References	5
11.1. Normative References	5
11.2. Informative References	5
Authors' Addresses	6

1. Problem Statement

PROBE [RFC8335] is a diagnostic tool that can be used to query the status of an interface. PROBE sends an ICMP Extended Echo Request message to a proxy interface. The ICMP Extended Echo Request message contains an ICMP Extension Structure and the ICMP Extension Structure contains an Interface Identification Object. The Interface Identification Object identifies the probed interface by name, ifIndex or address.

When the proxy interface receives the ICMP Extended Echo Request, the node upon which it resides executes access control procedures. If access is granted, the node determines the status of the probed interface and returns an ICMP Extended Echo Reply message. The ICMP Extended Echo Reply indicates the status of the probed interface.

This document enhances the PROBE so that it can identify the probed interface by Virtual Function Index (VFI) [SR-IOV]. In order to achieve that goal, this document extends the Interface Identification Object. The Interface Identification Object is an ICMP Extension Object class.

2. Requirements Language

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

14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. ICMP Extended Echo Request Message

Section 2.1 of [RFC8335] defines the ICMP Extended Echo Request message. As per [RFC8335], the ICMP Extended Echo Request message contains the following fields:

- o Type
- o Code
- o Checksum
- o Identifier
- o Reserved
- o L (local)
- o ICMP Extension Structure

Section 7 of [RFC4884] defines the ICMP Extension Structure. As per [RFC4884], the Extension Structure contains exactly one Extension Header followed by one or more objects. When applied to the ICMP Extended Echo Request message, the ICMP Extension Structure contains exactly one instance of the Interface Identification Object. Section 2.1 of [RFC8335] defines the Interface Identification Object. Section 4 of this document extends that definition.

If the L-bit is set, the Interface Identification Object can identify the probed interface by name, index, address or VFI. If the L-bit is clear, the Interface Identification Object identifies the probed interface by address.

4. Interface Identification Object

Section 2.1 of [RFC8335] defines the Interface Identification Object. The Interface Identification Object identifies the probed interface by name, index, or address. Like any other ICMP Extension Object, it contains an Object Header and Object Payload. The Object Header contains the following fields:

- o Class-Num: Interface Identification Object. The value is 3.
- o C-Type: Determines how the probed interface is identified.

- o Length: Length of the object, measured in octets, including the Object Header and Object Payload.

Currently, the following values are defined for C-Type:

- o (0) Reserved
- o (1) Identifies Interface by Name
- o (2) Identifies Interface by Index
- o (3) Identifies Interface by Address

This document defines the following, new C-Type:

- o (value TBD by IANA) Identifies Interfaces by Virtual Function Index (VFI)

If the Interface Identification Object identifies the probed interface by Virtual Function Index, the length is equal to 8 and the payload contains the Virtual Function Index.

5. ICMP Extended Echo Reply Message

Section 3 of [RFC8335] defines the ICMP Extended Echo Reply message. This document does not change that definition.

6. ICMP Message Processing

Section 4 of [RFC8335] defines the ICMP message processing. This document does not change that definition.

7. Updates To RFC 8335

Section 2 of [RFC8335] states:

"If the L-bit is set, the Interface Identification Object can identify the probed interface by name, index, or address. If the L-bit is clear, the Interface Identification Object MUST identify the probed interface by address."

This document updates that text as follows:

"If the L-bit is set, the Interface Identification Object can identify the probed interface by name, index, address, or Virtual Function Index (VFI). If the L-bit is clear, the Interface Identification Object MUST identify the probed interface by address."

8. IANA Considerations

IANA is requested to add the following a new C-type:

- o (value TBD by IANA) Identifies Interfaces by Virtual Function Index (VFI)

This new C-Type is to be added to the Interface Identification Object under the "ICMP Extension Object Classes and Class Sub-types" registry.

9. Security Considerations

This document neither extends nor mitigates any of the security considerations mentioned in [RFC8335].

10. Acknowledgements

The authors wish to acknowledge Ross Callon for his helpful comments.

11. References

11.1. 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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", RFC 4884, DOI 10.17487/RFC4884, April 2007, <<https://www.rfc-editor.org/info/rfc4884>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8335] Bonica, R., Thomas, R., Linkova, J., Lenart, C., and M. Boucadair, "PROBE: A Utility for Probing Interfaces", RFC 8335, DOI 10.17487/RFC8335, February 2018, <<https://www.rfc-editor.org/info/rfc8335>>.

11.2. Informative References

[SR-IOV] PCI-SIG, ., "Single Root I/O Virtualization and Sharing Specification Revision 1.1", January 2010, <<https://members.pcisig.com/wg/PCI-SIG/document/download/8238>>.

Authors' Addresses

Manoj Nayak
Juniper Networks
Bangalore, KA 560103
India

Email: manojnayak@juniper.net

Ron Bonica
Juniper Networks
Herndon, Virginia 20171
USA

Email: rbonica@juniper.net

Rafik Puttur
Juniper Networks
Bangalore, KA 560103
India

Email: rafikp@juniper.net

Internet Area Working Group
Internet-Draft
Intended status: Experimental
Expires: September 12, 2019

V. Olteanu
D. Niculescu
University Politehnica of Bucharest
March 11, 2019

SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-06

Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Revision log	4
2. Requirements language	8
3. Mode of operation	8
4. Requests	10
5. Version Mismatch Replies	12
6. Authentication Replies	12
7. Operation Replies	13
7.1. Handling CONNECT	14
7.2. Handling BIND	15
7.3. Handling UDP ASSOCIATE	15
7.3.1. Proxying UDP servers	17
8. SOCKS Options	17
8.1. Stack options	17
8.1.1. IP TOS options	19
8.1.2. TFO options	19
8.1.3. Multipath TCP options	20
8.1.4. MPTCP Scheduler options	20
8.1.5. Listen Backlog options	21
8.2. Authentication Method options	22
8.3. Authentication Data options	23
8.4. Session options	24
8.4.1. Session initiation	24
8.4.2. Further SOCKS Requests	26
8.4.3. Tearing down the session	27
8.5. Idempotence options	28
8.5.1. Requesting a fresh token window	29
8.5.2. Spending a token	30
8.5.3. Handling Token Window Advertisements	32
9. Username/Password Authentication	32
10. TCP Fast Open on the Client-Proxy Leg	32
11. False Starts	33
12. Security Considerations	33
12.1. Large requests	33
12.2. Replay attacks	34
12.3. Resource exhaustion	34
13. IANA Considerations	34
14. Acknowledgements	34
15. References	35
15.1. Normative References	35
15.2. Informative References	35

Authors' Addresses	35
--------------------	----

1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satellite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [RFC8446] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- o The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.
- o The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.
- o The protocol can be extended via options without breaking backward-compatibility.

- o The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-06

- o Session options
- o Options now have a 2-byte length field.
- o Stack options
 - * Stack options can no longer contain duplicate information.
 - * TFO: Better payload size semantics
 - * TOS: Added missing code field.
 - * MPTCP Scheduler options:
 - + Removed support for round-robin
 - + "Default" renamed to "Lowest latency first"
 - * Listen Backlog options: now tied to sessions, instead of an authenticated user
- o Idempotence options
 - * Now used in the context of a session (no longer tied to an authenticated user)
 - * Idempotence options have a different codepoint: 0x05. (Was 0x04.)
 - * Clarified that implementations that support Idempotence Options must support all Idempotence Option Types.
 - * Shifted Idempotence Option Types by 1. (Makes implementation easier.)
- o Shrunk vendor-specific option range to 32 (down from 64).
- o Removed reference to dropping initial data. (It could no longer be done as of -05.)

- o Initial data size capped at 16KB.
- o Application data is never encrypted by SOCKS 6. (It can still be encrypted by the TLS layer under SOCKS.)
- o Messages now carry the total length of the options, rather than the number of options. Limited options length to 16KB.
- o Security Considerations
 - * Updated the section to reflect the smaller maximum message size.
 - * Added a subsection on resource exhaustion.

draft-05

- o Limited the "slow" authentication negotiations to one (and Authentication Replies to 2)
- o Revamped the handling of the first bytes in the application data stream
 - * False starts are now recommended. (Added the "False Start" section.)
 - * Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.
 - * Initial data size capped at 2^{13} . Initial data can no longer be dropped by the proxy.
 - * The TFO option can hint at the desired SYN payload size.
- o Request: clarified the meaning of the Address and Port fields.
- o Better reverse TCP proxy support: optional listen backlog for TCP BIND
- o TFO options can no longer be placed inside Operation Replies.
- o IP TOS stack option
- o Suggested a range for vendor-specific options.
- o Revamped UDP functionality

- * Now using fixed UDP ports

- * DTLS support

- o Stack options: renamed Proxy-Server leg to Proxy-Remote leg

draft-04

- o Moved Token Expenditure Replies to the Authentication Reply.

- o Shifted the Initial Data Size field in the Request, in order to make it easier to parse.

draft-03

- o Shifted some fields in the Operation Reply to make it easier to parse.

- o Added connection attempt timeout response code to Operation Replies.

- o Proxies send an additional Authentication Reply after the authentication phase. (Useful for token window advertisements.)

- o Renamed the section "Connection Requests" to "Requests"

- o Clarified the fact that proxies don't need to support any command in particular.

- o Added the section "TCP Fast Open on the Client-Proxy Leg"

- o Options:

- * Added constants for option kinds

- * Salt options removed, along with the relevant section from Security Considerations. (TLS 1.3 Makes AEAD mandatory.)

- * Limited Authentication Data options to one per method.

- * Relaxed proxy requirements with regard to handling multiple Authentication Data options. (When the client violates the above bullet point.)

- * Removed interdependence between Authentication Method and Authentication Data options.

- * Clients SHOULD omit advertising the "No authentication required" option. (Was MAY.)
- * Idempotence options:
 - + Token Window Advertisements are now part of successful Authentication Replies (so that the proxy-server RTT has no impact on their timeliness).
 - + Proxies can't advertise token windows of size 0.
 - + Tweaked token expenditure response codes.
 - + Support no longer mandatory on the proxy side.
- * Revamped Socket options
 - + Renamed Socket options to Stack options.
 - + Banned contradictory socket options.
 - + Added socket level for generic IP. Removed the "socket" socket level.
 - + Stack options no longer use option codes from `setsockopt()`.
 - + Changed MPTCP Scheduler constants.

draft-02

- o Made support for Idempotence options mandatory for proxies.
- o Clarified what happens when proxies can not or will not issue tokens.
- o Limited token windows to $2^{31} - 1$.
- o Fixed definition of "less than" for tokens.
- o NOOP commands now trigger Operation Replies.
- o Renamed Authentication options to Authentication Data options.
- o Authentication Data options are no longer mandatory.
- o Authentication methods are now advertised via options.
- o Shifted some Request fields.

- o Option range for vendor-specific options.
- o Socket options.
- o Password authentication.
- o Salt options.

draft-01

- o Added this section.
- o Support for idempotent commands.
- o Removed version numbers from operation replies.
- o Request port number for SOCKS over TLS. Deprecate encryption/encapsulation within SOCKS.
- o Added Version Mismatch Replies.
- o Renamed the AUTH command to NOOP.
- o Shifted some fields to make requests and operation replies easier to parse.

2. 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].

3. Mode of operation

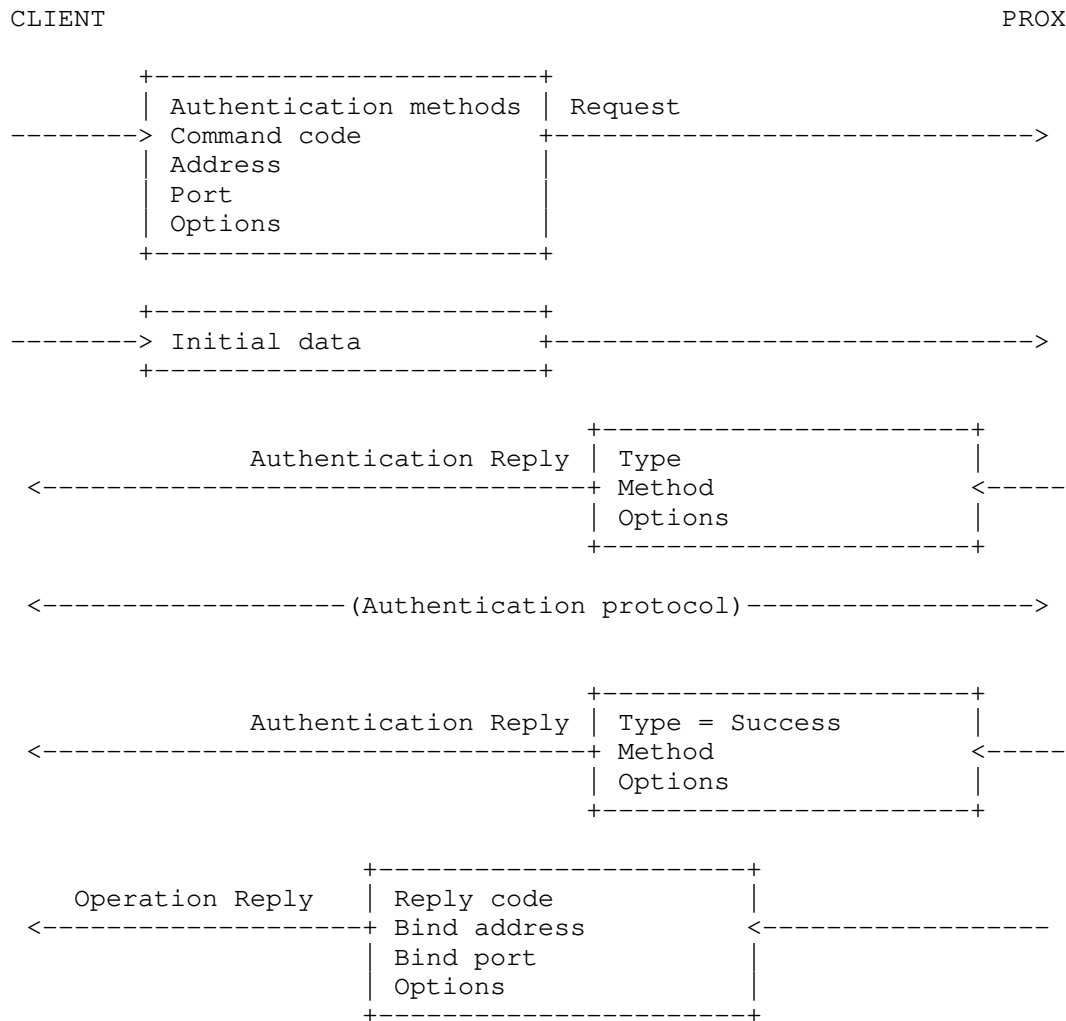


Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the SOCKS proxy. The client then enters a negotiation phase, by sending the request in Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy

indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational connection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Requests

The client starts by sending a request to the proxy.

Version		Command	Port	Address	Address
Major	Minor	Code		Type	
1	1	1	2	1	Variable
Options		Options			
Length					
2		Variable			

Figure 2: SOCKS 6 Request

- o Version: The major byte is 0x06, and the minor byte is 0x00.
- o Command Code:
 - * 0x00 NOOP: does nothing.
 - * 0x01 CONNECT: requests the establishment of a TCP connection. TFO MUST NOT be used unless explicitly requested.
 - * 0x02 BIND: requests the establishment of a TCP port binding.

- * 0x03 UDP ASSOCIATE: requests a UDP port association.
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Address: this field's format depends on the address type:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address
- o Port: the port in network byte order.
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

The Address and Port fields have different meanings based on the Command Code:

- o NOOP: The fields have no meaning. The Address Type field MUST be either 0x01 (IPv4) or 0x04 (IPv6). The Address and Port fields MUST be 0.
- o CONNECT: The fields signify the address and port to which the client wishes to connect.
- o BIND, UDP ASSOCIATE: The fields indicate the desired bind address and port. If the client does not require a certain address, it can set the Address Type field to 0x01 (IPv4) or 0x04 (IPv6), and the Address field to 0. Likewise, if the client does not require a certain port, it can set the Port field to 0.

Clients can advertise their supported authentication methods by including an Authentication Method option (see Section 8.2).

5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6.0, the proxy sends the following response:

Version	
Major	Minor
1	1

Figure 3: SOCKS 6 Version Mismatch Reply

- o Version: The major byte is 0x06, and the minor byte is 0x00.

A client **MUST** close the connection after receiving such a reply.

6. Authentication Replies

Upon receipt of a valid request, the proxy sends an Authentication Reply:

Version		Type	Method	Options Length	Options
Major	Minor				
1	1	1	1	2	Variable

Figure 4: SOCKS 6 Authentication Reply

- o Version: The major byte is 0x06, and the minor byte is 0x00.
- o Type:
 - * 0x00: authentication successful.
 - * 0x01: further authentication needed.
- o Method: The chosen authentication method.
- o Options Length: the total size of the SOCKS options that appear in the Options field. **MUST NOT** exceed 16KB.
- o Options: see Section 8.

Multihomed clients SHOULD cache the chosen method on a per-interface basis and SHOULD NOT include Authentication Data options related to any other methods in further requests originating from the same interface.

If the server signals that further authentication is needed and selects "No Acceptable Methods", the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an Authentication Data option. Application data is not subject to any encryption negotiated during this phase. Descriptions of such negotiations are beyond the scope of this memo.

When the negotiation is complete (either successfully or unsuccessfully), the proxy sends a second Authentication Reply. The second Authentication Reply MUST either signal success or that there are no more acceptable authentication methods.

7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:

Reply Code	Bind Port	Address Type	Bind Address	Options Length	Options
1	2	1	Variable	2	Variable

Figure 5: SOCKS 6 Operation Reply

o Reply Code:

- * 0x00: Success
- * 0x01: General SOCKS server failure
- * 0x02: Connection not allowed by ruleset
- * 0x03: Network unreachable
- * 0x04: Host unreachable

- * 0x05: Connection refused
- * 0x06: TTL expired
- * 0x07: Command not supported
- * 0x08: Address type not supported
- * 0x09: Connection attempt timed out
- o Bind Port: the proxy bound port in network byte order.
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Bind Address: the proxy bound address in the following format:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

Proxy implementations MAY support any subset of the client commands listed in Section 4.

If the proxy returns a reply code other than "Success", the client MUST close the connection.

If the client issued an NOOP command, the client MUST close the connection after receiving the Operation Reply.

7.1. Handling CONNECT

In case the client has issued a CONNECT request, data can now pass.

7.2. Handling BIND

In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

7.3. Handling UDP ASSOCIATE

Proxies offering UDP functionality must be configured with a UDP port used for relaying UDP datagrams to and from the client, and/or a port used for relaying datagrams over DTLS.

Following a successful Operation Reply, the proxy sends a UDP Association Initialization message:

+-----+	
Association ID	
+-----+	
4	
+-----+	

Figure 6: UDP Association Initialization

- o Association ID: the identifier of the UDP association

Proxy implementations SHOULD generate Association IDs randomly or pseudo-randomly.

Clients may start sending UDP datagrams to the proxy either in plaintext, or over an established DTLS session, using the proxy's configured UDP ports. A client's datagrams are prefixed by a SOCKS Datagram Header, indicating the remote host's address and port:

Version		Association ID	Port	Address Type	Address
Major	Minor				
1	1	4	2	1	Variable

Figure 7: SOCKS 6 Datagram Header

- o Version: The major byte is 0x06, and the minor byte is 0x00.

- o Association ID: the identifier of the UDP association
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Address: this field's format depends on the address type:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address
- o Port: the port in network byte order.

Following the receipt of the first datagram from the client, the proxy makes a one-way mapping between the Association ID and:

- o the 5-tuple of the UDP conversation, if the datagram was received over plain UDP, or
- o the DTLS connection, if the datagram was received over DTLS. The DTLS connection is identified either by its 5-tuple, or some other mechanism, like [I-D.ietf-tls-dtls-connection-id].

Further datagrams carrying the same Association ID, but not matching the established mapping, are silently dropped.

The proxy then sends an UDP Association Confirmation message over the TCP connection with the client:

```
+-----+
| Status |
+-----+
|   1    |
+-----+
```

Figure 8: UDP Association Confirmation

- o Status: MUST be 0x00

Following the confirmation message, UDP packets bound for the proxy's bind address and port are relayed to the client, also prefixed by a Datagram Header.

The UDP association remains active for as long as the TCP connection between the client and the proxy is kept open.

7.3.1. Proxying UDP servers

Under some circumstances (e.g. when hosting a server), the SOCKS client expects the remote host to send UDP datagrams first. As such, the SOCKS client must trigger a UDP Association Confirmation without having the proxy relay any datagrams on its behalf.

To that end, it sends an empty datagram prefixed by a Datagram Header with an IP address and port consisting of zeroes. The client SHOULD resend the empty datagram if an UDP Association Confirmation is not received after a timeout.

8. SOCKS Options

SOCKS options have the following format:

Kind	Length	Option Data
1	2	Variable

Figure 9: SOCKS 6 Option

- o Kind: Allocated by IANA. (See Section 13.)
- o Length: The total length of the option.
- o Option Data: The contents are specific to each option kind.

Unless otherwise noted, client and proxy implementations MAY omit supporting any of the options described in this document. Upon encountering an unsupported option, a SOCKS endpoint MUST silently ignore it.

8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well the protocols used by the proxy to communicate with the remote host (i.e. IP, TCP,

UDP). A Stack option can affect either the proxy's protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY include Stack options in their Operation Replies to signal their behavior. Said options MAY be unsolicited, i. e. the proxy MAY send them to signal behaviour that was not explicitly requested by the client.

In case of UDP ASSOCIATE, the stack options refer to the UDP traffic relayed by the proxy.

Stack options that are part of the same message MUST NOT contradict one another or contain duplicate information.

Kind	Length	Leg	Level	Code	Data
1	2	2 bits	6 bits	1	Variable

Figure 10: Stack Option

- o Kind: 0x01 (Stack option)
- o Length: The length of the option.
- o Leg:
 - * 0x1: Client-Proxy Leg
 - * 0x2: Proxy-Remote Leg
 - * 0x3: Both Legs
- o Level:
 - * 0x01: IP: options that apply to either IPv4 or IPv6
 - * 0x02: IPv4
 - * 0x03: IPv6
 - * 0x04: TCP
 - * 0x05: UDP

- o Code: Option code
- o Data: Option-specific data

8.1.1. IP TOS options

Kind	Length	Leg	Level	Code	TOS
1	2	2 bits	6 bits	1	1

Figure 11: IP TOS Option

- o Kind: 0x01 (Stack option)
- o Length: 4
- o Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Remote or Both legs)
- o Level: 0x04 (TCP).
- o Code: 0x01
- o TOS: The IP TOS code

The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.

8.1.2. TFO options

Kind	Length	Leg	Level	Code	Payload Size
1	2	2 bits	6 bits	1	2

Figure 12: TFO Option

- o Kind: 0x01 (Stack option)
- o Length: 4
- o Leg: 0x02 (Proxy-Remote leg).

- o Level: 0x04 (TCP).
- o Code: 0x01
- o Payload Size: The desired payload size of the TFO SYN. Ignored in case of a BIND command.

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the client can indicate the desired payload size of the SYN. If the field is 0, the proxy can use an arbitrary payload size. If the field is non-zero, the proxy MUST NOT use a payload size larger than the one indicated. The proxy MAY use a smaller payload size than the one indicated.

8.1.3. Multipath TCP options

In case of a CONNECT command, the proxy can inform the client that the connection to the server is an MPTCP connection.

Kind	Length	Leg	Level	Code
1	2	2 bits	6 bits	1

Figure 13: Multipath TCP Option

- o Kind: 0x01 (Stack option)
- o Length: 4
- o Leg: 0x02 (Proxy-Remote leg)
- o Level: 0x04 (TCP).
- o Code: 0x02

8.1.4. MPTCP Scheduler options

In case of a CONNECT or BIND command, a client can use an MPTCP Scheduler option to indicate its preferred scheduler for the connection.

A proxy can use an MPTCP Scheduler option to inform the client about what scheduler is in use.

Kind	Length	Leg	Level	Code	Scheduler
1	2	2 bits	6 bits	1	1

Figure 14: MPTCP Scheduler Option

- o Kind: 0x01 (Stack option)
- o Length: 5
- o Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Remote or Both legs).
- o Level: 0x04 (TCP)
- o Code: 0x03
- o Scheduler:
 - * 0x01: Lowest latency first
 - * 0x02: Redundant

8.1.5. Listen Backlog options

Kind	Length	Leg	Level	Backlog
1	2	2 bits	6 bits	2

Figure 15: Listen Backlog Option

- o Kind: 0x01 (Stack option)
- o Length: 5
- o Leg: 0x02 (Proxy-Remote leg)
- o Level: 0x04 (TCP)

- o Code: 0x04
- o Backlog: The length of the listen backlog. MUST be greater than 1.

The default behavior of the BIND does not allow a client to simultaneously handle multiple connections to the same bind address. A client can alter BIND's behavior by adding a TCP Listen Backlog Option to a BIND Request, provided that the Request is part of a Session.

In response, the proxy sends a TCP Listen Backlog Option as part of the Operation Reply, with the Backlog field signalling the actual backlog used. The proxy SHOULD NOT use a backlog longer than requested.

Following the successful negotiation of a backlog, the proxy listens for incoming connections for as long as the initial connection stays open. The initial connection is not used to relay data between the client and a remote host.

To accept connections, the client issues further BIND Requests using the bind address and port supplied by the proxy in the initial Operation Reply. Said BIND requests must belong to the same Session as the original Request.

If a proxy can not or will not honor a Listen Backlog option, it does so silently.

8.2. Authentication Method options

Authentication Method options are placed in SOCKS Requests to advertise supported authentication methods. In case of a CONNECT Request, they are also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.

Kind	Length	Initial Data Length	Methods
1	2	2	Variable

Figure 16: Authentication Method Option

- o Kind: 0x02 (Authentication Method option)

- o Length: The length of the option.
- o Initial Data Size: A two-byte number in network byte order. In case of CONNECT, this is the number of bytes of initial data that are supplied by the client immediately following the Request. This number MUST NOT be larger than 2^{14} .
- o Methods: One byte per advertised method. Method numbers are assigned by IANA.

Clients MUST support the "No authentication required" method. Clients SHOULD omit advertising the "No authentication required" option.

8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

Kind	Length	Method	Authentication Data
1	2	1	Variable

Figure 17: Authentication Data Option

- o Kind: 0x03 (Authentication Data option)
- o Length: The length of the option.
- o Method: The number of the authentication method. These numbers are assigned by IANA.
- o Authentication Data: The contents are specific to each method.

Clients SHOULD only place one Authentication Data option per authentication method. Server implementations MAY silently ignore all Authentication Data options for the same method aside from an arbitrarily chosen one.

8.4. Session options

Clients and proxies can establish SOCKS sessions, which span one or more Requests. All session-related negotiations are done via Session Options, which are placed in Requests and Authentication Replies by the client and, respectively, by the proxy.

Session Options have the following format:

Kind	Length	Type	Session Option Data
1	2	1	Variable

Figure 18: Session Option

- o Kind: 0x04 (Session option)
- o Length: The length of the option.
- o Type:
 - * 0x01: Session Request
 - * 0x02: Session ID
 - * 0x02: Session Teardown
 - * 0x04: Session OK
 - * 0x05: Session Invalid
 - * 0x06: Session Untrusted
- o Session Option Data: The contents are specific to each type.

Client and proxy implementations MUST either support all Session Option Types, or none.

8.4.1. Session initiation

A client can initiate a session by sending a Session Request Option:

Kind	Length	Type
1	2	1

Figure 19: Session Request Option

- o Kind: 0x04 (Session option)
- o Length: 4
- o Type: 0x01 (Session Request)

The proxy then replies with a Session ID Option in the successful Operation Reply:

Kind	Length	Type	Session ID
1	2	1	Variable

Figure 20: Session ID Option

- o Kind: 0x04 (Session option)
- o Length: The length of the option.
- o Type: 0x02 (Session ID)
- o Session ID: An opaque sequence of bytes specific to the session. MUST be at least one byte in size.

The Session ID serves to identify the session and is opaque to the client.

The credentials, or lack thereof, used to initiate the session are tied to the session. If authentication is to be performed for further Requests, the session is deemed "untrusted", and the proxy also places a Session Untrusted option in the Authentication Reply:

Kind	Length	Type
1	2	1

Figure 21: Session Untrusted Option

- o Kind: 0x04 (Session option)
- o Length: 5
- o Type: 0x06 (Session Untrusted)

The SOCKS Request that initiated the session is considered part of the session. A client **MUST NOT** attempt to initiate a session from within a different session.

If the proxy can not or will not honor the Session Request, it does so silently.

8.4.2. Further SOCKS Requests

Any further SOCKS Requests that are part of the session **MUST** include a Session ID Option (as seen in Figure 20).

The authentication procedure is altered based on the Session ID's validity and whether or not the Session is untrusted.

For valid Session IDs:

- o If the session is untrusted, the proxy **MUST** reject clients that do not authenticate using the same method and credentials that were used to initiate the session.
- o Otherwise, the proxy **MUST** ignore any authentication attempt in the Request, and **MUST NOT** require any authentication.

The proxy then replies by placing a Session OK option in the successful Authentication Reply:

Kind	Length	Type
1	2	1

Figure 22: Session OK Option

- o Kind: 0x04 (Session option)
- o Length: 5
- o Type: 0x04 (Session OK)

If the ticket is invalid, the first Authentication Reply MUST signal that authentication failed and can not continue (by setting the Type field to 0x01 and the Method field to 0xff). Further, it SHALL contain a Session Invalid option:

Kind	Length	Type
1	2	1

Figure 23: Session Invalid Option

- o Kind: 0x04 (Session option)
- o Length: 5
- o Type: 0x05 (Session Invalid)

8.4.3. Tearing down the session

Proxies can, at their discretion, tear down a session and free all associated state. Proxy implementations SHOULD feature a timeout mechanism that destroys sessions after a period of inactivity.

Clients can signal that a session is no longer needed, and can be torn down, by sending a Session Teardown option in addition to the Session ID option:

Kind	Length	Type
1	2	1

Figure 24: Session Teardown Option

- o Kind: 0x04 (Session option)
- o Length: 4
- o Type: 0x02 (Session Teardown)

After sending such an option, the client **MUST** assume that the session is no longer valid.

8.5. Idempotence options

To protect against duplicate SOCKS Requests, clients can request, and then spend, idempotence tokens. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if x and y are tokens, x is less than y if $0 < (y - x) < 2^{31}$ in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called token windows. Token windows are defined by their base (the first token in the range) and size. Windows can be shifted (i. e. have their base increased, while retaining their size) unilaterally by the proxy.

Requesting and spending tokens is done via Idempotence options:

Kind	Length	Type	Option Data
1	2	1	Variable

Figure 25: Idempotence Option

- o Kind: 0x05 (Idempotence option)
- o Length: The length of the option.

- o Type:
 - * 0x01: Token Request
 - * 0x02: Token Window Advertisement
 - * 0x03: Token Expenditure
 - * 0x04: Token Expenditure Reply
- o Option Data: The contents are specific to each type.

Idempotence options are only valid in the context of a SOCKS Session. If a SOCKS Request is not part of a Session (either by supplying a valid Session ID or successfully initiating one via a Session Request), the proxy MUST silently ignore any Idempotence options.

Token windows are tracked by the proxy on a per-session basis. There can be at most one token window for every session and its tokens can only be spent from within said session.

Client and proxy implementations MUST either support all Idempotence Option Types, or none.

8.5.1. Requesting a fresh token window

A client can obtain a fresh window of tokens by sending a Token Request option as part of a SOCKS Request:

Kind	Length	Type	Window Size
1	2	1	4

Figure 26: Token Request

- o Kind: MUST be allocated by IANA. (See Section 13.)
- o Length: 7
- o Type: 0x00 (Token Request)
- o Window Size: The requested window size.

If a token window is issued, the proxy then includes a Token Window Advertisement option in the corresponding successful Authentication Reply:

Kind	Length	Type	Window Base	Window Size
1	2	1	4	4

Figure 27: Token Window Advertisement

- o Kind: 0x05 (Idempotence option)
- o Length: 11
- o Type: 0x01 (Token Grant)
- o Window Base: The first token in the window.
- o Window Size: The window size. This value SHOULD be lower or equal to the requested window size. Window sizes MUST be less than 2^{31} . Window sizes MUST NOT be 0.

If no token window is issued, the proxy MUST silently ignore the Token Request.

8.5.2. Spending a token

The client can attempt to spend a token by including a Token Expenditure option in its SOCKS request:

Kind	Length	Type	Token
1	2	1	4

Figure 28: Token Expenditure

- o Kind: 0x05 (Idempotence option)
- o Length: 7
- o Type: 0x02 (Token Expenditure)

- o Token: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending a Token Expenditure Reply option as part of the successful Authentication Reply:

Kind	Length	Type	Response Code
1	2	1	1

Figure 29: Token Expenditure Response

- o Kind: 0x05 (Idempotence option)
- o Length: 4
- o Type: 0x03 (Token Expenditure Response)
- o Response Code:
 - * 0x01: Success: The token was spent successfully.
 - * 0x02: No Window: The proxy does not have a token window associated with the client.
 - * 0x03: Out of Window: The token is not within the window.
 - * 0x04: Duplicate: The token has already been spent.

If eligible, the token is spent before attempting to honor the Request. If the token is not eligible for spending, the proxy MUST NOT attempt to honor the client's Request; further, it MUST indicate a General SOCKS server failure in the Operation Reply.

Proxy implementations SHOULD also send a Token Window Advertisement if:

- o the token is out of window, or
- o by the proxy's internal logic, successfully spending the token caused the window to shift.

Proxy implementations SHOULD NOT shift the window's base beyond the highest unspent token.

Proxy implementations MAY include a Token Window Advertisement in any Authentication Reply that indicates success.

8.5.3. Handling Token Window Advertisements

Even though the proxy increases the window's base monotonically, there is no mechanism whereby a SOCKS client can receive the Token Window Advertisements in order. As such, clients SHOULD disregard unsolicited Token Window Advertisements with a Window Base less than the previously known value.

9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929].

Clients can also attempt to authenticate by placing the Username/Password request in an Authentication Data Option.

Kind	Length	Method	Username/Password request
1	2	1	Variable

Figure 30: Password authentication via a SOCKS Option

- o Kind: 0x03 (Authentication Data option)
- o Length: The length of the option.
- o Method: 0x02 (Username/Password).
- o Username/Password request: The Username/Password request, as described in [RFC1929].

10. TCP Fast Open on the Client-Proxy Leg

TFO breaks TCP semantics, causing replays of the data in the SYN's payload under certain rare circumstances [RFC7413]. A replayed SOCKS Request could itself result in a replayed connection on behalf of the client.

As such, client implementations SHOULD NOT use TFO on the client-proxy leg unless:

- o The protocol running on top of SOCKS tolerates the risks of TFO, or

- o The SYN's payload does not contain any application data (so that no data is replayed to the server, even though duplicate connections are still possible), or
- o The client uses Idempotence Options, making replays very unlikely, or
- o SOCKS is running on top of TLS and Early Data is not used.

11. False Starts

In case of CONNECT Requests, the client MAY start sending application data as soon as possible, as long as doing so does not incur the risk of breaking the SOCKS protocol.

Clients must work around the authentication phase by doing any of the following:

- o If the Request does not contain an Authentication Method option, the authentication phase is guaranteed not to happen. In this case, application data MAY be sent immediately after the Request.
- o Application data MAY be sent immediately after receiving an Authentication Reply indicating success.
- o When performing a method-specific authentication sequence, application data MAY be sent immediately after the last client message.

12. Security Considerations

12.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 16 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- o the request is not fully received after a certain timeout, or
- o the number of options or their size exceeds an imposed hard cap.

12.2. Replay attacks

In TLS 1.3, early data (which is likely to contain a full SOCKS request) is prone to replay attacks.

While Token Expenditure options can be used to mitigate replay attacks, the initial Token Request is still vulnerable. As such, client implementations SHOULD NOT make use of TLS early data unless the Request attempts to spend a token.

12.3. Resource exhaustion

Malicious clients can issue a large number of Session Requests, forcing the proxy to keep large amounts of state.

To mitigate this, the proxy MAY implement policies restricting the number of concurrent sessions on a per-IP or per-user basis, or barring unauthenticated clients from establishing sessions.

13. IANA Considerations

This document requests that IANA allocate 1-byte option kinds for SOCKS 6 options. Further, this document requests the following option kinds:

- o Stack options: 0x01
- o Authentication Method options: 0x02
- o Authentication Data options: 0x03
- o Session options: 0x04
- o Idempotence options: 0x05
- o A range for vendor-specific options: 0xE0-0xFF

This document also requests that IANA allocate a TCP and UDP port for SOCKS over TLS and DTLS, respectively.

14. Acknowledgements

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

15. References

15.1. Normative References

- [RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, DOI 10.17487/RFC1929, March 1996, <<https://www.rfc-editor.org/info/rfc1929>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

15.2. Informative References

- [I-D.ietf-tls-dtls-connection-id] Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-04 (work in progress), March 2019.
- [RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <<https://www.rfc-editor.org/info/rfc1928>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<https://www.rfc-editor.org/info/rfc6824>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Authors' Addresses

Vladimir Olteanu
University Politehnica of Bucharest

Email: vladimir.olteanu@cs.pub.ro

Dragos Niculescu
University Politehnica of Bucharest

Email: dragos.niculescu@cs.pub.ro

Network Working Group
Internet-Draft
Updates: 2863 (if approved)
Intended status: Standards Track
Expires: September 6, 2019

D. Thaler
Microsoft
D. Romascanu
Independent
March 05, 2019

Guidelines and Registration Procedures for Interface Types
draft-thaler-ifttype-reg-01

Abstract

The registration and use of interface types ("ifType" values) predated the use of IANA Considerations sections and YANG modules, and so confusion has arisen about the interface type allocation process. This document provides updated guidelines for the definition of new interface types, for consideration by those who are defining, registering, or evaluating those definitions.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on September 6, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	2
3. Problems	3
4. Interface Sub-Layers and Sub-Types	3
5. Registration	4
5.1. Procedures	5
5.2. Media-specific OID-subtree assignments	6
5.3. Registration Template	6
6. IANA Considerations	7
7. Security Considerations	8
8. References	8
8.1. Normative References	8
8.2. Informative References	9
Authors' Addresses	9

1. Introduction

The IANA IfType-MIB was originally defined in [RFC1573] as a separate MIB module together with the Interfaces Group MIB (IF-MIB) module. The IF-MIB module has since been updated and is currently specified in [RFC2863], but this latest IF-MIB RFC no longer contains the IANA IfType-MIB. Instead, the IANA IfType-MIB is now maintained as a separate module. Similarly, [RFC7224] created an initial INANA Interface Type YANG Module, and the current version is maintained by IANA.

The current IANA IfType registries are in [iana-if-type], [IANAifType-MIB], and [ifType].

Although the ifType registry was originally defined in a MIB module, the assignment and use of interface type values are not tied to MIB modules or any other management mechanism. Interface type values can be used as values of data model objects (MIB objects, YANG objects, etc.), as parts of a unique identifier of a data model for a given interface type (e.g., in an OID), or simply as values exposed by local APIs or UI on a device.

2. Terminology

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

14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Problems

This document addresses the following issues:

1. As noted in Section 1, the former guidance was written with wording specific to MIB modules, and accordingly some confusion has resulted when using YANG modules. This document clarifies that ifTypes are independent from the type of, or even existence of, a data model.
2. The use of, and requirements around, sub-layers and sub-types are not well understood even though good examples of both exist. This is discussed in Section 4.
3. The registry is kept in the format of MIB and YANG modules, but there was no process guidance written to check that updates were syntactically correct, which led to the registry having syntax errors that broke tools. Section 5.1 adds a validation step to the documented assignment procedure.
4. Transmission values [ifType] have often been allocated as part of ifType allocation, but no guidance exists about whether a requester must ask for it or not, and the request form has no such required field. As a result, IANA has asked the Designated Expert to decide for each allocation, but no relevant guidance for the Designated Expert has been documented. This is discussed in Section 5.2.
5. Various documents and registries say to submit requests via email, but a web form exists for submitting requests, which has caused some confusion around which is to be used. This is discussed in Section 6.

4. Interface Sub-Layers and Sub-Types

When multiple sub-layers exist below the network layer, each sub-layer can be represented by its own row in the ifTable with its own ifType, with the ifStackTable being used to identify the upward and downward multiplexing relationships between rows. Section 3.1.1 of [RFC2863] provides more discussion, and Section 3.1.2 of that RFC provides guidance for defining interface sub-layers. More recent experience shows that these guidelines are phrased in a way that is now too restrictive, since at the time [RFC2863] was written, MIB modules were the dominant data model.

This document clarifies that such guidance also applies to YANG modules.

Some ifTypes may define sub-types. For example, the tunnel(131) ifType defines sub-types, where each IANA tunnelType can have its own MIB and/or YANG module with protocol-specific information, but there is enough in common that some information is exposed in a generic IP Tunnel MIB corresponding to the tunnel(131) ifType.

For requests that involve multiple sub-layers below the network layer, requests MUST include (or reference) a discussion of the multiplexing relationships between sub-layers, ideally with a diagram. Various well-written examples exist of such definitions, including [RFC3637] section 3.4.1, [RFC4087] section 3.1.1, and [RFC5066] section 3.1.1.

Definers of sub-layers and sub-types should consider which model is more appropriate for their needs. A sub-layer is generally used whenever either a dynamic relationship exists (i.e., which instances layer over which other instances can change over time) or a multiplexing relationship exists with another sub-layer. A sub-type can be used when neither of these are true, but where one interface type is conceptually a subclass of another interface type, as far as a management data model is concerned.

PROPOSED CLARIFICATION/ELABORATION: The intent of an interface type or sub-type is that its definition should be sufficient to identify an interoperable protocol. In some cases, a protocol might be defined in a way that is not sufficient to provide interoperability with other compliant implementations of that protocol. In such a case, an ifType definition should discuss whether specific instantiations (or profiles) of behavior should use a sub-layer model (e.g., each vendor might layer the protocol over its own sub-layer that provides the missing details), or a sub-type model (i.e., each vendor might subclass the protocol without any layering relationship). If a sub-type model is more appropriate, then the data model for the protocol might include a sub-type identifier so that management software can discover objects specific to the subtype. In either case, such discussion is important to guide definers of a data model for the more specific information (i.e., a lower sub-layer or a subtype), as well as the Designated Expert that must review requests for any such ifTypes or sub-types.

5. Registration

The IANA policy (using terms defined in [RFC5226]) for registration is Expert Review. The role of the Designated Expert in the procedure is to raise possible concerns about wider implications of proposals

for use and deployment of interface types. While it is recommended that the responsible Area Director and the IESG take into consideration the Designated Expert opinions, nothing in the procedure empowers the Designated Expert to override properly arrived-at IETF or working group consensus.

5.1. Procedures

Someone wishing to register a new ifType value MUST:

1. Check the IANA registry to see whether there is already an entry that could easily satisfy the modeling and functional requirements for the requested entry. If there is already such an entry, use it or update the existing specification. Text in an Internet-Draft, or part of some other permanently available, stable specification may be written to clarify the usage of an existing entry or entries for the desired purpose.
2. Check the IANA registry to see whether there is already some other entry with the desired name. If there is already an unrelated entry under the name, choose a different name.
3. Prepare a registration request using the template specified in Section 5.3. The registration request can be contained in an Internet-Draft, submitted alone, or as part of some other permanently available, stable, specification. The registration request can also be submitted in some other form (as part of another document or as a stand-alone document), but the registration request will be treated as an "IETF Contribution" under the guidelines of [RFC5378].
4. Submit the registration request (or pointer to document containing it) to IANA at iana@iana.org or via the web form at <https://www.iana.org/form/iftypes>.

Upon receipt of a registration request, the following steps MUST be followed:

1. IANA checks the submission for completeness; if required information is missing or any citations are not correct, IANA will reject the registration request. A registrant can resubmit a corrected request if desired.
2. IANA requests Expert Review of the registration request against the corresponding guidelines from this document.
3. The Designated Expert will evaluate the request against the criteria.

4. Once the Designated Expert approves registration, IANA updates [ifType], [IANAifType-MIB], and [iana-if-type] to show the registration. When adding values to the IANAifType-MIB, IANA should verify that the updated MIB module is syntactically correct before publishing the update. There are various existing tools or web sites that can be used to do this verification.
5. If instead the Designated Expert does not approve registration (e.g., for any of the reasons in [RFC5226] section 3), a registrant can resubmit a corrected request if desired, or the IESG can override the Designated Expert and approve it per the process in Section 5.3 of [RFC5226].

5.2. Media-specific OID-subtree assignments

The current IANAifType-MIB notes:

The relationship between the assignment of ifType values and of OIDs to particular media-specific MIBs is solely the purview of IANA and is subject to change without notice. Quite often, a media-specific MIB's OID-subtree assignment within MIB-II's 'transmission' subtree will be the same as its ifType value. However, in some circumstances this will not be the case, and implementors must not pre-assume any specific relationship between ifType values and transmission subtree OIDs.

The following change is proposed:

CURRENT: For every ifType registration, the corresponding transmission number value should be registered or marked "Reserved."

PROPOSED: For future ifType assignments, an OID-subtree assignment MIB-II's 'transmission' subtree will be made with the same value.

RATIONALE: (1) This saves effort in the future since if a transmission number is later needed, no IANA request is needed that would then require another Expert Review. (2) The transmission numbering space is not scarce, so there seems little need to reserve the number for a different purpose than what the ifType is for. (3) The Designated Expert need not review whether a transmission number value should be registered when processing each ifType request, thus reducing the possibility of delaying assignment of ifType values.

5.3. Registration Template

This template describes the fields that MUST be supplied in a registration request suitable for adding to the registry:

Label for IANA ifType requested: As explained in Section 7.1.1 of [RFC2578], a label for a named-number enumeration must consist of one or more letters or digits, up to a maximum of 64 characters, and the initial character must be a lower-case letter. (However, labels longer than 32 characters are not recommended.) Note that hyphens are not allowed.

Name of IANA ifType requested: A short description (e.g., a protocol name), suitable to appear in a comment in the registry.

Description of the proposed use of the IANA ifType: Requesters MUST include answers, either directly or via a link to some document with the answers, to the following questions in the explanation of the proposed use of the IANA IfType:

- o How would IP run over your ifType?
- o Would there be another interface sublayer between your ifType and IP?
- o Would your ifType be vendor-specific or proprietary? (If so, the label MUST start with a string that shows that. For example, if your company's name or acronym is xxx, then the ifType label would be something like xxxSomeIfTypeLabel.)
- o (ADDED) Would your ifType require or allow vendor-specific extensions? If so, would the vendor use their own ifType in sub-layer below the requested ifType, or a sub-type of the ifType, or some other mechanism?

Reference, Internet-Draft, or Specification: A link to some document is required.

Additional information or comments: Optionally any additional comments for IANA or the Designated Expert.

6. IANA Considerations

This entire document is about IANA considerations.

CURRENT: The registries say to use email, but a web form exists (<https://www.iana.org/form/iftypes>), which is an apparent contradiction. Should IANA require using the form? Or require using email? Or accept submissions either way?

PROPOSED: In addition, IANA is requested to make the following changes:

1. [IANAifType-MIB] currently says: "Requests for new values should be made to IANA via email (iana@iana.org)." This should be updated to instead say: "Requests for new values should be made at <https://www.iana.org/form/iftypes> or by email (iana@iana.org)."
2. [iana-if-type] currently says: "Requests for new values should be made to IANA via email (iana@iana.org)." This should be updated to instead say: "Requests for new values should be made at <https://www.iana.org/form/iftypes> or by email (iana@iana.org)."

7. Security Considerations

Since this document does not introduce any technology or protocol, there are no security issues to be considered for this document itself.

8. References

8.1. 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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<https://www.rfc-editor.org/info/rfc2578>>.
- [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000, <<https://www.rfc-editor.org/info/rfc2863>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <<https://www.rfc-editor.org/info/rfc5226>>.
- [RFC5378] Bradner, S., Ed. and J. Contreras, Ed., "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, DOI 10.17487/RFC5378, November 2008, <<https://www.rfc-editor.org/info/rfc5378>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

8.2. Informative References

- [iana-if-type]
IANA, "iana-if-type YANG Module", February 2019,
<<http://www.iana.org/assignments/iana-if-type>>.
- [IANAifType-MIB]
IANA, "IANAifType-MIB", February 2019,
<<http://www.iana.org/assignments/ianaiftype-mib>>.
- [ifType] IANA, "ifType definitions", February 2019,
<<https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-5>>.
- [RFC1573] McCloghrie, K. and F. Kastenholz, "Evolution of the
Interfaces Group of MIB-II", RFC 1573,
DOI 10.17487/RFC1573, January 1994, <<https://www.rfc-editor.org/info/rfc1573>>.
- [RFC3637] Heard, C., Ed., "Definitions of Managed Objects for the
Ethernet WAN Interface Sublayer", RFC 3637,
DOI 10.17487/RFC3637, September 2003, <<https://www.rfc-editor.org/info/rfc3637>>.
- [RFC4087] Thaler, D., "IP Tunnel MIB", RFC 4087,
DOI 10.17487/RFC4087, June 2005, <<https://www.rfc-editor.org/info/rfc4087>>.
- [RFC5066] Beili, E., "Ethernet in the First Mile Copper (EFMCu)
Interfaces MIB", RFC 5066, DOI 10.17487/RFC5066, November
2007, <<https://www.rfc-editor.org/info/rfc5066>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module",
RFC 7224, DOI 10.17487/RFC7224, May 2014,
<<https://www.rfc-editor.org/info/rfc7224>>.

Authors' Addresses

David Thaler
Microsoft

EMail: dthaler@microsoft.com

Dan Romascanu
Independent

EMail: dromasca@gmail.com