Stream: Internet Engineering Task Force (IETF)
RFC: 9197
Category: Standards Track
Published: May 2022
ISSN: 2070-1721
Authors: F. Brockners, Ed. *Cisco*
S. Bhandari, Ed. *Thoughtspot*
T. Mizrahi, Ed. *Huawei*

RFC 9197
Data Fields for In Situ Operations, Administration, and Maintenance (IOAM)

Abstract

In situ Operations, Administration, and Maintenance (IOAM) collects operational and telemetry information in the packet while the packet traverses a path between two points in the network. This document discusses the data fields and associated data types for IOAM. IOAM-Data-Fields can be encapsulated into a variety of protocols, such as Network Service Header (NSH), Segment Routing, Generic Network Virtualization Encapsulation (Geneve), or IPv6. IOAM can be used to complement OAM mechanisms based on, e.g., ICMP or other types of probe packets.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9197.

Copyright Notice

Table of Contents

1. Introduction

This document defines data fields for In situ Operations, Administration, and Maintenance (IOAM). IOAM records OAM information within the packet while the packet traverses a particular network domain. The term "in situ" refers to the fact that the OAM data is added to the data packets rather than being sent within packets specifically dedicated to OAM. IOAM is used to complement mechanisms, such as Ping or Traceroute. In terms of "active" or "passive" OAM, IOAM can be considered a hybrid OAM type. "In situ" mechanisms do not require extra packets to be sent. IOAM adds information to the already available data packets and therefore cannot be

considered passive. In terms of the classification given in [RFC7799], IOAM could be portrayed as Hybrid Type I. IOAM mechanisms can be leveraged where mechanisms using, e.g., ICMP do not apply or do not offer the desired results, such as proving that a certain traffic flow takes a predefined path, Service Level Agreement (SLA) verification for the data traffic, detailed statistics on traffic distribution paths in networks that distribute traffic across multiple paths, or scenarios in which probe traffic is potentially handled differently from regular data traffic by the network devices.

The term "in situ OAM" was originally motivated by the use of OAM-related mechanisms that add information into a packet. This document uses IOAM as a term defining the IOAM technology. IOAM includes "in situ" mechanisms but also mechanisms that could trigger the creation of additional packets dedicated to OAM.

2. Conventions

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.

Abbreviations and definitions used in this document:

E2E:	Edge to Edge
Geneve:	Generic Network Virtualization Encapsulation [RFC8926]
IOAM:	In situ Operations, Administration, and Maintenance
MTU:	Maximum Transmission Unit
NSH:	Network Service Header [RFC8300]
OAM:	Operations, Administration, and Maintenance
PMTU:	Path MTU
POT:	Proof of Transit
Short format:	refers to an IOAM-Data-Field that comprises 4 octets
SID:	Segment Identifier
SR:	Segment Routing
VXLAN-GPE:	Virtual eXtensible Local Area Network, Generic Protocol Extension [NVO3-VXLAN-GPE]
Wide format:	refers to an IOAM-Data-Field that comprises 8 octets

3. Scope, Applicability, and Assumptions

IOAM assumes a set of constraints as well as guiding principles and concepts that go hand in hand with the definition of the IOAM-Data-Fields. These constraints, guiding principles, and concepts are described in this section. A discussion of how IOAM-Data-Fields and the associated concepts are applied to an IOAM deployment are out of scope for this document. Please refer to [IPPM-IOAM-DEPLOYMENT] for IOAM deployment considerations.

Scope:
: This document defines the data fields and associated data types for IOAM. The IOAM-Data-Fields can be encapsulated in a variety of protocols, including NSH, Segment Routing, Geneve, and IPv6. Specification details for these different protocols are outside the scope of this document. It is expected that each such encapsulation would be specified by an RFC and jointly designed by the working group that develops or maintains the encapsulation protocol and the IETF IP Performance Measurement (IPPM) Working Group.

Domain (or scope) of in situ OAM deployment:
: IOAM is focused on "limited domains", as defined in [RFC8799]. For IOAM, a limited domain could, for example, be an enterprise campus using physical connections between devices or an overlay network using virtual connections/tunnels for connectivity between said devices. A limited domain that uses IOAM may constitute one or multiple "IOAM-Domains", each disambiguated through separate namespace identifiers. An IOAM-Domain is bounded by its perimeter or edge. IOAM-Domains may overlap inside the limited domain. Designers of protocol encapsulations for IOAM specify mechanisms to ensure that IOAM data stays within an IOAM-Domain. In addition, the operator of such a domain is expected to put provisions in place to ensure that IOAM data does not leak beyond the edge of an IOAM-Domain using, for example, packet filtering methods. The operator **SHOULD** consider the potential operational impact of IOAM to mechanisms, such as ECMP processing (e.g., load-balancing schemes based on packet length could be impacted by the increased packet size due to IOAM), PMTU (i.e., ensure that the MTU of all links within a domain is sufficiently large to support the increased packet size due to IOAM), and ICMP message handling (i.e., in case of IPv6, IOAM support for ICMPv6 echo request/reply is desired, which would translate into ICMPv6 extensions to enable IOAM-Data-Fields to be copied from an echo request message to an echo reply message).

IOAM control points:
: IOAM-Data-Fields are added to or removed from the user traffic by the devices that form the edge of a domain. Devices that form an IOAM-Domain can add, update, or remove IOAM-Data-Fields. Edge devices of an IOAM-Domain can be hosts or network devices.

Traffic sets that IOAM is applied to:
: IOAM can be deployed on all or only on subsets of the user traffic. Using IOAM on a selected set of traffic (e.g., per interface, based on an access control list or flow specification defining a specific set of traffic, etc.) could be useful in deployments where the cost of processing IOAM-

Data-Fields by encapsulating, transit, or decapsulating nodes might be a concern from a performance or operational perspective. Thus, limiting the amount of traffic IOAM is applied to could be beneficial in some deployments.

Encapsulation independence:
: The definition of IOAM-Data-Fields is independent from the protocols the IOAM-Data-Fields are encapsulated into. IOAM-Data-Fields can be encapsulated into several encapsulating protocols.

Layering:
: If several encapsulation protocols (e.g., in case of tunneling) are stacked on top of each other, IOAM-Data-Fields could be present at multiple layers. The behavior follows the "ships-in-the-night" model, i.e., IOAM-Data-Fields in one layer are independent from IOAM-Data-Fields in another layer. Layering allows operators to instrument the protocol layer they want to measure. The different layers could, but do not have to, share the same IOAM encapsulation mechanisms.

IOAM implementation:
: The definition of the IOAM-Data-Fields takes the specifics of devices with hardware data planes and software data planes into account.

4. IOAM Data-Fields, Types, and Nodes

This section details IOAM-related nomenclature and describes data types, such as IOAM-Data-Fields, IOAM-Types, IOAM-Namespaces, as well as the different types of IOAM nodes.

4.1. IOAM Data-Fields and Option-Types

An IOAM-Data-Field is a set of bits with a defined format and meaning, which can be stored at a certain place in a packet for the purpose of IOAM.

To accommodate the different uses of IOAM, IOAM-Data-Fields fall into different categories. In IOAM, these categories are referred to as "IOAM-Option-Types". A common registry is maintained for IOAM-Option-Types (see Section 7.1 for details). Corresponding to these IOAM-Option-Types, different IOAM-Data-Fields are defined.

This document defines four IOAM-Option-Types:

- Pre-allocated Trace Option-Type
- Incremental Trace Option-Type
- POT Option-Type
- E2E Option-Type

Future IOAM-Option-Types can be allocated by IANA, as described in Section 7.1.

4.2. IOAM-Domains and Types of IOAM Nodes

Section 3 already mentioned that IOAM is expected to be deployed in a limited domain [RFC8799]. One or more IOAM-Option-Types are added to a packet upon entering an IOAM-Domain and are removed from the packet when exiting the domain. Within the IOAM-Domain, the IOAM-Data-Fields **MAY** be updated by network nodes that the packet traverses. An IOAM-Domain consists of "IOAM encapsulating nodes", "IOAM decapsulating nodes", and "IOAM transit nodes". The role of a node (i.e., encapsulating, transit, and decapsulating) is defined within an IOAM-Namespace (see below). A node can have different roles in different IOAM-Namespaces.

A device that adds at least one IOAM-Option-Type to the packet is called an "IOAM encapsulating node", whereas a device that removes an IOAM-Option-Type is referred to as an "IOAM decapsulating node". Nodes within the domain that are aware of IOAM data and read, write, and/or process IOAM data are called "IOAM transit nodes". IOAM nodes that add or remove the IOAM-Data-Fields can also update the IOAM-Data-Fields at the same time. Or, in other words, IOAM encapsulating or decapsulating nodes can also serve as IOAM transit nodes at the same time. Note that not every node in an IOAM-Domain needs to be an IOAM transit node. For example, a deployment might require that packets traverse a set of firewalls that support IOAM. In that case, only the set of firewall nodes would be IOAM transit nodes, rather than all nodes.

An IOAM encapsulating node incorporates one or more IOAM-Option-Types (from the list of IOAM-Types, see Section 7.1) into packets that IOAM is enabled for. If IOAM is enabled for a selected subset of the traffic, the IOAM encapsulating node is responsible for applying the IOAM functionality to the selected subset.

An IOAM transit node reads, writes, and/or processes one or more of the IOAM-Data-Fields. If both the Pre-allocated and the Incremental Trace Option-Types are present in the packet, each IOAM transit node, based on configuration and available implementation of IOAM, might populate IOAM trace data in either a Pre-allocated or Incremental Trace Option-Type but not both. Note that not populating any of the Trace Option-Types is also valid behavior for an IOAM transit node. A transit node **MUST** ignore IOAM-Option-Types that it does not understand. A transit node **MUST NOT** add new IOAM-Option-Types to a packet, **MUST NOT** remove IOAM-Option-Types from a packet, and **MUST NOT** change the IOAM-Data-Fields of an IOAM Edge-to-Edge Option-Type.

An IOAM decapsulating node removes IOAM-Option-Type(s) from packets.

The role of an IOAM encapsulating, IOAM transit, or IOAM decapsulating node is always performed within a specific IOAM-Namespace. This means that an IOAM node that is, e.g., an IOAM decapsulating node for IOAM-Namespace "A" but not for IOAM-Namespace "B" will only remove the IOAM-Option-Types for IOAM-Namespace "A" from the packet. Note that this applies even for IOAM-Option-Types that the node does not understand, for example, an IOAM-Option-Type other than the four described above, which is added in a future revision.

IOAM-Namespaces allow for a namespace-specific definition and interpretation of IOAM-Data-Fields. An interface identifier could, for example, point to a physical interface (e.g., to understand which physical interface of an aggregated link is used when receiving or transmitting a packet), whereas, in another case, it could refer to a logical interface (e.g., in case of tunnels). Please refer to Section 4.3 for details on IOAM-Namespaces.

4.3. IOAM-Namespaces

IOAM-Namespaces add further context to IOAM-Option-Types and associated IOAM-Data-Fields. The IOAM-Option-Types and associated IOAM-Data-Fields are interpreted as defined in this document, regardless of the value of the IOAM-Namespace. However, IOAM-Namespaces provide a way to group nodes to support different deployment approaches of IOAM (see a few example use cases below). IOAM-Namespaces also help to resolve potential issues that can occur due to IOAM-Data-Fields not being globally unique (e.g., IOAM node identifiers do not have to be globally unique). The significance of IOAM-Data-Fields is always within a particular IOAM-Namespace. Given that IOAM-Data-Fields are always interpreted as the context of a specific namespace, the Namespace-ID field always needs to be carried along with the IOAM data-fields themselves.

An IOAM-Namespace is identified by a 16-bit namespace identifier (Namespace-ID). The IOAM-Namespace field is included in all the IOAM-Option-Types defined in this document and **MUST** be included in all future IOAM-Option-Types. The Namespace-ID value is divided into two subranges:

- an operator-assigned range from 0x0001 to 0x7FFF and
- an IANA-assigned range from 0x8000 to 0xFFFF.

The IANA-assigned range is intended to allow future extensions to have new and interoperable IOAM functionality, while the operator-assigned range is intended to be domain specific and managed by the network operator. The Namespace-ID value of 0x0000 is the "Default-Namespace-ID". The Default-Namespace-ID indicates that no specific namespace is associated with the IOAM-Data-Fields in the packet. The Default-Namespace-ID **MUST** be supported by all nodes implementing IOAM. A use case for the Default-Namespace-ID are deployments that do not leverage specific namespaces for some or all of their packets that carry IOAM-Data-Fields.

Namespace identifiers allow devices that are IOAM capable to determine:

- whether one or more IOAM-Option-Types need to be processed by a device. If the Namespace-ID contained in a packet does not match any Namespace-ID the node is configured to operate on, then the node **MUST NOT** change the contents of the IOAM-Data-Fields.
- which IOAM-Option-Type needs to be processed/updated in case there are multiple IOAM-Option-Types present in the packet. Multiple IOAM-Option-Types can be present in a packet in case of overlapping IOAM-Domains or in case of a layered IOAM deployment.
- whether one or more IOAM-Option-Types have to be removed from the packet, e.g., at a domain edge or domain boundary.

IOAM-Namespaces support several different uses:

- IOAM-Namespaces can be used by an operator to distinguish different IOAM-Domains. Devices at edges of an IOAM-Domain can filter on Namespace-IDs to provide for proper IOAM-Domain isolation.
- IOAM-Namespaces provide additional context for IOAM-Data-Fields and, thus, can be used to ensure that IOAM-Data-Fields are unique and are interpreted properly by management stations or network controllers. The node identifier field (node_id, see below) does not need to be unique in a deployment. This could be the case if an operator wishes to use different node identifiers for different IOAM layers, even within the same device, or node identifiers might not be unique for other organizational reasons, such as after a merger of two formerly separated organizations. The Namespace-ID can be used as a context identifier, such that the combination of node_id and Namespace-ID will always be unique.
- Similarly, IOAM-Namespaces can be used to define how certain IOAM-Data-Fields are interpreted; IOAM offers three different timestamp format options. The Namespace-ID can be used to determine the timestamp format. IOAM-Data-Fields (e.g., buffer occupancy) that do not have a unit associated are to be interpreted within the context of an IOAM-Namespace.
- IOAM-Namespaces can be used to identify different sets of devices (e.g., different types of devices) in a deployment; if an operator wants to insert different IOAM-Data-Fields based on the device, the devices could be grouped into multiple IOAM-Namespaces. This could be due to the fact that the IOAM feature set differs between different sets of devices, or it could be for reasons of optimized space usage in the packet header. It could also stem from hardware or operational limitations on the size of the trace data that can be added and processed, preventing collection of a full trace for a flow.
- By assigning different IOAM Namespace-IDs to different sets of nodes or network partitions and using a separate instance of an IOAM-Option-Type for each Namespace-ID, a full trace for a flow could be collected and constructed via partial traces from each IOAM-Option-Type in each of the packets in the flow. For example, an operator could choose to group the devices of a domain into two IOAM-Namespaces in a way that each IOAM-Namespace is represented by one of two IOAM-Option-Types in the packet. Each node would record data only for the IOAM-Namespace that it belongs to, ignoring the other IOAM-Option-Type with an IOAM-Namespace to which it doesn't belong. To retrieve a full view of the deployment, the captured IOAM-Data-Fields of the two IOAM-Namespaces need to be correlated.

4.4. IOAM Trace Option-Types

In a typical deployment, all nodes in an IOAM-Domain would participate in IOAM; thus, they would be IOAM transit nodes, IOAM encapsulating nodes, or IOAM decapsulating nodes. If not all nodes within a domain support IOAM functionality as defined in this document, IOAM tracing information (i.e., node data, see below) can only be collected on those nodes that support IOAM functionality as defined in this document. Nodes that do not support IOAM functionality as defined in this document will forward the packet without any changes to the IOAM-Data-Fields. The maximum number of hops and the minimum PMTU of the IOAM-Domain is assumed to be

known. An overflow indicator (O-bit) is defined as one of the ways to deal with situations where the PMTU was underestimated, i.e., where the number of hops that are IOAM capable exceeds the available space in the packet.

To optimize hardware and software implementations, IOAM tracing is defined as two separate options. A deployment can choose to configure and support one or both of the following options.

Pre-allocated Trace-Option:
: This trace option is defined as a container of node data fields (see below) with pre-allocated space for each node to populate its information. This option is useful for implementations where it is efficient to allocate the space once and index into the array to populate the data during transit (e.g., software forwarders often fall into this class). The IOAM encapsulating node allocates space for the Pre-allocated Trace Option-Type in the packet and sets corresponding fields in this IOAM-Option-Type. The IOAM encapsulating node allocates an array that is used to store operational data retrieved from every node while the packet traverses the domain. IOAM transit nodes update the content of the array and possibly update the checksums of outer headers. A pointer that is part of the IOAM trace data points to the next empty slot in the array. An IOAM transit node that updates the content of the Pre-allocated Trace-Option also updates the value of the pointer, which specifies where the next IOAM transit node fills in its data. The "node data list" array (see below) in the packet is populated iteratively as the packet traverses the network, starting with the last entry of the array, i.e., "node data list [n]" is the first entry to be populated, "node data list [n-1]" is the second one, etc.

Incremental Trace-Option:
: This trace option is defined as a container of node data fields, where each node allocates and pushes its node data immediately following the option header. This type of trace recording is useful for some of the hardware implementations, as it eliminates the need for the transit network elements to read the full array in the option and allows for as arbitrarily long packets as the MTU allows. The IOAM encapsulating node allocates space for the Incremental Trace Option-Type. Based on the operational state and configuration, the IOAM encapsulating node sets the fields in the Option-Type that control what IOAM-Data-Fields have to be collected and how large the node data list can grow. IOAM transit nodes push their node data to the node data list subject to any protocol constraints of the encapsulating layer. They then decrease the remaining length available to subsequent nodes and adjust the lengths and possibly checksums in outer headers.

IOAM encapsulating nodes and IOAM decapsulating nodes that support tracing **MUST** support both Trace Option-Types. For IOAM transit nodes, it is sufficient to support one of the Trace Option-Types. In the event that both options are utilized in a deployment at the same time, the Incremental Trace-Option **MUST** be placed before the Pre-allocated Trace-Option. Deployments that mix devices with either the Incremental Trace-Option or the Pre-allocated Trace-Option could result in both Option-Types being present in a packet. Given that the operator knows which equipment is deployed in a particular IOAM-Domain, the operator will decide by means of configuration which type(s) of trace options will be used for a particular domain.

Every node data entry holds information for a particular IOAM transit node that is traversed by a packet. The IOAM decapsulating node removes the IOAM-Option-Types and processes and/or exports the associated data. Like all IOAM-Data-Fields, the IOAM-Data-Fields of the IOAM Trace Option-Types are defined in the context of an IOAM-Namespace.

IOAM tracing can collect the following types of information:

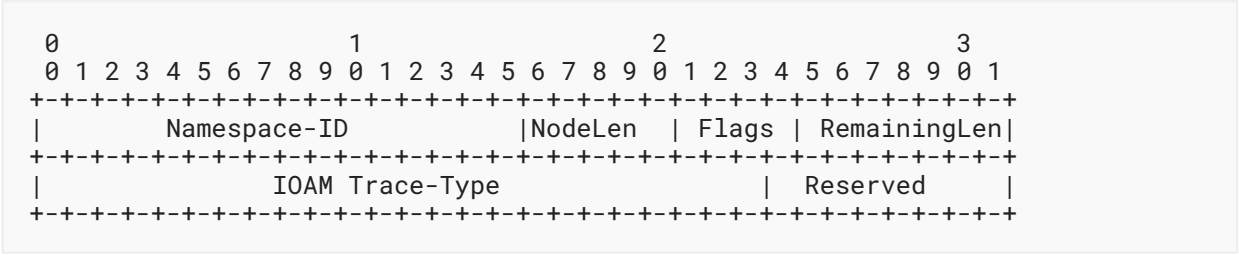
- Identification of the IOAM node. An IOAM node identifier can match to a device identifier or a particular control point or subsystem within a device.
- Identification of the interface that a packet was received on, i.e., ingress interface.
- Identification of the interface that a packet was sent out on, i.e., egress interface.
- Time of day when the packet was processed by the node, as well as the transit delay. Different definitions of processing time are feasible and expected, though it is important that all devices of an IOAM-Domain follow the same definition.
- Generic data, i.e., format-free information where syntax and semantics of the information is defined by the operator in a specific deployment. For a specific IOAM-Namespace, all IOAM nodes have to interpret the generic data the same way. Examples for generic IOAM data include geolocation information (location of the node at the time the packet was processed), buffer queue fill level or cache fill level at the time the packet was processed, or even a battery-charge level.
- Information to detect whether IOAM trace data was added at every hop or whether certain hops in the domain weren't IOAM transit nodes.

It should be noted that the semantics of some of the node data fields that are defined below, such as the queue depth and buffer occupancy, are implementation specific. This approach is intended to allow IOAM nodes with various different architectures.

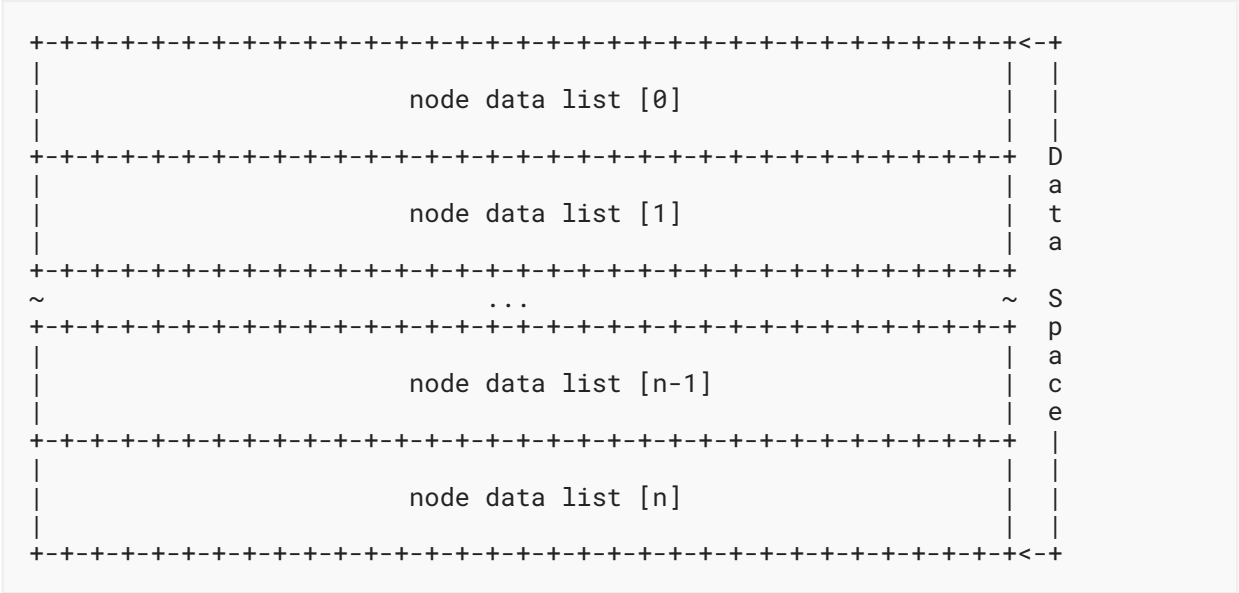
4.4.1. Pre-allocated and Incremental Trace Option-Types

The IOAM Pre-allocated Trace-Option and the IOAM Incremental Trace-Option have similar formats. Except where noted below, the internal formats and fields of the two trace options are identical. Both trace options consist of a fixed-size "trace option header" and a variable data space to store gathered data, i.e., the "node data list". An IOAM transit node (that is, not an IOAM encapsulating node or IOAM decapsulating node) **MUST NOT** modify any of the fields in the fixed-size "trace option header", other than Flags" and "RemainingLen", i.e., an IOAM transit node **MUST NOT** modify the Namespace-ID, NodeLen, IOAM Trace-Type, or Reserved fields.

The Pre-allocated and Incremental Trace-Option headers:



The trace option data **MUST** be aligned by 4 octets:



Namespace-ID:
: 16-bit identifier of an IOAM-Namespace. The Namespace-ID value of 0x0000 is defined as the "Default-Namespace-ID" (see Section 4.3) and **MUST** be known to all the nodes implementing IOAM. For any other Namespace-ID value that does not match any Namespace-ID the node is configured to operate on, the node **MUST NOT** change the contents of the IOAM-Data-Fields.

NodeLen:
: 5-bit unsigned integer. This field specifies the length of data added by each node in multiples of 4 octets, excluding the length of the "Opaque State Snapshot" field.

  If IOAM Trace-Type Bit 22 is not set, then NodeLen specifies the actual length added by each node. If IOAM Trace-Type Bit 22 is set, then the actual length added by a node would be (NodeLen + length of the "Opaque State Snapshot" field) in 4-octet units.

  For example, if 3 IOAM Trace-Type bits are set and none of them are in wide format, then NodeLen would be 3. If 3 IOAM Trace-Type bits are set and 2 of them are wide, then NodeLen would be 5.

  An IOAM encapsulating node **MUST** set NodeLen.

  A node receiving an IOAM Pre-allocated or Incremental Trace-Option relies on the NodeLen value.

Flags:
: 4-bit field. Flags are allocated by IANA, as specified in Section 7.3. This document allocates a single flag as follows:

Bit 0:
: "Overflow" (O-bit) (most significant bit). In case a network element is supposed to add node data to a packet but detects that there are not enough octets left to record the node data, the network element **MUST NOT** add any fields and **MUST** set the overflow "O-bit" to "1" in the IOAM Trace-Option header. This is useful for transit nodes to ignore further processing of the option.

RemainingLen:
: 7-bit unsigned integer. This field specifies the data space in multiples of 4 octets remaining for recording the node data before the node data list is considered to have overflowed. The sender **MUST** assign the initial value of the RemainingLen field. The sender **MAY** calculate the value of the RemainingLen field by computing the number of node data bytes allowed before exceeding the PMTU, given that the PMTU is known to the sender. Subsequent nodes can carry out a simple comparison between RemainingLen and NodeLen, along with the length of the "Opaque State Snapshot", if applicable, to determine whether or not data can be added by this node. When node data is added, the node **MUST** decrease RemainingLen by the amount of data added. In the Pre-allocated Trace-Option, RemainingLen is used to derive the offset in data space to record the node data element. Specifically, the recording of the node data element would start from RemainingLen - NodeLen - size of (opaque snapshot) in 4-octet units. If RemainingLen in a Pre-allocated Trace-Option exceeds the length of the option, as specified in the lower-layer header (which is not within the scope of this document), then the node **MUST NOT** add any fields.

IOAM Trace-Type:
: 24-bit identifier that specifies which data types are used in this node data list.

  The IOAM Trace-Type value is a bit field. The following bits are defined in this document, with details on each bit described in Section 4.4.2. The order of packing the data fields in each node data element follows the bit order of the IOAM Trace-Type field as follows:

  Bit 0
  : Most significant bit. When set, indicates the presence of Hop_Lim and node_id (short format) in the node data.

  Bit 1
  : When set, indicates the presence of ingress_if_id and egress_if_id (short format) in the node data.

  Bit 2
  : When set, indicates the presence of timestamp seconds in the node data.

  Bit 3
  : When set, indicates the presence of timestamp fraction in the node data.

  Bit 4
  : When set, indicates the presence of transit delay in the node data.

  Bit 5
  : When set, indicates the presence of IOAM-Namespace-specific data in short format in the node data.

  Bit 6
  : When set, indicates the presence of queue depth in the node data.

  Bit 7
  : When set, indicates the presence of the Checksum Complement node data.

Bit 8 When set, indicates the presence of Hop_Lim and node_id in wide format in the node data.

Bit 9 When set, indicates the presence of ingress_if_id and egress_if_id in wide format in the node data.

Bit 10 When set, indicates the presence of IOAM-Namespace-specific data in wide format in the node data.

Bit 11 When set, indicates the presence of buffer occupancy in the node data.

Bits 12-21 Undefined. These values are available for future assignment in the IOAM Trace-Type Registry (Section 7.2). Every future node data field corresponding to one of these bits **MUST** be 4 octets long. An IOAM encapsulating node **MUST** set the value of each undefined bit to 0. If an IOAM transit node receives a packet with one or more of these bits set to 1, it **MUST** either:

1. add corresponding node data filled with the reserved value 0xFFFFFFFF after the node data fields for the IOAM Trace-Type bits defined above, such that the total node data added by this node in units of 4 octets is equal to NodeLen or
2. not add any node data fields to the packet, even for the IOAM Trace-Type bits defined above.

Bit 22 When set, indicates the presence of the variable-length Opaque State Snapshot field.

Bit 23 Reserved; **MUST** be set to zero upon transmission and be ignored upon receipt. This bit is reserved to allow for future extensions of the IOAM Trace-Type bit field.

Section 4.4.2 describes the IOAM-Data-Types and their formats. Within an IOAM-Domain, possible combinations of these bits making the IOAM Trace-Type can be restricted by configuration knobs.

Reserved:
8 bits. An IOAM encapsulating node **MUST** set the value to zero upon transmission. IOAM transit nodes **MUST** ignore the received value.

Node data List [n]:
Variable-length field. This is a list of node data elements where the content of each node data element is determined by the IOAM Trace-Type. The order of packing the data fields in each node data element follows the bit order of the IOAM Trace-Type field. Each node **MUST** prepend its node data element in front of the node data elements that it received, such that the transmitted node data list begins with this node's data element as the first populated element in the list. The last node data element in this list is the node data of the first IOAM-capable node in the path. Populating the node data list in this way ensures that the order of the node data list is the same for Incremental and Pre-allocated Trace-Options. In the Pre-allocated Trace-Option, the index contained in RemainingLen identifies the offset for current active node data to be populated.

4.4.2. IOAM Node Data Fields and Associated Formats

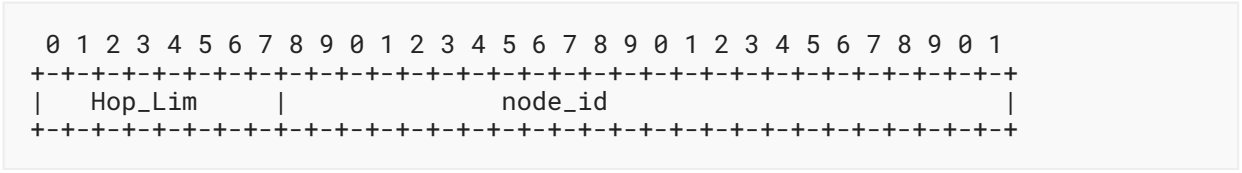
All the IOAM-Data-Fields **MUST** be aligned by 4 octets. If a node that is supposed to update an IOAM-Data-Field is not capable of populating the value of a field set in the IOAM Trace-Type, the field value **MUST** be set to 0xFFFFFFFF for 4-octet fields or 0xFFFFFFFFFFFFFFFF for 8-octet fields, indicating that the value is not populated, except when explicitly specified in the field description below.

Some IOAM-Data-Fields defined below, such as interface identifiers or IOAM-Namespace-specific data, are defined in both "short format" and "wide format". The use of "short format" or "wide format" is not mutually exclusive. A deployment could choose to leverage both. For example, ingress_if_id_(short format) could be an identifier for the physical interface, whereas ingress_if_id_(wide format) could be an identifier for a logical sub-interface of that physical interface.

Data fields and associated data types for each of the IOAM-Data-Fields are specified in the following sections. The definition of IOAM-Data-Fields focuses on the syntax of the data fields and avoids specifying the semantics where feasible. This is why no units are defined for data fields, e.g., like "buffer occupancy" or "queue depth". With this approach, nodes can supply the information in their original format and are not required to perform unit or format conversions. Systems that further process IOAM information, e.g., like a network management system, are assumed to also handle unit conversions as part of their IOAM-Data-Fields processing. The combination of a particular data field and the Namespace-ID provides for the context to interpret the provided data appropriately.

4.4.2.1. Hop_Lim and node_id Short

The "Hop_Lim and node_id short" field is a 4-octet field that is defined as follows:

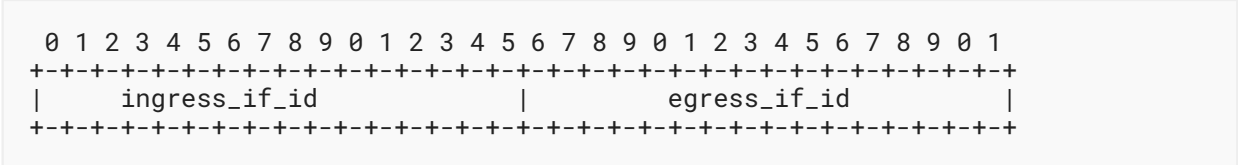


Hop_Lim:
: 1-octet unsigned integer. It is set to the Hop Limit value in the packet at egress from the node that records this data. Hop Limit information is used to identify the location of the node in the communication path. This is copied from the lower layer, e.g., TTL value in IPv4 header or Hop Limit field from IPv6 header of the packet when the packet is ready for transmission. The semantics of the Hop_Lim field depend on the lower-layer protocol that IOAM is encapsulated into; therefore, its specific semantics are outside the scope of this memo. The value of this field **MUST** be set to 0xff when the lower level does not have a field equivalent to TTL / Hop Limit.

node_id:
: 3-octet unsigned integer. A node identifier field to uniquely identify a node within the IOAM-Namespace and associated IOAM-Domain. The procedure to allocate, manage, and map the node_ids is beyond the scope of this document. See [IPPM-IOAM-DEPLOYMENT] for a discussion of deployment-related aspects of the node_id.

4.4.2.2. ingress_if_id and egress_if_id Short

The "ingress_if_id and egress_if_id" field is a 4-octet field that is defined as follows:



ingress_if_id:
: 2-octet unsigned integer. An interface identifier to record the ingress interface the packet was received on.

egress_if_id:
: 2-octet unsigned integer. An interface identifier to record the egress interface the packet is forwarded out of.

Note that due to the fact that IOAM uses its own IOAM-Namespaces for IOAM-Data-Fields, data fields, like interface identifiers, can be used in a flexible way to represent system resources that are associated with ingressing or egressing packets, i.e., ingress_if_id could represent a physical interface, a virtual or logical interface, or even a queue.

4.4.2.3. Timestamp Seconds

The "timestamp seconds" field is a 4-octet unsigned integer field. It contains the absolute timestamp in seconds that specifies the time at which the packet was received by the node. This field has three possible formats, based on either the Precision Time Protocol (PTP) (see e.g., [RFC8877]), NTP [RFC5905], or POSIX [POSIX]. The three timestamp formats are specified in Section 5. In all three cases, the timestamp seconds field contains the 32 most significant bits of the timestamp format that is specified in Section 5. If a node is not capable of populating this field, it assigns the value 0xFFFFFFFF. Note that this is a legitimate value that is valid for 1 second in approximately 136 years; the analyzer has to correlate several packets or compare the timestamp value to its own time of day in order to detect the error indication.

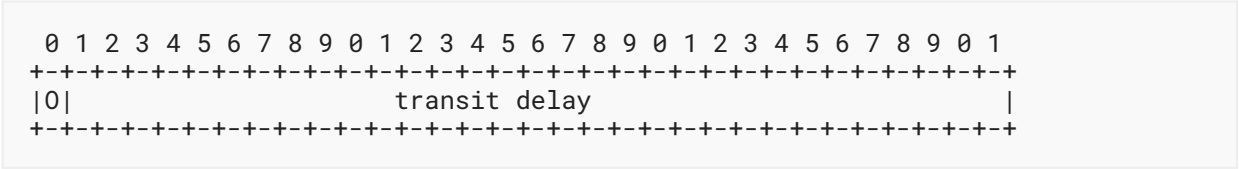
4.4.2.4. Timestamp Fraction

The "timestamp fraction" field is a 4-octet unsigned integer field. Fraction specifies the fractional portion of the number of seconds since the NTP epoch [RFC8877]. The field specifies the time at which the packet was received by the node. This field has three possible formats, based on either PTP (see e.g., [RFC8877]), NTP [RFC5905], or POSIX [POSIX]. The three timestamp formats are specified in Section 5. In all three cases, the timestamp fraction field contains the 32 least

significant bits of the timestamp format that is specified in Section 5. If a node is not capable of populating this field, it assigns the value 0xFFFFFFFF. Note that this is a legitimate value in the NTP format, valid for approximately 233 picoseconds in every second. If the NTP format is used, the analyzer has to correlate several packets in order to detect the error indication.

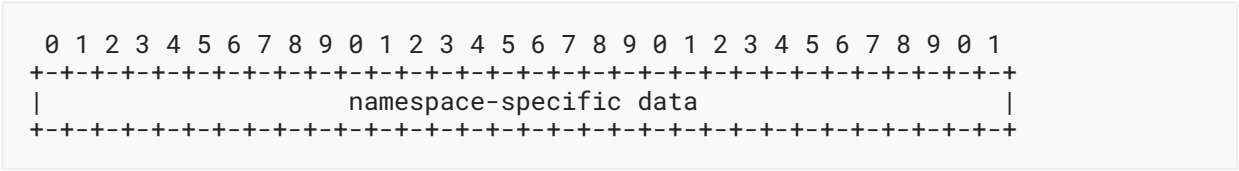
4.4.2.5. Transit Delay

The "transit delay" field is a 4-octet unsigned integer in the range 0 to 2^{31}-1. It is the time in nanoseconds the packet spent in the transit node. This can serve as an indication of the queuing delay at the node. If the transit delay exceeds 2^{31}-1 nanoseconds, then the top bit 'O' is set to indicate overflow and value set to 0x80000000. When this field is part of the data field but a node populating the field is not able to fill it, the field position in the field **MUST** be filled with value 0xFFFFFFFF to mean not populated.



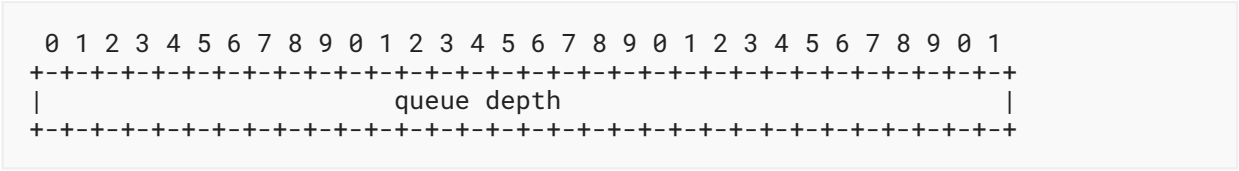
4.4.2.6. Namespace-Specific Data

The "namespace-specific data" field is a 4-octet field that can be used by the node to add IOAM-Namespace-specific data. This represents a "free-format" 4-octet bit field with its semantics defined in the context of a specific IOAM-Namespace.



4.4.2.7. Queue Depth

The "queue depth" field is a 4-octet unsigned integer field. This field indicates the current length of the egress interface queue of the interface from where the packet is forwarded out. The queue depth is expressed as the current amount of memory buffers used by the queue (a packet could consume one or more memory buffers, depending on its size).



4.4.2.8. Checksum Complement

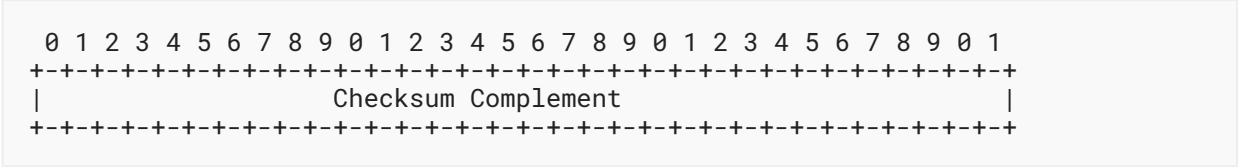
The "Checksum Complement" field is a 4-octet node data that contains the Checksum Complement value. The Checksum Complement is useful when IOAM is transported over encapsulations that make use of a UDP transport, such as VXLAN-GPE or Geneve. Without the

Checksum Complement, nodes adding IOAM node data update the UDP Checksum field following the recommendation of the encapsulation protocols. When the Checksum Complement is present, an IOAM encapsulating node or IOAM transit node adding node data **MUST** carry out one of the following two alternatives in order to maintain the correctness of the UDP Checksum value:

1. recompute the UDP Checksum field or
2. use the Checksum Complement to make a checksum-neutral update in the UDP payload; the Checksum Complement is assigned a value that complements the rest of the node data fields that were added by the current node, causing the existing UDP Checksum field to remain correct.

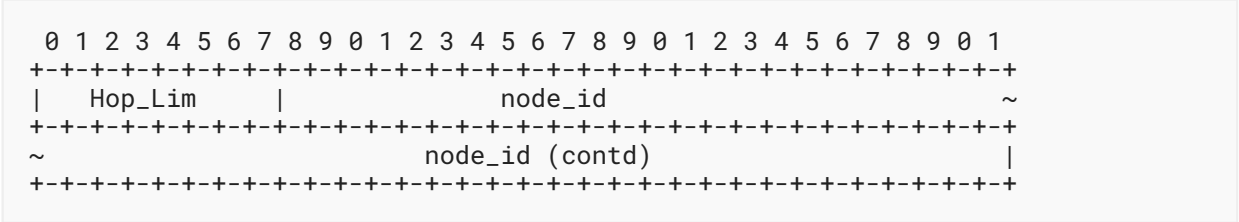
IOAM decapsulating nodes **MUST** recompute the UDP Checksum field, since they do not know whether previous hops modified the UDP Checksum field or the Checksum Complement field.

Checksum Complement fields are used in a similar manner in [RFC7820] and [RFC7821].



4.4.2.9. Hop_Lim and node_id Wide

The "Hop_Lim and node_id wide" field is an 8-octet field defined as follows:

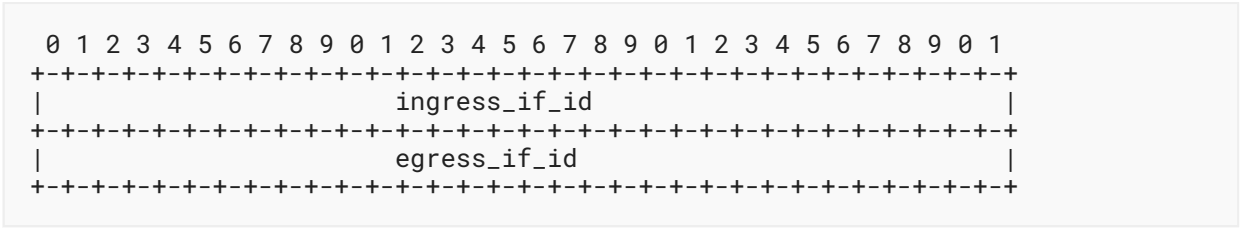


Hop_Lim:
: 1-octet unsigned integer. See Section 4.4.2.1 for the definition of the field.

node_id:
: 7-octet unsigned integer. It is a node identifier field to uniquely identify a node within the IOAM-Namespace and associated IOAM-Domain. The procedure to allocate, manage, and map the node_ids is beyond the scope of this document.

4.4.2.10. ingress_if_id and egress_if_id Wide

The "ingress_if_id and egress_if_id wide" field is an 8-octet field, which is defined as follows:

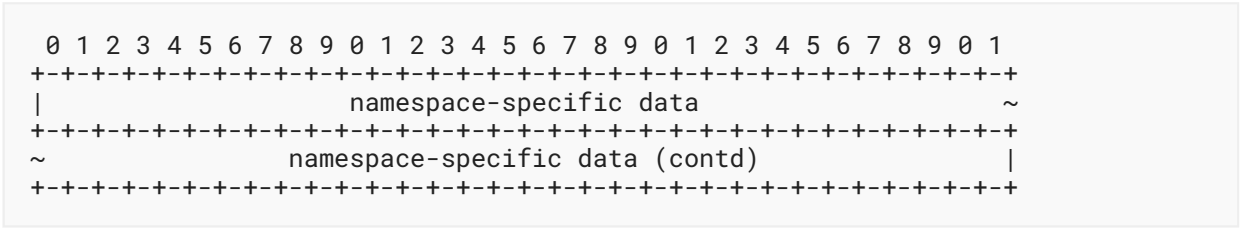


ingress_if_id:
: 4-octet unsigned integer. It is an interface identifier to record the ingress interface the packet was received on.

egress_if_id:
: 4-octet unsigned integer. It is an interface identifier to record the egress interface the packet is forwarded out of.

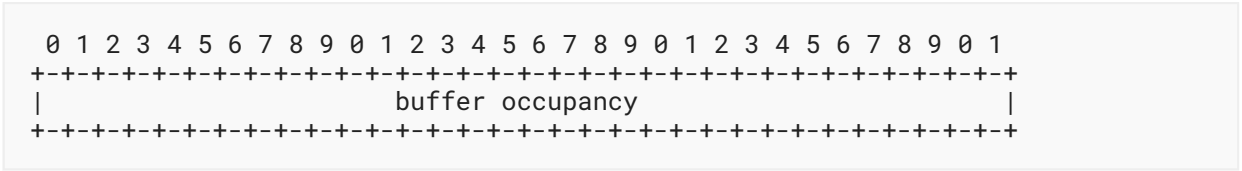
4.4.2.11. Namespace-Specific Data Wide

The "namespace-specific data wide" field is an 8-octet field that can be used by the node to add IOAM-Namespace-specific data. This represents a "free-format" 8-octet bit field with its semantics defined in the context of a specific IOAM-Namespace.



4.4.2.12. Buffer Occupancy

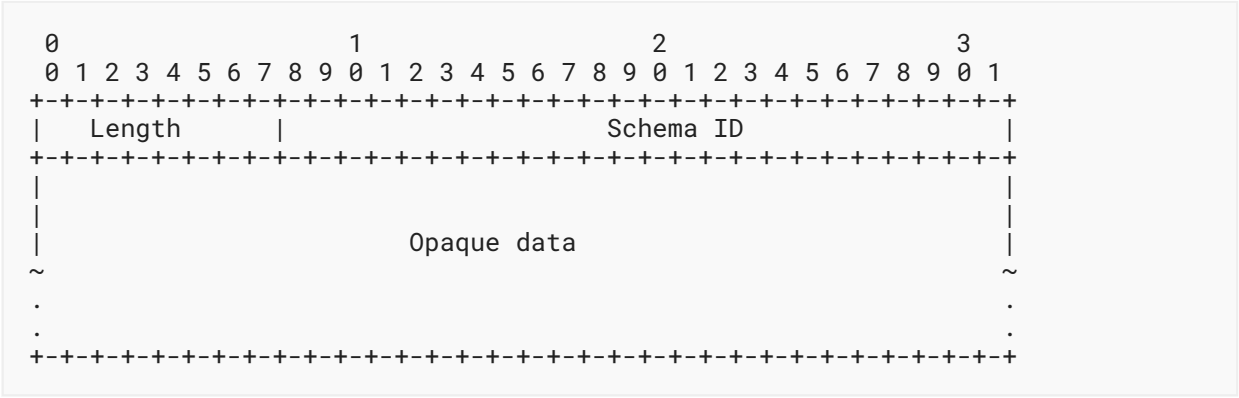
The "buffer occupancy" field is a 4-octet unsigned integer field. This field indicates the current status of the occupancy of the common buffer pool used by a set of queues. The units of this field are implementation specific. Hence, the units are interpreted within the context of an IOAM-Namespace and/or node identifier if used. The authors acknowledge that, in some operational cases, there is a need for the units to be consistent across a packet path through the network; hence, it is recommended for implementations to use standard units, such as bytes.



4.4.2.13. Opaque State Snapshot

The "Opaque State Snapshot" field is a variable-length field and follows the fixed-length IOAM-Data-Fields defined above. It allows the network element to store an arbitrary state in the node data field without a predefined schema. The schema is to be defined within the context of an

IOAM-Namespace. The schema needs to be made known to the analyzer by some out-of-band mechanism. The specification of this mechanism is beyond the scope of this document. A 24-bit "Schema ID" field, interpreted within the context of an IOAM-Namespace, indicates which particular schema is used and has to be configured on the network element by the operator.



Length:
: 1-octet unsigned integer. It is the length in multiples of 4 octets of the Opaque data field that follows Schema ID.

Schema ID:
: 3-octet unsigned integer identifying the schema of Opaque data.

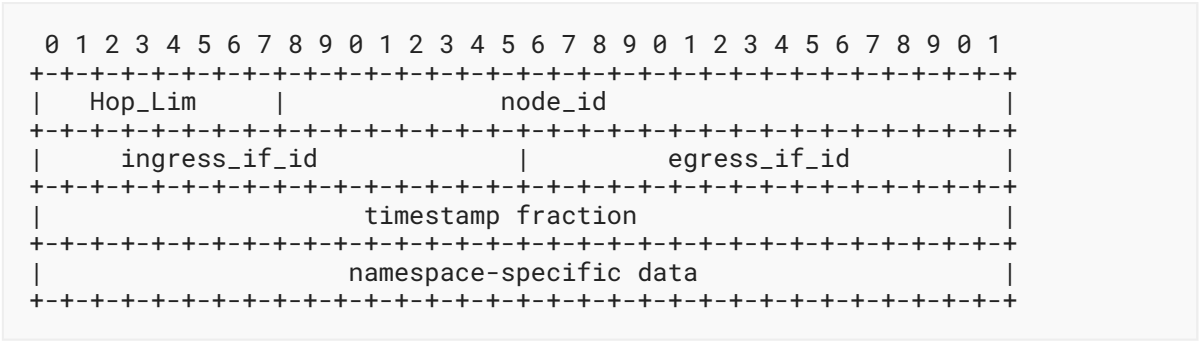
Opaque data:
: Variable-length field. This field is interpreted as specified by the schema identified by the Schema ID.

When this field is part of the data field, but a node populating the field has no opaque state data to report, the Length **MUST** be set to 0 and the Schema ID **MUST** be set to 0xFFFFFF to mean no schema.

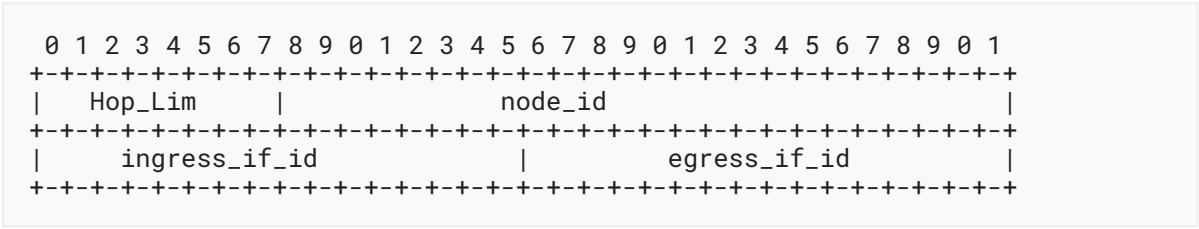
4.4.3. Examples of IOAM Node Data

The format used for the entries in a packet's "node data list" array can vary from packet to packet and deployment to deployment. Some deployments might only be interested in recording the node identifiers, whereas others might be interested in recording node identifiers and timestamps. This section provides example entries of the "node data list" array.

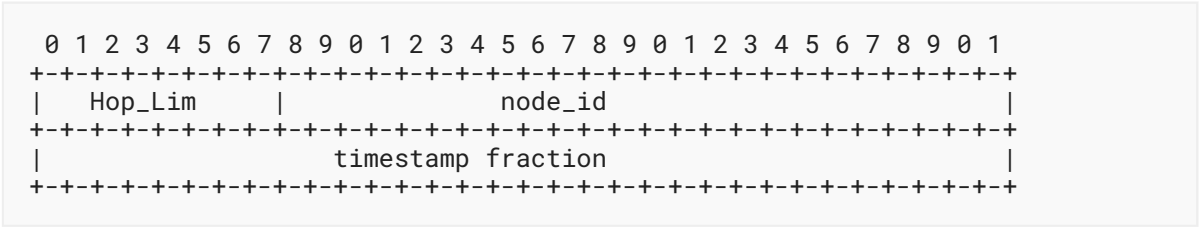
0xD40000:
: If the IOAM Trace-Type is 0xD40000 (0b11010100000000000000000), then the format of node data is:



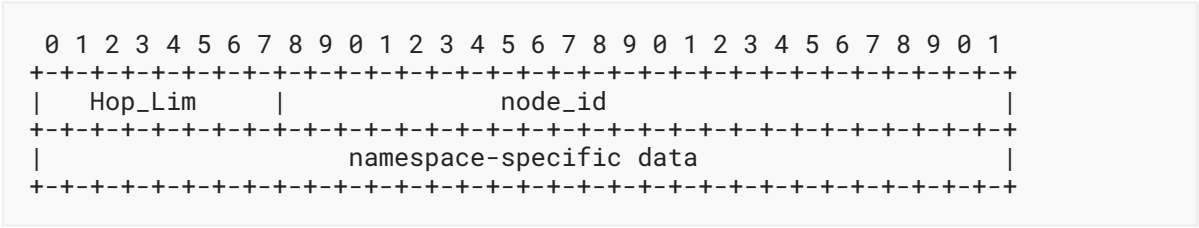
0xC00000: If the IOAM Trace-Type is 0xC00000 (0b110000000000000000000000), then the format is:



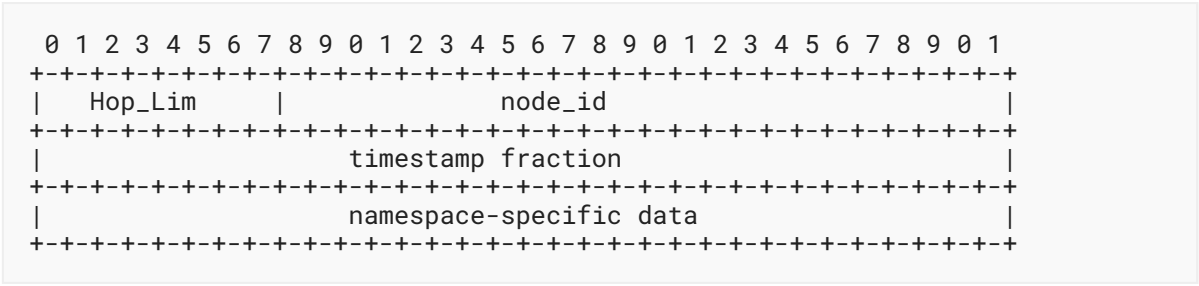
0x900000: If the IOAM Trace-Type is 0x900000 (0b100100000000000000000000), then the format is:



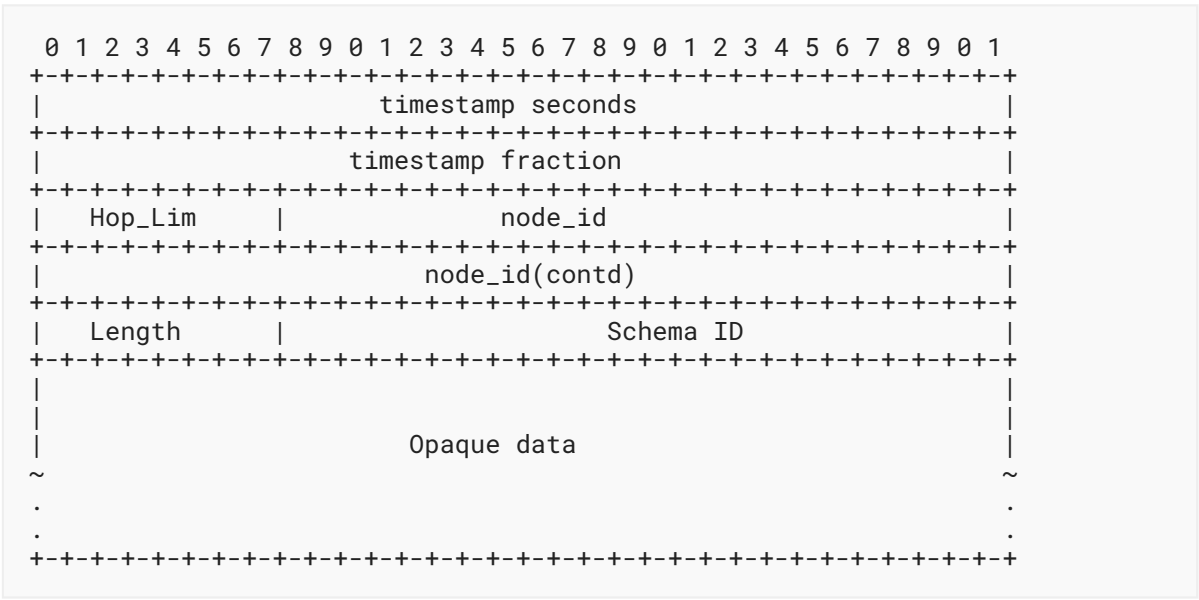
0x840000: If the IOAM Trace-Type is 0x840000 (0b100001000000000000000000), then the format is:



0x940000: If the IOAM Trace-Type is 0x940000 (0b100101000000000000000000), then the format is:



0x308002: If the IOAM Trace-Type is 0x308002 (0b001100001000000000000010), then the format is:



4.5. IOAM Proof of Transit Option-Type

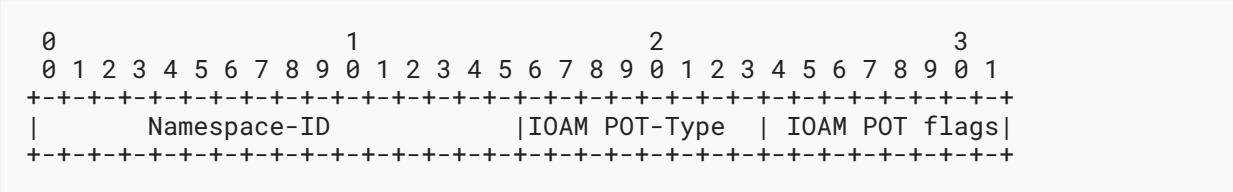
The IOAM Proof of Transit Option-Type is used to support path or service function chain [RFC7665] verification use cases, i.e., prove that traffic transited a defined path. While the details on how the IOAM data for the Proof of Transit Option-Type is processed at IOAM encapsulating, decapsulating, and transit nodes are outside the scope of the document, Proof of Transit approaches share the need to uniquely identify a packet, as well as iteratively operate on a set of information that is handed from node to node. Correspondingly, two pieces of information are added as IOAM-Data-Fields to the packet:

PktID:
: unique identifier for the packet

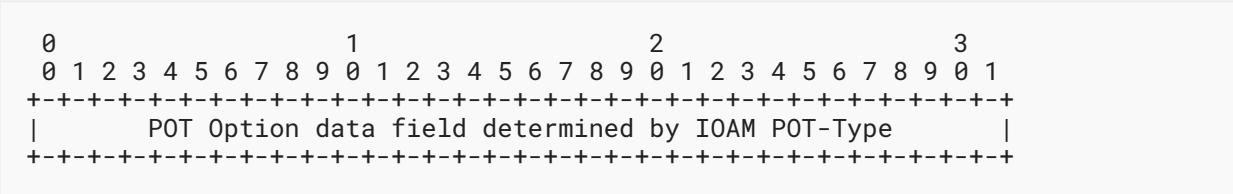
Cumulative:
: information that is handed from node to node and updated by every node according to a verification algorithm

The IOAM Proof of Transit Option-Type consist of a fixed-size "IOAM Proof of Transit Option header" and "IOAM Proof of Transit Option data fields":

IOAM Proof of Transit Option header:



IOAM Proof of Transit Option-Type IOAM-Data-Fields **MUST** be aligned by 4 octets:



Namespace-ID:
: 16-bit identifier of an IOAM-Namespace. The Namespace-ID value of 0x0000 is defined as the "Default-Namespace-ID" (see Section 4.3) and **MUST** be known to all the nodes implementing IOAM. For any other Namespace-ID value that does not match any Namespace-ID the node is configured to operate on, the node **MUST NOT** change the contents of the IOAM-Data-Fields.

IOAM POT-Type:
: 8-bit identifier of a particular POT variant that specifies the POT data that is included. This document defines IOAM POT-Type 0:

  0: POT data is a 16-octet field to carry data associated to POT procedures.

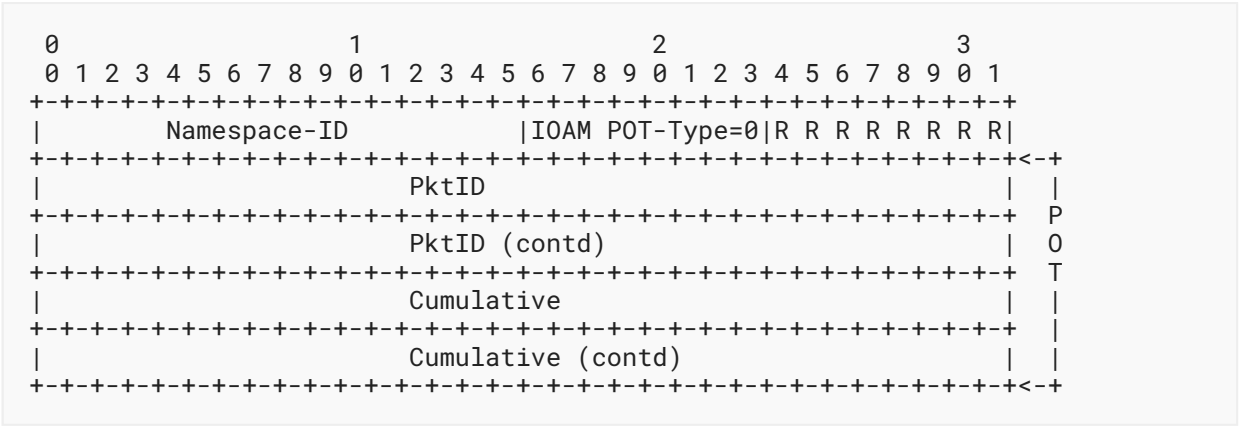
  If a node receives an IOAM POT-Type value that it does not understand, the node **MUST NOT** change, add to, or remove the contents of the IOAM-Data-Fields.

IOAM POT flags:
: 8 bits. This document does not define any flags. Bits 0-7 are available for assignment (see Section 7.5). Bits that have not been assigned **MUST** be set to zero upon transmission and be ignored upon receipt.

POT Option data:
: Variable-length field. The type of which is determined by the IOAM POT-Type.

4.5.1. IOAM Proof of Transit Type 0

IOAM Proof of Transit Option of IOAM POT-Type 0:



Namespace-ID:
: 16-bit identifier of an IOAM-Namespace (see Section 4.3 above).

IOAM POT-Type:
: 8-bit identifier of a particular POT variant that specifies the POT data that is included (see Section 4.5 above). For this case here, IOAM POT-Type is set to the value 0.

Bit 0-7:
: Undefined (see Section 4.5 above).

PktID:
: 64-bit packet identifier.

Cumulative:
: 64-bit Cumulative that is updated at specific nodes by processing per packet PktID field and configured parameters.

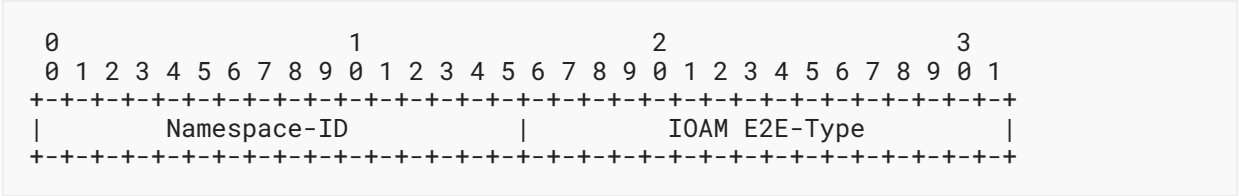
> Note: Larger or smaller sizes of "PktID" and "Cumulative" data are feasible and could be required for certain deployments, e.g., in case of space constraints in the encapsulation protocols used. Future documents could introduce different sizes of data for "Proof of Transit".

4.6. IOAM Edge-to-Edge Option-Type

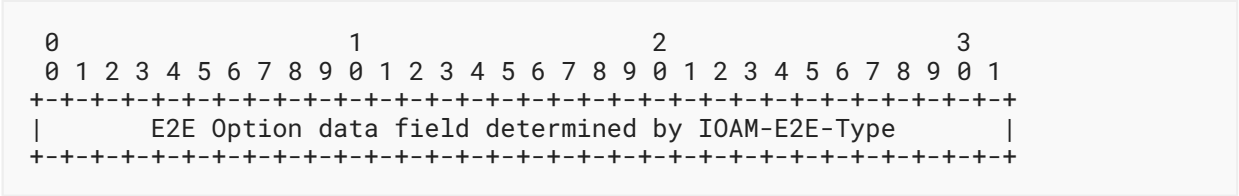
The IOAM Edge-to-Edge Option-Type carries data that is added by the IOAM encapsulating node and interpreted by the IOAM decapsulating node. The IOAM transit nodes **MAY** process the data but **MUST NOT** modify it.

The IOAM Edge-to-Edge Option-Type consist of a fixed-size "IOAM Edge-to-Edge Option-Type header" and "IOAM Edge-to-Edge Option-Type data fields":

IOAM Edge-to-Edge Option-Type header:



The IOAM Edge-to-Edge Option-Type IOAM-Data-Fields **MUST** be aligned by 4 octets:



Namespace-ID:
: 16-bit identifier of an IOAM-Namespace. The Namespace-ID value of 0x0000 is defined as the "Default-Namespace-ID" (see Section 4.3) and **MUST** be known to all the nodes implementing IOAM. For any other Namespace-ID value that does not match any Namespace-ID the node is configured to operate on, the node **MUST NOT** change the contents of the IOAM-Data-Fields.

IOAM-E2E-Type:
: 16-bit identifier that specifies which data types are used in the E2E Option data. The IOAM-E2E-Type value is a bit field. The order of packing the E2E Option data field elements follows the bit order of the IOAM E2E-Type field as follows:

  Bit 0
  : Most significant bit. When set, it indicates the presence of a 64-bit sequence number added to a specific "packet group" that is used to detect packet loss, packet reordering, or packet duplication within the group. The "packet group" is deployment dependent and defined at the IOAM encapsulating node, e.g., by n-tuple-based classification of packets. When this bit is set, "Bit 1" (for a 32-bit sequence number, see below) **MUST** be zero.

  Bit 1
  : When set, it indicates the presence of a 32-bit sequence number added to a specific "packet group" that is used to detect packet loss, packet reordering, or packet duplication within that group. The "packet group" is deployment dependent and defined at the IOAM encapsulating node, e.g., by n-tuple-based classification of packets. When this bit is set, "Bit 0" (for a 64-bit sequence number, see above) **MUST** be zero.

  Bit 2
  : When set, it indicates the presence of timestamp seconds, representing the time at which the packet entered the IOAM-Domain. Within the IOAM encapsulating node, the time that the timestamp is retrieved can depend on the implementation. Some possibilities are 1) the time at which the packet was received by the node, 2) the time at which the packet was transmitted by the node, or 3) when a tunnel encapsulation is used, the point at which the packet is encapsulated into the tunnel. Each implementation has to document when the E2E timestamp that is going to be put in

the packet is retrieved. This 4-octet field has three possible formats, based on either PTP (see e.g., [RFC8877]), NTP [RFC5905], or POSIX [POSIX]. The three timestamp formats are specified in Section 5. In all three cases, the timestamp seconds field contains the 32 most significant bits of the timestamp format that is specified in Section 5. If a node is not capable of populating this field, it assigns the value 0xFFFFFFFF. Note that this is a legitimate value that is valid for 1 second in approximately 136 years; the analyzer has to correlate several packets or compare the timestamp value to its own time of day in order to detect the error indication.

Bit 3 When set, it indicates the presence of timestamp fraction, representing the time at which the packet entered the IOAM-Domain. This 4-octet field has three possible formats, based on either PTP (see e.g., [RFC8877]), NTP [RFC5905], or POSIX [POSIX]. The three timestamp formats are specified in Section 5. In all three cases, the timestamp fraction field contains the 32 least significant bits of the timestamp format that is specified in Section 5. If a node is not capable of populating this field, it assigns the value 0xFFFFFFFF. Note that this is a legitimate value in the NTP format, valid for approximately 233 picoseconds in every second. If the NTP format is used, the analyzer has to correlate several packets in order to detect the error indication.

Bit 4-15 Undefined. An IOAM encapsulating node **MUST** set the value of these bits to zero upon transmission and ignore them upon receipt.

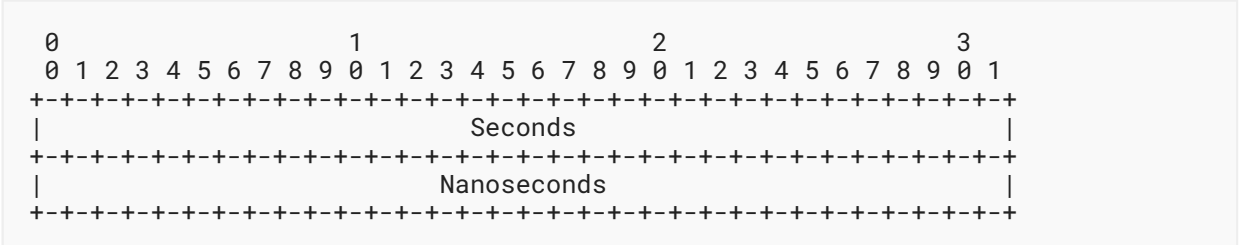
E2E Option data:
Variable-length field. The type of which is determined by the IOAM E2E-Type.

5. Timestamp Formats

The IOAM-Data-Fields include a timestamp field that is represented in one of three possible timestamp formats. It is assumed that the management plane is responsible for determining which timestamp format is used.

5.1. PTP Truncated Timestamp Format

The Precision Time Protocol (PTP) uses an 80-bit timestamp format. The truncated timestamp format is a 64-bit field, which is the 64 least significant bits of the 80-bit PTP timestamp. The PTP truncated format is specified in Section 4.3 of [RFC8877], and the details are presented below for the sake of completeness.



Timestamp field format:

Seconds: Specifies the integer portion of the number of seconds since the PTP epoch

Size: 32 bits

Units: seconds

Nanoseconds: Specifies the fractional portion of the number of seconds since the PTP epoch

Size: 32 bits

Units: nanoseconds. The value of this field is in the range 0 to (10^9)-1.

Epoch:

PTP epoch. For details, see e.g., [RFC8877].

Resolution:

The resolution is 1 nanosecond.

Wraparound:

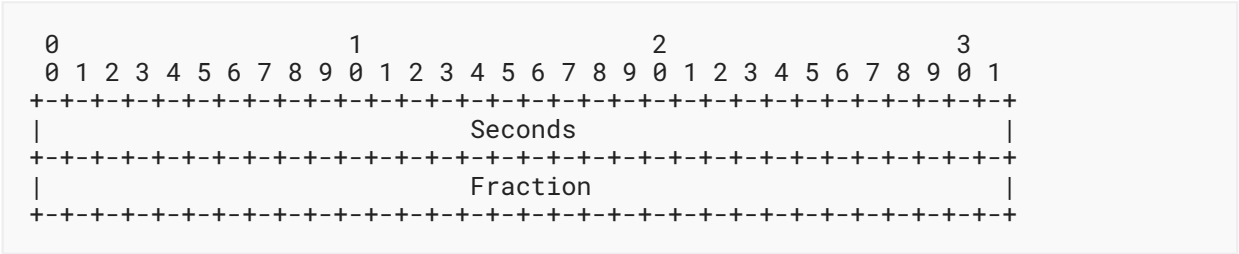
This time format wraps around every 2^{32} seconds, which is roughly 136 years. The next wraparound will occur in the year 2106.

Synchronization Aspects:

It is assumed that the nodes that run this protocol are synchronized among themselves. Nodes **MAY** be synchronized to a global reference time. Note that if PTP is used for synchronization, the timestamp **MAY** be derived from the PTP-synchronized clock, allowing the timestamp to be measured with respect to the clock of a PTP Grandmaster clock.

5.2. NTP 64-Bit Timestamp Format

The Network Time Protocol (NTP) [RFC5905] timestamp format is 64 bits long. This specification uses the NTP timestamp format that is specified in Section 4.2.1 of [RFC8877], and the details are presented below for the sake of completeness.



Timestamp field format:

Seconds: specifies the integer portion of the number of seconds since the NTP epoch

Size: 32 bits

Units: seconds

Fraction: specifies the fractional portion of the number of seconds since the NTP epoch

Size: 32 bits

Units: the unit is $2^{(-32)}$ seconds, which is roughly equal to 233 picoseconds.

Epoch:
NTP epoch. For details, see [RFC5905].

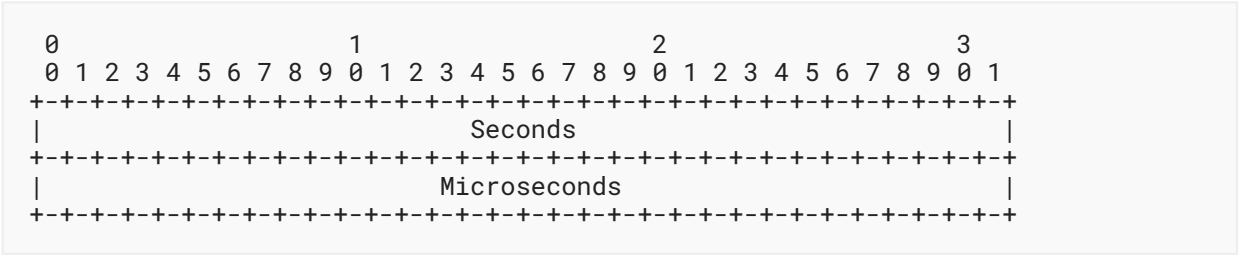
Resolution:
The resolution is $2^{(-32)}$ seconds.

Wraparound:
This time format wraps around every 2^{32} seconds, which is roughly 136 years. The next wraparound will occur in the year 2036.

Synchronization Aspects:
Nodes that use this timestamp format will typically be synchronized to UTC using NTP [RFC5905]. Thus, the timestamp **MAY** be derived from the NTP-synchronized clock, allowing the timestamp to be measured with respect to the clock of an NTP server.

5.3. POSIX-Based Timestamp Format

This timestamp format is based on the POSIX time format [POSIX]. The detailed specification of the timestamp format used in this document is presented below.



Timestamp field format:
Seconds: specifies the integer portion of the number of seconds since the POSIX epoch

Size: 32 bits

Units: seconds

Microseconds: specifies the fractional portion of the number of seconds since the POSIX epoch

Size: 32 bits

Units: the unit is microseconds. The value of this field is in the range 0 to (10^6)-1.

Epoch:
: POSIX epoch. For details, see [POSIX], Appendix A.4.16.

Resolution:
: The resolution is 1 microsecond.

Wraparound:
: This time format wraps around every 2^{32} seconds, which is roughly 136 years. The next wraparound will occur in the year 2106.

Synchronization Aspects:
: It is assumed that nodes that use this timestamp format run the Linux operating system and hence use the POSIX time. In some cases, nodes **MAY** be synchronized to UTC using a synchronization mechanism that is outside the scope of this document, such as NTP [RFC5905]. Thus, the timestamp **MAY** be derived from the NTP-synchronized clock, allowing the timestamp to be measured with respect to the clock of an NTP server.

6. IOAM Data Export

IOAM nodes collect information for packets traversing a domain that supports IOAM. IOAM decapsulating nodes, as well as IOAM transit nodes, can choose to retrieve IOAM information from the packet, process the information further, and export the information using e.g., IP Flow Information Export (IPFIX). The mechanisms and associated data formats for exporting IOAM data are outside the scope of this document.

A way to perform raw data export of IOAM data using IPFIX is discussed in [IPPM-IOAM-RAWEXPORT].

7. IANA Considerations

IANA has defined a registry group named "In Situ OAM (IOAM)".

This group includes the following registries:

- IOAM Option-Type
- IOAM Trace-Type
- IOAM Trace-Flags
- IOAM POT-Type
- IOAM POT-Flags
- IOAM E2E-Type
- IOAM Namespace-ID

The subsequent subsections detail the registries therein contained.

7.1. IOAM Option-Type Registry

This registry defines 128 code points for the IOAM Option-Type field for identifying IOAM-Option-Types, as explained in Section 4. The following code points are defined in this document:

0: IOAM Pre-allocated Trace Option-Type

1: IOAM Incremental Trace Option-Type

2: IOAM POT Option-Type

3: IOAM E2E Option-Type

Code points 4-127 are available for assignment via the "IETF Review" process, as per [RFC8126].

New registration requests **MUST** use the following template:

Name: name of the newly registered Option-Type

Code point: desired value of the requested code point

Description: brief description of the newly registered Option-Type

Reference: reference to the document that defines the new Option-Type

The evaluation of a new registration request **MUST** also include checking whether the new IOAM-Option-Type includes an IOAM-Namespace field and that the IOAM-Namespace field is the first field in the newly defined header of the new Option-Type.

7.2. IOAM Trace-Type Registry

This registry defines code points for each bit in the 24-bit IOAM Trace-Type field for the Pre-allocated Trace Option-Type and Incremental Trace Option-Type defined in Section 4.4. Bits 0-11 are defined in this document in Paragraph 5 of Section 4.4.1:

Bit 0: hop_Lim and node_id in short format

Bit 1: ingress_if_id and egress_if_id in short format

Bit 2: timestamp seconds

Bit 3: timestamp fraction

Bit 4: transit delay

Bit 5: namespace-specific data in short format

Bit 6: queue depth

Bit 7: checksum complement

Bit 8: hop_Lim and node_id in wide format

Bit 9: ingress_if_id and egress_if_id in wide format

Bit 10: namespace-specific data in wide format

Bit 11: buffer occupancy

Bit 22: variable-length Opaque State Snapshot

Bit 23: reserved

Bits 12-21 are available for assignment via the "IETF Review" process, as per [RFC8126].

New registration requests **MUST** use the following template:

Bit: desired bit to be allocated in the 24-bit IOAM Trace Option-Type field for the Pre-allocated Trace Option-Type and Incremental Trace Option-Type

Description: brief description of the newly registered bit

Reference: reference to the document that defines the new bit

7.3. IOAM Trace-Flags Registry

This registry defines code points for each bit in the 4-bit flags for the Pre-allocated Trace-Option and Incremental Trace-Option defined in Section 4.4. The meaning of Bit 0 (the most significant bit) for trace flags is defined in this document in Paragraph 3 of Section 4.4.1:

Bit 0: "Overflow" (O-bit)

Bits 1-3 are available for assignment via the "IETF Review" process, as per [RFC8126].

New registration requests **MUST** use the following template:

Bit: desired bit to be allocated in the 8-bit flags field of the Pre-allocated Trace Option-Type and Incremental Trace Option-Type

Description: brief description of the newly registered bit

Reference: reference to the document that defines the new bit

7.4. IOAM POT-Type Registry

This registry defines 256 code points to define the IOAM POT-Type for the IOAM Proof of Transit Option (Section 4.5). The code point value 0 is defined in this document:

0: 16-Octet POT data

Code points 1-255 are available for assignment via the "IETF Review" process, as per [RFC8126].

New registration requests **MUST** use the following template:

Name: name of the newly registered POT-Type

Code point: desired value of the requested code point

Description: brief description of the newly registered POT-Type

Reference: reference to the document that defines the new POT-Type

7.5. IOAM POT-Flags Registry

This registry defines code points for each bit in the 8-bit flags for the IOAM POT Option-Type defined in Section 4.5.

Bits 0-7 are available for assignment via the "IETF Review" process, as per [RFC8126].

New registration requests **MUST** use the following template:

Bit: desired bit to be allocated in the 8-bit flags field of the IOAM POT Option-Type

Description: brief description of the newly registered bit

Reference: reference to the document that defines the new bit

7.6. IOAM E2E-Type Registry

This registry defines code points for each bit in the 16-bit IOAM E2E-Type field for the IOAM E2E Option (Section 4.6). Bits 0-3 are defined in this document:

Bit 0: 64-bit sequence number

Bit 1: 32-bit sequence number

Bit 2: timestamp seconds

Bit 3: timestamp fraction

Bits 4-15 are available for assignment via the "IETF Review" process, as per [RFC8126].

New registration requests **MUST** use the following template:

Bit: desired bit to be allocated in the 16-bit IOAM E2E-Type field

Description: brief description of the newly registered bit

Reference: reference to the document that defines the new bit

7.7. IOAM Namespace-ID Registry

IANA has set up the "IOAM Namespace-ID" registry that contains 16-bit values and follows the template for requests shown below. The meaning of 0x0000 is defined in this document. IANA has reserved the values 0x0001 to 0x7FFF for private use (managed by operators), as specified in Section 4.3 of this document. Registry entries for the values 0x8000 to 0xFFFF are to be assigned via the "Expert Review" policy, as per [RFC8126].

Upon receiving a new allocation request, a designated expert will perform the following:

- Review whether the request is complete, i.e., the registration template has been filled in. The expert will send incomplete requests back to the requester.
- Check whether the request is neither a duplicate of nor conflicting with either an already existing allocation or a pending allocation. In case of duplicates or conflicts, the expert will ask the requester to update the allocation request accordingly.
- Solicit feedback from relevant working groups and communities to ensure that the new allocation request has been properly reviewed and that rough consensus on the request exists. At a minimum, the expert will solicit feedback from the IPPM Working Group by posting the request to the ippm@ietf.org mailing list. The expert will allow for a 3-week review period on the mailing lists. If the feedback received from the relevant working groups and communities within the review period indicates rough consensus on the request, the expert will approve the request and ask IANA to allocate the new Namespace-ID. In case the expert senses a lack of consensus from the feedback received, the expert will ask the requester to engage with the corresponding working groups and communities to further review and refine the request.

It is intended that any allocation will be accompanied by a published RFC. In order to allow for the allocation of code points prior to the RFC being approved for publication, the designated expert can approve allocations once it seems clear that an RFC will be published.

0x0000: default namespace (known to all IOAM nodes)

0x0001 - 0x7FFF: reserved for private use

0x8000 - 0xFFFF: unassigned

New registration requests **MUST** use the following template:

Name: name of the newly registered Namespace-ID

Code point: desired value of the requested Namespace-ID

Description: brief description of the newly registered Namespace-ID

Reference: reference to the document that defines the new Namespace-ID

Status of the registration: Status can be either "permanent" or "provisional". Namespace-ID registrations following a successful expert review will have the status "provisional". Once the RFC that defines the new Namespace-ID is published, the status is changed to "permanent".

8. Management and Deployment Considerations

This document defines the structure and use of IOAM-Data-Fields. This document does not define the encapsulation of IOAM-Data-Fields into different protocols. Management and deployment aspects for IOAM have to be considered within the context of the protocol IOAM-Data-Fields are encapsulated into and, as such, are out of scope for this document. For a discussion of IOAM deployment, please also refer to [IPPM-IOAM-DEPLOYMENT], which outlines a framework for IOAM deployment and provides best current practices.

9. Security Considerations

As discussed in [RFC7276], a successful attack on an OAM protocol in general, and specifically on IOAM, can prevent the detection of failures or anomalies or create a false illusion of nonexistent ones. In particular, these threats are applicable by compromising the integrity of IOAM data, either by maliciously modifying IOAM options in transit or by injecting packets with maliciously generated IOAM options. All nodes in the path of an IOAM-carrying packet can perform such an attack.

The Proof of Transit Option-Type (see Section 4.5) is used for verifying the path of data packets, i.e., proving that packets transited through a defined set of nodes.

In case an attacker gains access to several nodes in a network and would be able to change the system software of these nodes, IOAM-Data-Fields could be misused and repurposed for a use different from what is specified in this document. One type of misuse is the implementation of a covert channel between network nodes.

From a confidentiality perspective, although IOAM options are not expected to contain user data, they can be used for network reconnaissance, allowing attackers to collect information about network paths, performance, queue states, buffer occupancy, etc. Moreover, if IOAM data leaks from the IOAM-Domain, it could enable reconnaissance beyond the scope of the IOAM-Domain. One possible application of such reconnaissance is to gauge the effectiveness of an ongoing attack, e.g., if buffers and queues are overflowing.

IOAM can be used as a means for implementing Denial-of-Service (DoS) attacks or for amplifying them. For example, a malicious attacker can add an IOAM header to packets in order to consume the resources of network devices that take part in IOAM or entities that receive, collect, or analyze the IOAM data. Another example is a packet length attack in which an attacker pushes headers associated with IOAM-Option-Types into data packets, causing these packets to be increased beyond the MTU size, resulting in fragmentation or in packet drops. In case POT is used, an attacker could corrupt the POT data fields in the packet, resulting in a verification failure of the POT data, even if the packet followed the correct path.

Since IOAM options can include timestamps, if network devices use synchronization protocols, then any attack on the time protocol [RFC7384] can compromise the integrity of the timestamp-related data fields.

At the management plane, attacks can be set up by misconfiguring or by maliciously configuring IOAM-enabled nodes in a way that enables other attacks. IOAM configuration should only be managed by authorized processes or users.

IETF protocols require features to ensure their security. While IOAM-Data-Fields don't represent a protocol by themselves, the IOAM-Data-Fields add to the protocol that the IOAM-Data-Fields are encapsulated into. Any specification that defines how IOAM-Data-Fields carried in an encapsulating protocol **MUST** provide for a mechanism for cryptographic integrity protection of the IOAM-Data-Fields. Cryptographic integrity protection could be achieved through a mechanism of the encapsulating protocol, or it could incorporate the mechanisms specified in [IPPM-IOAM-DATA-INTEGRITY].

The current document does not define a specific IOAM encapsulation. It has to be noted that some IOAM encapsulation types can introduce specific security considerations. A specification that defines an IOAM encapsulation is expected to address the respective encapsulation-specific security considerations.

Notably, IOAM is expected to be deployed in limited domains, thus confining the potential attack vectors to within the limited domain. A limited administrative domain provides the operator with the means to select, monitor, and control the access of all the network devices, making these devices trusted by the operator. Indeed, in order to limit the scope of threats mentioned above to within the current limited domain, the network operator is expected to enforce policies that prevent IOAM traffic from leaking outside of the IOAM-Domain and prevent IOAM data from outside the domain to be processed and used within the domain.

This document does not define the data contents of custom fields, like "Opaque State Snapshot" and "namespace-specific data" IOAM-Data-Fields. These custom data fields will have security considerations corresponding to their defined data contents that need to be described where those formats are defined.

IOAM deployments that leverage both IOAM Trace Option-Types, i.e., the Pre-allocated Trace Option-Type and Incremental Trace Option-Type, can suffer from incomplete visibility if the information gathered via the two Trace Option-Types is not correlated and aggregated appropriately. If IOAM transit nodes leverage the IOAM-Data-Fields in the packet for further actions or insights, then IOAM transit nodes that only support one IOAM Trace Option-Type in an IOAM deployment that leverages both Trace Option-Types have limited visibility and thus can draw inappropriate conclusions or take wrong actions.

The security considerations of a system that deploys IOAM, much like any system, has to be reviewed on a per-deployment-scenario basis based on a systems-specific threat analysis, which can lead to specific security solutions that are beyond the scope of the current document.

Specifically, in an IOAM deployment that is not confined to a single LAN but spans multiple inter-connected sites (for example, using an overlay network), the inter-site links can be secured (e.g., by IPsec) in order to avoid external threats.

IOAM deployment considerations, including approaches to mitigate the above discussed threads and potential attacks, are outside the scope of this document. IOAM deployment considerations are discussed in [IPPM-IOAM-DEPLOYMENT].

10. References

10.1. Normative References

[POSIX] IEEE, "IEEE/Open Group 1003.1-2017 - IEEE Standard for Information Technology--Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7", IEEE Std 1003.1-2017, January 2018, <https://standards.ieee.org/ieee/1003.1/7101/>.

[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>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[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>.

[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>.

10.2. Informative References

[IPPM-IOAM-DATA-INTEGRITY] Brockners, F., Bhandari, S., Mizrahi, T., and J. Iurman, "Integrity of In-situ OAM Data Fields", Work in Progress, Internet-Draft, draft-ietf-ippm-ioam-data-integrity-01, 2 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ioam-data-integrity-01>.

[IPPM-IOAM-DEPLOYMENT] Brockners, F., Bhandari, S., Bernier, D., and T. Mizrahi, "In-situ OAM Deployment", Work in Progress, Internet-Draft, draft-ietf-ippm-ioam-deployment-01, 11 April 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ioam-deployment-01>.

[IPPM-IOAM-RAWEXPORT] Spiegel, M., Brockners, F., Bhandari, S., and R. Sivakolundu, "In-situ OAM raw data export with IPFIX", Work in Progress, Internet-Draft, draft-spiegel-ippm-ioam-rawexport-06, 21 February 2022, <https://datatracker.ietf.org/doc/html/draft-spiegel-ippm-ioam-rawexport-06>.

[IPV6-RECORD-ROUTE] Kitamura, H., "Record Route for IPv6 (RR6) Hop-by-Hop Option Extension", Work in Progress, Internet-Draft, draft-kitamura-ipv6-record-route-00, 17 November 2000, <https://datatracker.ietf.org/doc/html/draft-kitamura-ipv6-record-route-00>.

[NVO3-VXLAN-GPE] Maino, F., Ed., Kreeger, L., Ed., and U. Elzur, Ed., "Generic Protocol Extension for VXLAN (VXLAN-GPE)", Work in Progress, Internet-Draft, draft-ietf-nvo3-vxlan-gpe-12, 22 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-nvo3-vxlan-gpe-12>.

[RFC7276] Mizrahi, T., Sprecher, N., Bellagamba, E., and Y. Weingarten, "An Overview of Operations, Administration, and Maintenance (OAM) Tools", RFC 7276, DOI 10.17487/RFC7276, June 2014, <https://www.rfc-editor.org/info/rfc7276>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

[RFC7799] Morton, A., "Active and Passive Metrics and Methods (with Hybrid Types In-Between)", RFC 7799, DOI 10.17487/RFC7799, May 2016, <https://www.rfc-editor.org/info/rfc7799>.

[RFC7820] Mizrahi, T., "UDP Checksum Complement in the One-Way Active Measurement Protocol (OWAMP) and Two-Way Active Measurement Protocol (TWAMP)", RFC 7820, DOI 10.17487/RFC7820, March 2016, <https://www.rfc-editor.org/info/rfc7820>.

[RFC7821] Mizrahi, T., "UDP Checksum Complement in the Network Time Protocol (NTP)", RFC 7821, DOI 10.17487/RFC7821, March 2016, <https://www.rfc-editor.org/info/rfc7821>.

[RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <https://www.rfc-editor.org/info/rfc8300>.

[RFC8799] Carpenter, B. and B. Liu, "Limited Domains and Internet Protocols", RFC 8799, DOI 10.17487/RFC8799, July 2020, <https://www.rfc-editor.org/info/rfc8799>.

[RFC8877] Mizrahi, T., Fabini, J., and A. Morton, "Guidelines for Defining Packet Timestamps", RFC 8877, DOI 10.17487/RFC8877, September 2020, <https://www.rfc-editor.org/info/rfc8877>.

[RFC8926] Gross, J., Ed., Ganga, I., Ed., and T. Sridhar, Ed., "Geneve: Generic Network Virtualization Encapsulation", RFC 8926, DOI 10.17487/RFC8926, November 2020, <https://www.rfc-editor.org/info/rfc8926>.

Acknowledgements

The authors would like to thank Éric Vyncke, Nalini Elkins, Srihari Raghavan, Ranganathan T S, Karthik Babu Harichandra Babu, Akshaya Nadahalli, LJ Wobker, Erik Nordmark, Vengada Prasad Govindan, Andrew Yourtchenko, Aviv Kfir, Tianran Zhou, Zhenbin (Robin), and Greg Mirsky for the comments and advice.

This document leverages and builds on top of several concepts described in [IPV6-RECORD-ROUTE]. The authors would like to acknowledge the work done by the author Hiroshi Kitamura and people involved in writing it.

The authors would like to gracefully acknowledge useful review and insightful comments received from Joe Clarke, Al Morton, Tom Herbert, Carlos J. Bernardos, Haoyu Song, Mickey Spiegel, Roman Danyliw, Benjamin Kaduk, Murray S. Kucherawy, Ian Swett, Martin Duke, Francesca Palombini, Lars Eggert, Alvaro Retana, Erik Kline, Robert Wilton, Zaheduzzaman Sarker, Dan Romascanu, and Barak Gafni.

Contributors

This document was the collective effort of several authors. The text and content were contributed by the editors and the coauthors listed below.

Carlos Pignataro
Cisco Systems, Inc.
Research Triangle Park
7200-11 Kit Creek Road
NC 27709
United States of America
Email: cpignata@cisco.com

Mickey Spiegel
Barefoot Networks, an Intel company
101 Innovation Drive
San Jose, CA 95134-1941
United States of America
Email: mickey.spiegel@intel.com

Barak Gafni
Nvidia
Suite 100
350 Oakmead Parkway
Sunnyvale, CA 94085
United States of America
Email: gbarak@nvidia.com

Jennifer Lemon
Broadcom
270 Innovation Drive
San Jose, CA 95134
United States of America
Email: jennifer.lemon@broadcom.com

Hannes Gredler
RtBrick Inc.
Email: hannes@rtbrick.com

John Leddy
United States of America
Email: john@leddy.net

Stephen Youell
JP Morgan Chase
25 Bank Street
London
E14 5JP
United Kingdom
Email: stephen.youell@jpmorgan.com

David Mozes
Email: mosesster@gmail.com

Petr Lapukhov
Facebook
1 Hacker Way
Menlo Park, CA 94025
United States of America
Email: petr@fb.com

Remy Chang
Barefoot Networks, an Intel company
101 Innovation Drive
San Jose, CA 95134-1941
United States of America
Email: remy.chang@intel.com

Daniel Bernier
Bell Canada
Canada
Email: daniel.bernier@bell.ca

Authors' Addresses

Frank Brockners (EDITOR)
Cisco Systems, Inc.
3rd Floor
Nordhein-Westfalen
Hansaallee 249
40549 Duesseldorf
Germany
Email: fbrockne@cisco.com

Shwetha Bhandari (EDITOR)
Thoughtspot
3rd Floor
Indiqube Orion
Garden Layout
HSR Layout
24th Main Rd
Bangalore 560 102
Karnataka
India
Email: shwetha.bhandari@thoughtspot.com

Tal Mizrahi (EDITOR)
Huawei
8-2 Matam
Haifa 3190501
Israel
Email: tal.mizrahi.phd@gmail.com