SIG on EIP S. Salsano
Internet-Draft Univ. of Rome Tor Vergata / CNIT
Intended status: Informational A. Mayer
Expires: 20 September 2026 Univ. of Rome Tor Vergata / CNIT / Common Net
19 March 2026

# Extensible In-band Processing (EIP) Headers Definitions draft-eip-headers-definitions-00

## Abstract

This document discusses the format of EIP protocol elements and the transport of EIP in different IPv6-based protocol headers.

In particular, this document defines the standalone EIP formats for the IPv6 Hop-by-Hop Options Header and for the Segment Routing Header, and it defines the generic format of EIP Information Elements. Other approaches to transport EIP, including integration with the IOAM framework through the Global Opaque Block (GOB), are described in separate documents.

Caveat: this document is still in brainstorming stage, it is distributed to stimulate discussion.

## About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://eip-home.github.io/eip-headers/draft-eip-headers-definitions.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-eip-headers-definitions/.

Discussion of this document takes place on the EIP SIG mailing list (mailto:eip@cnit.it), which is archived at http://postino.cnit.it/cgi-bin/mailman/private/eip/.

Source for this draft and an issue tracker can be found at https://github.com/eip-home/eip-headers.

## Status of This Memo

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

## Copyright Notice



## Table of Contents

## 1. Introduction

The EIP header is used to carry information that IPv6 nodes (hosts and routers) can read and write to support different use cases. The use cases for EIP are discussed in [id-eip-use-cases]. EIP provides a common architecture and framework which can be extended/tailored for the different use cases. The EIP architecture is discussed in [I-D.draft-eip-arch].

The present document focuses on the standalone EIP formats and on the generic definition of EIP Information Elements, while other documents describe additional approaches to transport EIP.

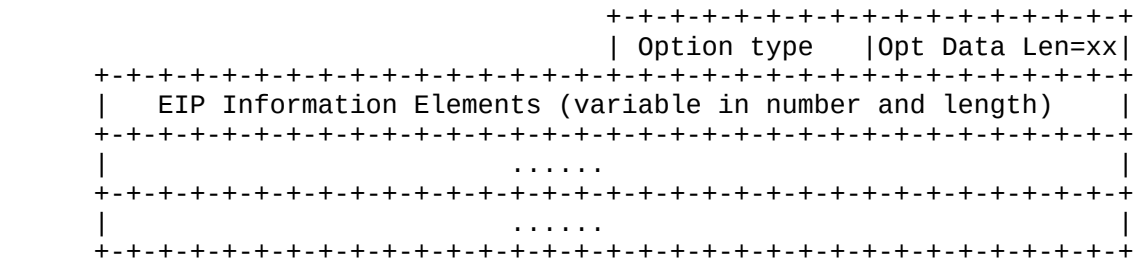
The design of the EIP header takes into account the requirement to be efficient and "hardware friendly" (i.e. the effort and cost to implement EIP in hardware achieving line rate forwarding needs to be reasonable).

The benefits of having EIP as a common header and framework to support multiple use cases are discussed in [I-D.draft-eip-arch].

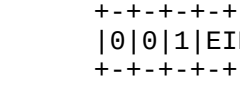
EIP can be transported in different ways within IPv6. This document defines two standalone approaches: 1) an EIP Option for the Hop-by-Hop Options Header; and 2) an EIP TLV for the Segment Routing Header. Additional approaches to transport EIP are also possible. In particular, EIP Information Elements can be carried within the IOAM framework through the Global Opaque Block (GOB) of the IOAM Pre-allocated Trace Option, as described in [I-D.draft-mayer-ioam-gob].

## 2. Standalone EIP Transport in the Hop-by-Hop Options Header

The EIP header can be carried as an Option in the Hop by Hop Extension Header, as shown in the following figure.




Option type




First 3 bits in Option type field:

* 0 0 Skip if not implemented

* 1 Content might change at every hop

(eventually) EIP code needs to be allocated by IANA for the time being we use 11110, so overall the Option Type for EIP is

0x3E (001 11110) RFC3692-style Experiment

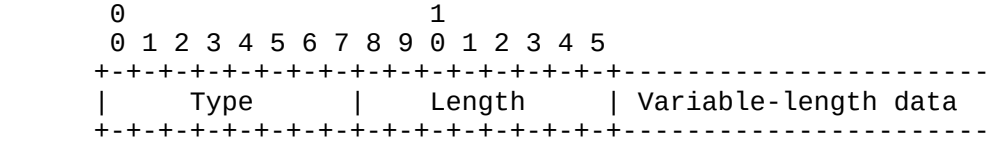
Opt Data Len is the length in bytes of the rest of the EIP Option

Within the EIP Option, we have a LTV structure:

EIP-LTV a code that is specific of each EIP LTV
LTV Len is the length in bytes of the rest of the LTV

## 3. Standalone EIP Transport in the Segment Routing Header

The EIP header can be carried as a TLV in the Segment Routing Header. A generic TLV in the SRH is defined as follows.




First bit in type field: 1 - Content might change at every hop

type code needs (eventually) to be allocated by IANA for the time being we use 252 for the EIP TLV. This is part of the experimental range

252-254 Experimentation and Test [RFC8754]

NB current IANA allocation for Types starting with 1 is (see https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#segment-routing-header-tlvs)

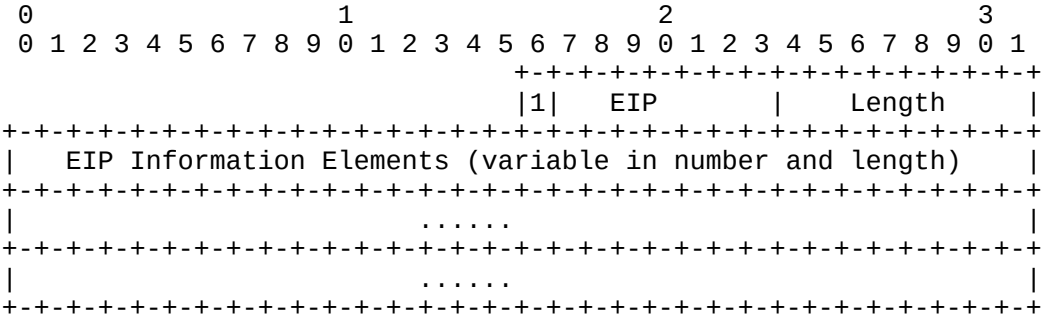
127 possible Option Types starting with 1
123 not allocated
3 allocated for Experimentation and Test
1 Reserved

128-251 Unassigned

252-254 Experimentation and Test [RFC8754]

255 Reserved [RFC8754]

The EIP TLV for SRH will carry a set of EIP Information Elements as shown hereafter.




## 4. Other Approaches to Transport EIP

The standalone formats defined in this document are not the only possible ways to transport EIP Information Elements.

As discussed in [I-D.draft-eip-arch], EIP is an architectural framework for structured in-band metadata and processing, and it can be mapped onto different IPv6-based protocol containers.

In addition to the standalone EIP Option for the Hop-by-Hop Options Header and the standalone EIP TLV for the Segment Routing Header, another important approach is to carry EIP Information Elements

within the IOAM framework through the Global Opaque Block (GOB) of the IOAM Pre-allocated Trace Option, as described in [I-D.draft-mayer-ioam-gob].

For this reason, this document should be read as defining the generic EIP Information Elements and two standalone transport formats, rather than as restricting EIP to a single protocol representation.

## 5. Generic Format of EIP Information Elements

EIP Information Elements are used to carry the information needed by the different use cases. The same Information Element can be reused across multiple use cases. The same EIP Information Elements can also be used across different approaches to transport EIP, including the standalone formats defined in this document and other mappings such as the IOAM/GOB-based approach.

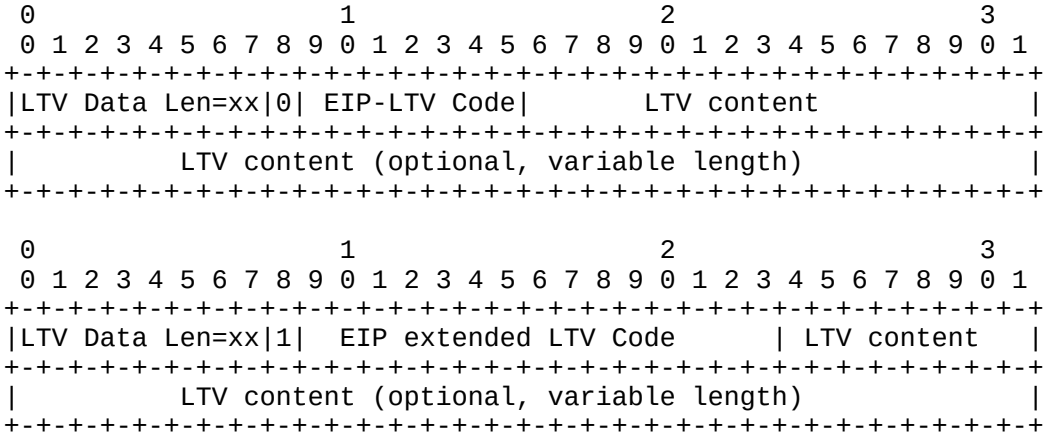
A fundamental requirement for EIP is to be "Extensible", therefore we need to have a potentially large number of different Information Elements. On the other hand, we may need to be efficient, limiting the overhead in bytes for carrying a given information. In order to have the possibility to find the optimal trade-off between these contrasting requirements, the Codes or "Tags" for the Information Elements can have different sizes. In particular, we select a solution in which the Codes can have three different lengths (respectively one byte, two bytes or three bytes).

In order to support the variability in the size of the Code of the Information Element, we use an LTV (Length-Tag-Value) approach instead of TLVs (Tag-Length-Value).

We have considered two approaches for the structure of the EIP Information Elements or EIP LTVs.

### 5.1. Proposed approach #1 for EIP LTVs.

In this approach, we have one byte LTV Data Length field. The LTV Data Len is in octets (up to 255 bytes for the LTV Data Length).

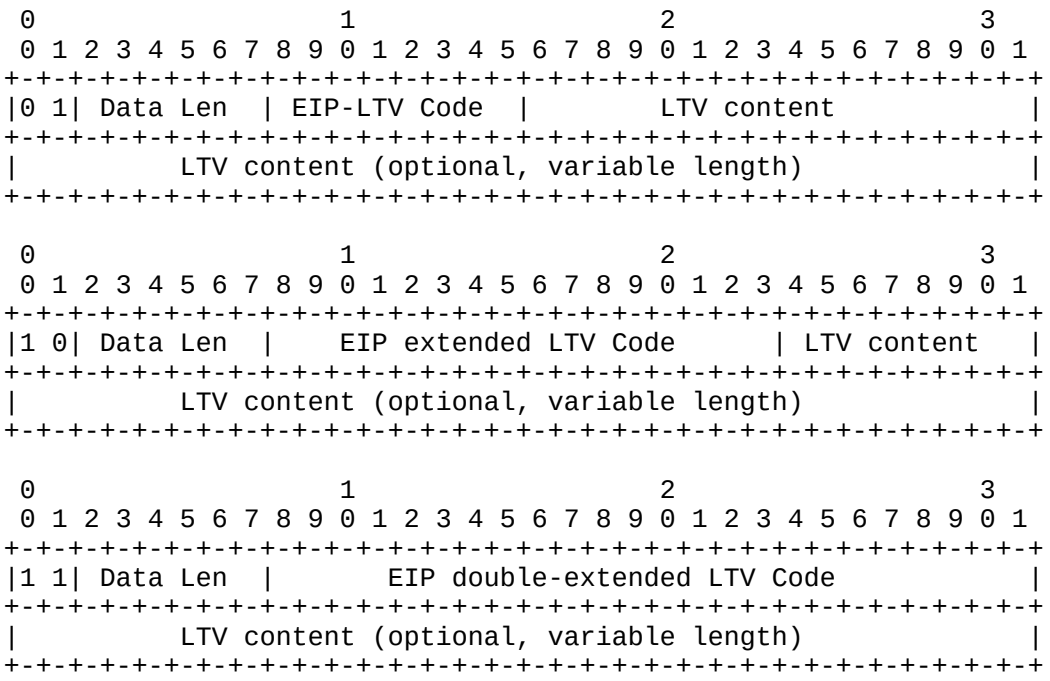



In this approach, we can have 128 codes of one byte and 32768 codes of 2 bytes.

## 5.2. Proposed approach #2 for EIP LTVs.

In this second approach, the first two bits of the Len field can differentiate the code length, and the remaining 6 bits will specify the length of the optional part of the LTV content in 32 bits units (4-octets units). The LTV Data length can be up to 63x4=252 bytes (covering the optional part).

Note that when the Data Len is 0, the optional part of the LTV content is not present, and the Information Element is made only of 4 bytes (i.e. one row in the figures below).




In this approach, we can have 256 codes of one byte, 65536 codes of 2 bytes and 2^24 codes of 3 bytes. The codes of one byte will be referred to as "Short codes", the codes of two bytes will be referred to as "Extended codes", the codes of three bytes will be referred to as "Double extended codes".

### 5.3. Decision on the approach for EIP LTVs

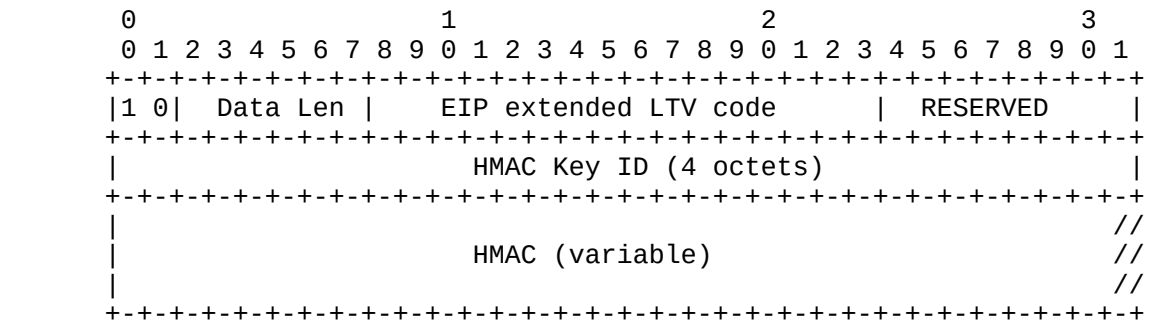
The selected approach is the #2, because it is more flexible and it supports a much higher number of Information Elements Codes.

## 6. Definition of EIP Information Elements (a.k.a. EIP LTVs)

### 6.1. HMAC LTV

Alignment requirement: 8n (TODO: maybe it will be 4n)

The keyed Hashed Message Authentication Code (HMAC) LTV is OPTIONAL and has the following format:




EIP extended LTV code: HMAC (see Table 3 in Section 7.2)

Data Len: the length of HMAC LTV in 4-bytes units, excluding the first row: the counting starts from the second row (HMAC Key ID).

RESERVED: 8 bits. MUST be 0 on transmission.

HMAC Key ID: A 4-octet opaque number that uniquely identifies the pre-shared key and algorithm used to generate the HMAC.

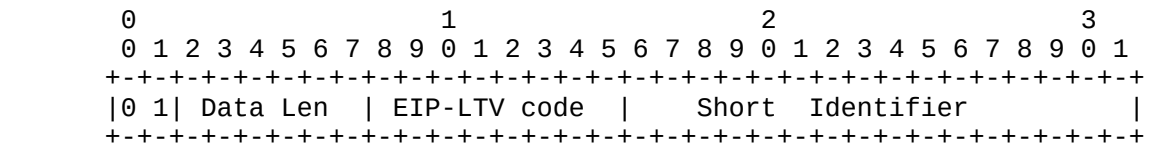
HMAC: Keyed HMAC, in multiples of 8 octets, at most 32 octets.

The details of the use of the HMAC LTV (HMAC Generation and Verification and HMAC Algorithms) are borrowed from section 2.1.2 of RFC 8754 [RFC8754].

## 6.2. EIP Identifiers LTVs

EIP identifiers can be used for different use-cases. For example, they can be used to identify a "slice", or a Customer, or they can be used to carry a "Contract Identifier". Two classes of EIP Identifies are defined, Short and Long Identifiers. Short Identifiers are 16 bits Identifiers. Long Identifiers are Nx32 bits long (N>=1). Long Identifiers can be further structured according to the specific use case. EIP Long Identifiers can also be used to carry sequence numbers or transaction identifiers to identify a specific packet or a specific transaction.

### 6.2.1. EIP Short Identifier LTV

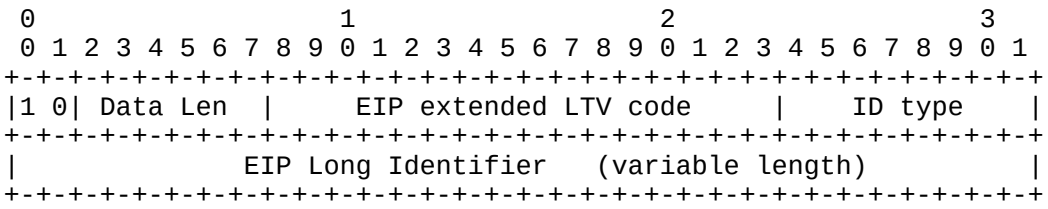


EIP-LTV code: Short Identifier (see Table 2 in Section 7.1)

Data Len: The length of the variable-length data in 4-bytes units is zero in this case.

Short Identifier: it is a 16 bits identifier, useful when up to 65536 different Identifiers are needed.

### 6.2.2. EIP Long Identifier LTV




EIP extended LTV code: Long Identifier (see Table 3 in Section 7.2)

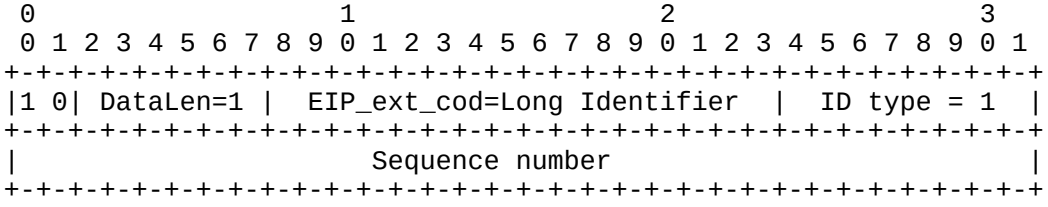
Data Len: The length of the variable-length part of the data in 4-bytes units.

ID type: maybe used to qualify different types of identifiers. By default it is zero. An ID type can be used to specify a structure for the variable length part of the Long Identifier.

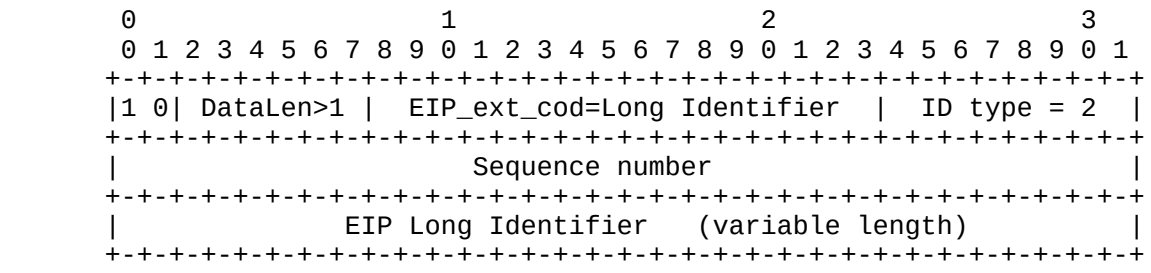
EIP Long Identifier: an identifier of variable length (in multiple of 4 bytes)

ID type = 0 : Generic Long Identifier
ID type = 1 : Sequence Number only
ID type = 2 : Sequence Number and Generic Long Identifier

The EIP Long Identifier LTV only carrying a Sequence Number is shown hereafter.

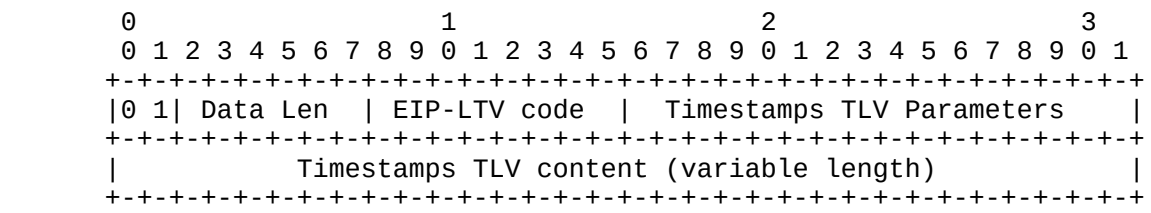



The EIP Long Identifier LTV carrying a Sequence Number in addition to a generic Long identifier is shown hereafter.



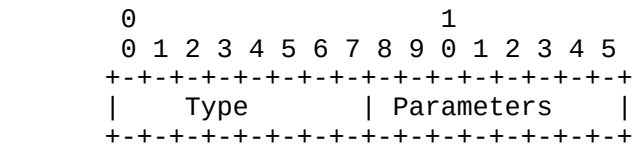

## 6.3. Timestamps LTV:

The Timestamp Information Element can used to collect timing information in different nodes along the packet path (source, destination, intermediate nodes), according to different use cases.



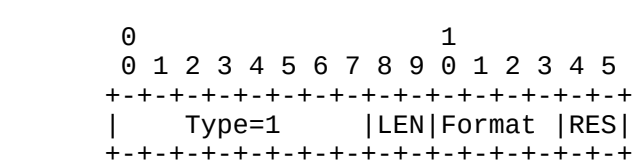

EIP-LTV code: Timestamps (see Table 2 in Section 7.1)

The "Timestamps TLV Parameters" is a 16 bits field, it is split into two 8 bits fields as follows:




Type : it further characterizes the format and the content of Timestamps

Timestamp Type: Basic = 1




LEN: the length of each timestamp

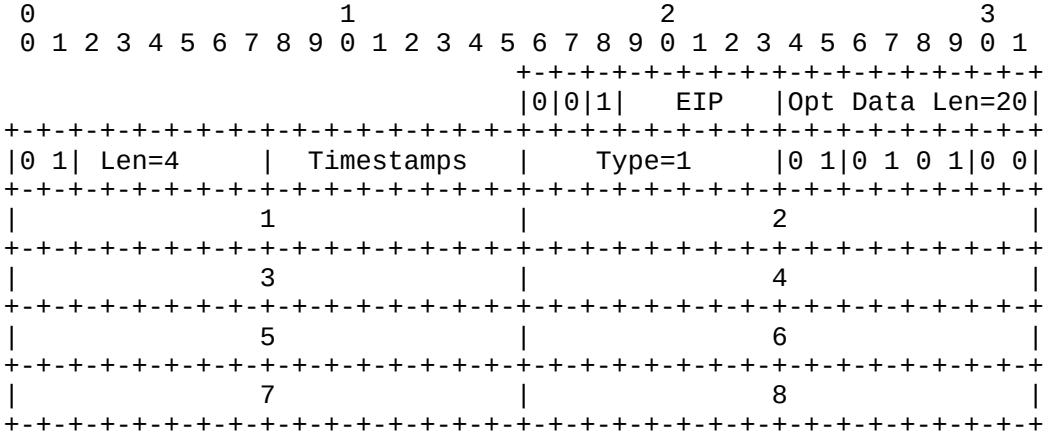
00 : 1 byte
01 : 2 bytes
10 : 4 bytes
11 : 8 bytes

Format: indicates the format of the timestamp

0001 : 1 ns
0010 : 10 ns
0011 : 100 ns
0100 : 1 us
0101 : 10 us
0110 : 100 us
0111 : 1 ms
1000 : NTP (only for 8 bytes)
1001 : Linux epoch (only for 8 bytes)

RES: Reserved, set to 0 0

Example of a Timestamp TLV of type basic, that carries 8 timestamps of length 2 bytes, each one representing a time granularity of 10 us.




NB using this granularity (10 us) and timestamp size (2 bytes), assuming that all node clocks are synchronized with a maximum error E [ms] it is possible to correctly evaluate hop-by-hop delays up to (655-E) [ms]

The entire timestamp will be reconstructed at last node using system time and subtracting intermediate timestamps.

## 6.4. Node selection and identification LTV

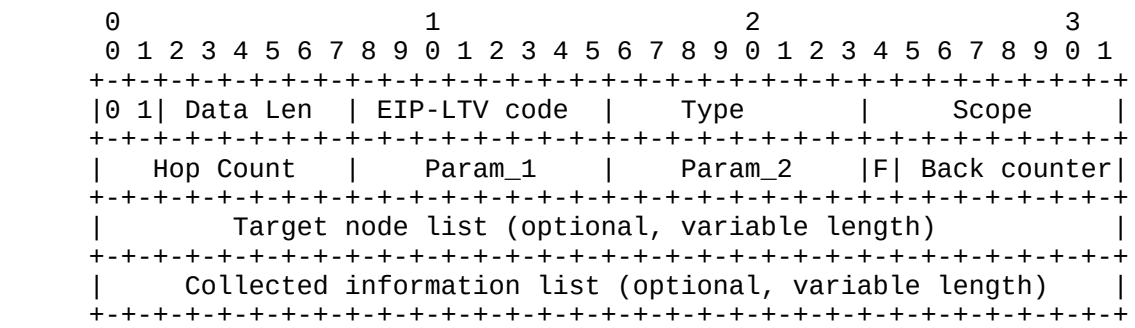
This Information Element can be used for two purposes:

1) associate some instruction contained in the EIP header to one or more target nodes in the "downstream" path of the packet

example 1.1: ask a specific node to write some information in the EIP header, if this specific node will be crossed by the packet

example 1.2: ask one node every N hops to write some information in the EIP header

2) identify the nodes that have inserted some information in the EIP header



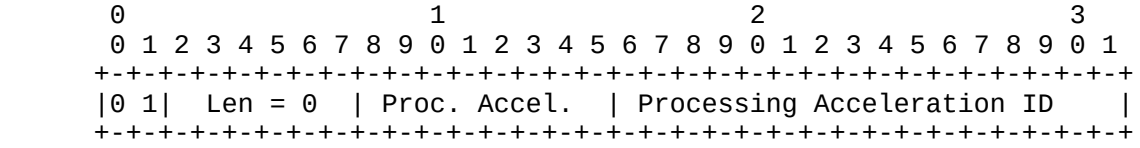

Type = All nodes (Skip M), Param_1 : M, Param_2 : limit (num. of nodes)
Type = Modulus, Param_1 : modulus, Param_2 : limit
Type = Destination only
Type = Segment End only, Param_2 is the limit
Type = Segment End Modulus, Param_1 : modulus, Param_2 : limit
Type = Target Node(s), Param_2 is the number of nodes

Scope : If Scope = 0x00 the node selection applies to all the following EIP Information Elements (LTVs) If Scope = N > 0x00 only to the first N LTVs following the Scope are affected by the selection.

## 6.5. Processing Accelerator LTV

The purpose of this LTV is to identify a portion of the EIP Header that has a fixed pre-known format so that it can be processed in a more efficient way, rather than proceeding with a "sequential" parsing. This is can be used for parallel hardware processing, but also software processing can be optimized. For example, a pre-known sequence of LTVs following the Processing Accelerator LTV can be processed in parallel. The Processing Accelerator LTV can be used to

know in advance the size and type of some fields of the following LTVs in order to speed up the processing.

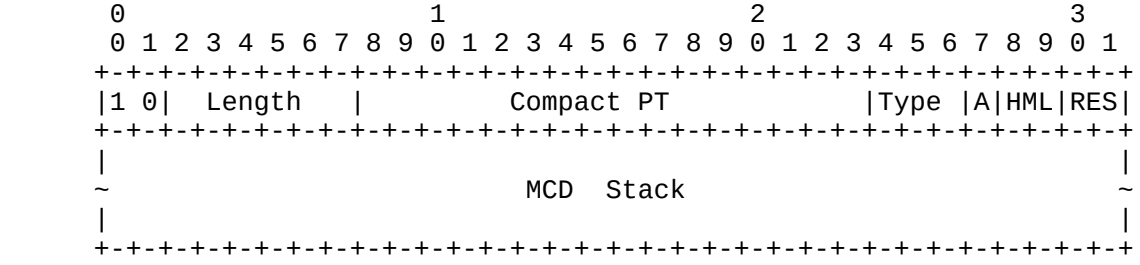



Proc. Accel. : EIP Short code (see Table 2 in Section 7.1)

The Processing Acceleration ID is an opaque identifier, its definition is domain-specific.

## 6.6. Compact Path Tracing (CPT) LTV

This LTV is a porting of the Internet Draft "draft-filsfils-spring-path-tracing" [I-D.filsfils-spring-path-tracing] into the EIP framework. It contains a stack of "MCD" (Midpoint Compressed Data) representing information inserted by every transit router for path tracing purposes.




Compact PT: EIP extended code (see Table 3 in Section 7.2)

Type: 3 bits, specifies the content of the CPT LTV including the format of the MCD element.

A: Authenticated, set to 1 in Authenticated Mode, 0 otherwise.

HML: HMAC Length, set to 00 in unauthenticated mode. In authenticated mode, it stores the length of the HMAC field in 8 octets.\ MAC length = (HML + 1) * 8 bytes.\ Max length of the HMAC will be 32 octets.

RES: Reserved, set to 00.

Ultra Compact (Type = 000)
MCD 24 Bits (3 bytes)
Timestamp (8 bit) | Interface ID (12 bit) | Load (4 bit)

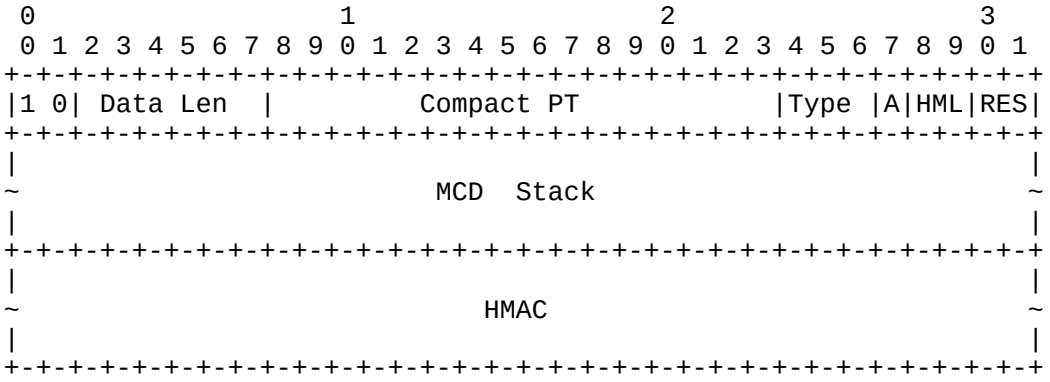
Compact (Type = 001)
MCD 32 Bits (4 bytes)
Timestamp (10 bit) | Interface ID (16 bit) | Load (4 bit) | Timeshift (2 bit)

The MCD Stack has variable size. [I-D.filsfils-spring-path-tracing] recommends 36 octets for a MCD of 3 bytes (12 MCDs).

In our case, taking into account the alignment requirements, we have the following recommendation.

When MCD is 3 bytes, we recommend 13 MCDs for a total of 39 bytes. When MCD is 4 bytes, we recommend an even number of MCD, for example 10 for a total of 40 bytes or 12 for a total of 48 bytes.

### 6.6.1. Authenticated mode for Compact Path Tracing




In the authenticated mode, an HMAC is appended at the end of the MCD Stack. The size of the HMAC is a multiple of 8 octets (maximum 32 octets).

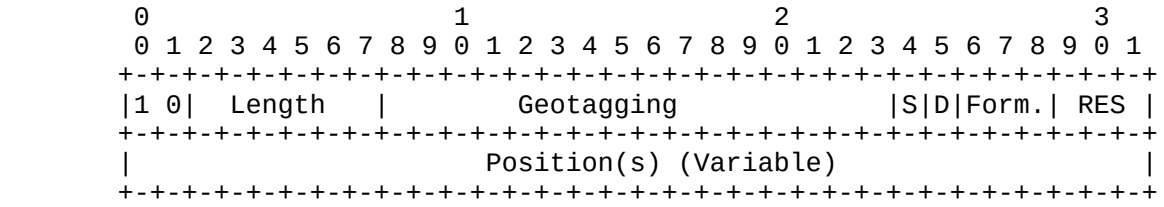
Each intermediate node calculates the HMAC for the whole CPT LTV, including the HMAC field. The newly calculated HMAC then overwrites the previous node's HMAC. The first node will just use an HMAC field set to all zeros. An identifier for every single node is not needed, because it can be derived from the MCD Stack. The destination node can reconstruct the different HMAC fields at each hop to check if the final HMAC is consistent.

### 6.6.2. Geotagged Compact Path Tracing

TODO

## 6.7. Geotagging LTV

We describe an LTV for Semantic Routing use case.




Geotagging for Semantic Routing: EIP extended code (see Table 3 in Section 7.2)

S: (for Source) the Position includes the Source Location

D: (for Destination) the Position includes the Destination Location

Format: 3 bits, specifies the encoding used to represent the geographical location(s).

RES: Reserved, set to 0.

Position(s): Contains the geographical location data. Its length must be a multiple of 4 octets.

| Format | Encoding |
|---|---|
| 0 | Quantized Long: Latitude 32 bits, Longitude 32 bits |
| 1 | Quantized Short: Latitude 16 bits, Longitude 16 bits |
| 2 | Geohash Long 60 bits (padded to 64 bits) |
| 3 | Geohash Short 30 bits (padded to 32 bits) |

Table 1

Quantized format as described in https://mmcloughlin.com/posts/geohash-assembly. Latitude and longitude are quantized by mapping to the unit interval [0, 1] and multiplying by 2^32 for Long format (32+32=64 bits) and by 2^16 for Short format (16+16=32 bits), as illustrated in the following formulas:

```
lat_long = floor( 2^32 * (lat + 90) / 180 )
lon_long = floor( 2^32 * (lon + 180) / 360 )

lat_short = floor( 2^16 * (lat + 90) / 180 )
lon_short = floor( 2^16 * (lon + 180) / 360 )
```

Geohash format is an open source format specified in https://en.wikipedia.org/wiki/Geohash

Quantized Long (Format = 0)
64 bits (32 lat + 32 lon)
LAT error: 賊 2.3 mm
LONG max error: 賊 4.6 mm

Quantized Short (Format = 1)
32 bits (16 lat + 16 lon)
LAT error: 賊 153 m
LONG max error: 賊 305 m

Geohash Long (Format = 2)
60 bits + 4 bits padding
LAT error: 賊 18.6 mm
LONG max error: 賊 9.3 mm

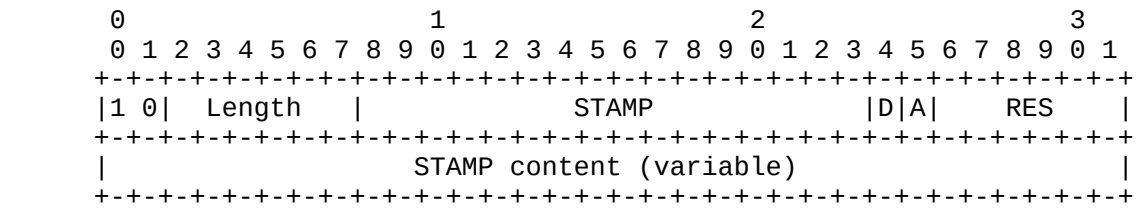
Geohash Short (Format = 3)
30 bits + 2 bits padding
LAT error: 賊 610 m
LONG max error: 賊 305 m

### 6.8. Simple Two-Way Active Measurement Protocol (STAMP) LTV

This is a porting to EIP of the STAMP implementation from RFC8972 (https://www.rfc-editor.org/rfc/rfc8972.html).

The original implementation writes the STAMP data in the UDP payload, if the data is carried by EIP instead, the delay measurement can be performed in-band. This is true when the delay introduced by the reflector is not relevant for the purpose of the measurement. Indeed, the reflected STAMP data will be returned to the sender when another packet is available for piggy-backing the STAMP data.

When the entire round-trip delay is relevant, at least the reflected packet should be sent out-of-band to achieve the smallest possible delay when the Reflector processes the packet.



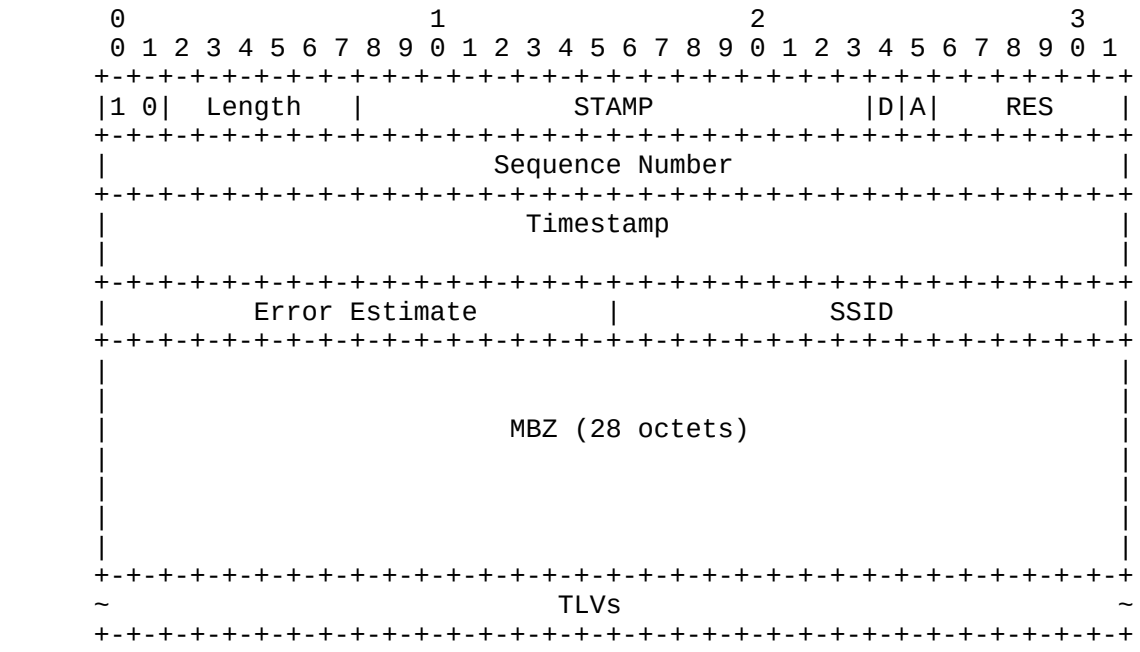

D: Direction, set to 0 if it carries Sender data, 1 for Reflector data.

A: Authenticated, set to 1 for Authenticated Mode.

RES: Reserved, 6 bits set to 0.

The STAMP content is variable. It changes depending on the Direction and Authenticated values.

Following is an example of a Session-Sender STAMP Information Element in Unauthenticated Mode:




The detailed description of the remaining fields and of the other formats, for the Reflector and Authenticated mode, can be found in RFC8972.

## 7. Code assignment for EIP Information Elements (a.k.a. EIP LTVs)

### 7.1. LTV Short Codes

| Information Element | Code |
|---|---|
| Short Identifier | 0x01 |
| Processing Accelerator | 0x02 |
| Timestamps | TBA |

Table 2: Short Codes (1 bytes) for EIP LTVs

### 7.2. LTV Extended Codes

| Information Element | Code |
|---|---|
| HMAC | 0x0001 |
| Compact Path Tracing (CPT) | 0x0002 |
| Long Identifier | 0x0003 |
| Geotagging | 0x0004 |

Table 3: Extended Codes (2 bytes) for EIP LTVs

## 8. Conventions and Definitions

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.

## 9. Security Considerations

EIP introduces in-band information that may be inspected, inserted, or updated by nodes along the packet path. As a consequence, unauthorized access to, or modification of, EIP information can affect telemetry, service behavior, or policy enforcement.

For this reason, EIP is primarily intended for deployment in controlled or limited domains, where the operator can manage protocol support and trust relationships among participating nodes.

The specific security implications may depend on the protocol container used to transport EIP and on the semantics of the EIP Information Elements carried in the packet.

## 10. IANA Considerations

This version of the document does not request IANA actions.

Future versions may request IANA actions related to standalone EIP transport codepoints or to EIP Information Element code assignments. Other approaches to transport EIP, such as IOAM/GOB-based transport, are handled by their corresponding specifications.

## 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/rfc/rfc2119>.

[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/rfc/rfc8174>.

[RFC8754] Filsfils, C., Ed., Dukes, D., Ed., Previdi, S., Leddy, J., Matsushima, S., and D. Voyer, "IPv6 Segment Routing Header (SRH)", RFC 8754, DOI 10.17487/RFC8754, March 2020, <https://www.rfc-editor.org/rfc/rfc8754>.

### 11.2. Informative References

[I-D.draft-eip-arch] Salsano, S., ElBakoury, H., and D. Lopez, "Extensible In-band Processing (EIP) Architecture and Framework", Work in Progress, Internet-Draft, draft-eip-arch-09, 16 March 2026, <https://datatracker.ietf.org/doc/html/draft-eip-arch-09>.

[I-D.draft-mayer-ioam-gob]
Mayer, A. and S. Salsano, "Global Opaque Block for IOAM Pre-allocated Trace Option", Work in Progress, Internet-Draft, draft-mayer-ioam-gob-01, 18 March 2026, <https://datatracker.ietf.org/doc/html/draft-mayer-ioam-gob-01>.

[I-D.filsfils-spring-path-tracing]
Filsfils, C., Abdelsalam, A., Camarillo, P., Yufit, M., Graf, T., Su, Y., Matsushima, S., Valentine, M., and Dhamija, "Path Tracing in SRv6 networks", Work in Progress, Internet-Draft, draft-filsfils-spring-path-tracing-05, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-filsfils-spring-path-tracing-05>.

[id-eip-use-cases]
Salsano, S. and H. ElBakoury, "Extensible In-band Processing (EIP) Use Cases", 2022, <https://eip-home.github.io/eip-headers/draft-eip-use-cases.txt>.

## Acknowledgments and Contributors

The authors would like to thank the following people who contributed to this work.

Giulio Sidoretti has contributed to the section on Compact Path Tracing and to a prototype eBPF-based implementation of EIP.

Carmine Scarpitta has contributed to an early prototype implementation of EIP encoding in Linux.

Hesham ElBakoury has contributed to the design of the EIP framework.

### Contributors’ Addresses

Hesham ElBakoury Consultant EMail: helbakoury@gmail.com

Giulio Sidoretti Univ. of Rome Tor Vergata / CNIT EMail: giulio.sidoretti@uniroma2.it

Carmine Scarpitta Univ. of Rome Tor Vergata / CNIT (at the time of contribution) Current affiliation: Cisco Systems EMail: cscarpit@cisco.com

## Authors’ Addresses

Stefano Salsano
Univ. of Rome Tor Vergata / CNIT
Email: stefano.salsano@uniroma2.it

Andrea Mayer
Univ. of Rome Tor Vergata / CNIT / Common Net
Email: andrea.mayer@uniroma2.it