                        Data Fields for In-situ OAM
                        draft-ietf-ippm-ioam-data-11

Abstract

   In-situ Operations, Administration, and Maintenance (IOAM) records
   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 in-situ OAM.
   In-situ OAM data fields can be encapsulated into a variety of
   protocols such as NSH, Segment Routing, Geneve, IPv6 (via extension
   header), or IPv4. In-situ OAM can be used to complement OAM
   mechanisms based on e.g. ICMP or other types of probe packets.

Status of This Memo

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

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

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

   This Internet-Draft will expire on May 26, 2021.

Copyright Notice

Table of Contents

1. Introduction

This document defines data fields for "in-situ" Operations,
Administration, and Maintenance (IOAM). In-situ OAM 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 to complement
mechanisms such as Ping or Traceroute. In terms of "active" or
"passive" OAM, "in-situ" OAM 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 1. 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 pre-defined path, SLA verification for
the live 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.

IOAM use cases and mechanisms have expanded as this document matured,
resulting in additional flags and options that could trigger creation
of additional packets dedicated to OAM. The term IOAM continues to
be used for such mechanisms, in addition to the "in-situ" mechanisms
that motivated this terminology.

2. Contributors

This document was the collective effort of several authors. The text
and content were contributed by the editors and the co-authors listed
below. The contact information of the co-authors appears at the end
of this document.

o Carlos Pignataro

o Mickey Spiegel

o Barak Gafni

o Jennifer Lemon

o Hannes Gredler

o John Leddy

o Stephen Youell

o David Mozes

o Petr Lapukhov

o Remy Chang

o Daniel Bernier

3. Conventions

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

Abbreviations used in this document:

E2E   Edge to Edge

Geneve: Generic Network Virtualization Encapsulation
           [I-D.ietf-nvo3-geneve]

IOAM: In-situ Operations, Administration, and Maintenance

MTU: Maximum Transmit Unit

NSH: Network Service Header [RFC8300]

OAM: Operations, Administration, and Maintenance

PMTU   Path MTU

POT: Proof of Transit

SFC: Service Function Chain

SID: Segment Identifier

SR: Segment Routing

      VXLAN-GPE: Virtual eXtensible Local Area Network, Generic Protocol
                 Extension [I-D.ietf-nvo3-vxlan-gpe]

[4](#). **Scope, Applicability, and Assumptions**

      IOAM deployment assumes a set of constraints, requirements, and
      guiding principles which are described in this section.

      Scope: This document defines the data fields and associated data
      types for in-situ OAM. The in-situ OAM data field can be
      encapsulated in a variety of protocols, including NSH, Segment
      Routing, Geneve, IPv6, or IPv4. Specification details for these
      different protocols are outside the scope of this document.

      Deployment domain (or scope) of in-situ OAM deployment: IOAM is a
      network domain focused feature, with "network domain" being a set of
      network devices or entities within a single administration. For
      example, a network domain can include an enterprise campus using
      physical connections between devices or an overlay network using
      virtual connections / tunnels for connectivity between said devices.
      A network domain is defined by its perimeter or edge. 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 has to 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), path MTU (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 live user traffic by the devices which form the edge of a domain.
      Devices which 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 live 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 node(s) 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. The specification of how IOAM-Data-Fields
   are encapsulated into "parent" protocols, like e.g., NSH or IPv6 is
   outside the scope of this document.

   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 take the
   specifics of devices with hardware data planes and software data
   planes into account.

[5](#). IOAM Data-Fields, Types, 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.

[5.1](#). 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 8.1](#) for details. Corresponding to these IOAM-
   Option-Types, different IOAM-Data-Fields are defined. IOAM-Data-
   Fields can be encapsulated into a variety of protocols, such as NSH,
   Geneve, IPv6, etc. The definition of how IOAM-Data-Fields are
   encapsulated into other protocols is outside the scope of this
   document.

   This document defines four IOAM-Option-Types:

   o Pre-allocated Trace Option-Type

   o Incremental Trace Option-Type

   o Proof of Transit (POT) Option-Type

   o Edge-to-Edge (E2E) Option-Type

5.2. IOAM-Domains and types of IOAM Nodes

   IOAM is expected to be deployed in a specific domain. The part of
   the network which employs IOAM is referred to as the "IOAM-Domain".
   One or more IOAM-Option-Types are added to a packet upon entering the
   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,
   decapsulating) is defined within an IOAM-Namespace (see below). A
   node can have different roles in different IOAM-Namespaces.

   A device which adds at least one IOAM-Option-Type to the packet is
   called the "IOAM encapsulating node", whereas a device which removes
   an IOAM-Option-Type is referred to as the "IOAM decapsulating node".
   Nodes within the domain which are aware of IOAM data and read and/or
   write or process the IOAM data are called "IOAM transit nodes". IOAM
   nodes which 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 which 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 8.1) 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" updates 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 populates IOAM trace data in
   either Pre-allocated or Incremental Trace Option-Type but not both.
   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 which 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, that is added in a future revision. An IOAM
decapsulating node situated at the edge of an IOAM domain MUST remove
all IOAM-Option-Types and associated encapsulation headers for all
IOAM-Namespaces from the packet.

IOAM-Namespaces allow for a namespace-specific definition and
interpretation of IOAM-Data-Fields. An interface-id 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 5.3](#) for details on IOAM-Namespaces.

[5.3](#). IOAM-Namespaces

A subset or all of the IOAM-Option-Types and their corresponding
IOAM-Data-Fields can be associated to an IOAM-Namespace. IOAM-
Namespaces add further context to IOAM-Option-Types and associated
IOAM-Data-Fields. Any IOAM-Namespace MUST interpret the IOAM-Option-
Types and associated IOAM-Data-Fields per the definition in this
document. IOAM-Namespaces group nodes to support different
deployment approaches of IOAM (see a few example use-cases below) as
well as resolve issues which can occur due to IOAM-Data-Fields not
being globally unique (e.g. IOAM node identifiers do not have to be
globally unique). IOAM-Data-Fields significance is always within a
particular IOAM-Namespace.

An IOAM-Namespace is identified by a 16-bit namespace identifier
(Namespace-ID). IOAM-Namespace identifiers MUST be present and
populated in all IOAM-Option-Types. The Namespace-ID value is
divided into two sub-ranges:

o An operator-assigned range from 0x0001 to 0x7FFF

o 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 which
do not leverage specific namespaces for some or all of their packets
that carry IOAM data fields.

Namespace identifiers allow devices which are IOAM capable to
determine:

o whether IOAM-Option-Type(s) 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.

o 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.

o whether IOAM-Option-Type(s) has to be removed from the packet,
   e.g. at a domain edge or domain boundary.

IOAM-Namespaces support several different uses:

o IOAM-Namespaces can be used by an operator to distinguish
   different operational domains. Devices at domain edges can filter
   on Namespace-IDs to provide for proper IOAM-Domain isolation.

o IOAM-Namespaces provide additional context for IOAM-Data-Fields
   and thus ensure that IOAM-Data-Fields are unique and can be
   interpreted properly by management stations or network
   controllers. While, for example, the node identifier field
   (node_id, see below) does not need to be unique in a deployment
   (e.g. 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 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) which

do not have a unit associated are to be interpreted within the
context of a IOAM-Namespace.

o IOAM-Namespaces can be used to identify different sets of devices
   (e.g., different types of devices) in a deployment: If an operator
   desires 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.

   * Assigning different IOAM Namespace-IDs to different sets of
      nodes or network partitions and using the Namespace-ID as a
      selector at the IOAM encapsulating node, a full trace for a
      flow could be collected and constructed via partial traces in
      different packets of the same flow. Example: An operator could
      choose to group the devices of a domain into two IOAM-
      Namespaces, in a way that on average, only every second hop
      would be recorded by any device. To retrieve a full view of
      the deployment, the captured IOAM-Data-Fields of the two IOAM-
      Namespaces need to be correlated.

   * 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. 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 a 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.

5.4. IOAM Trace Option-Types

   "IOAM tracing data" is expected to be collected at every IOAM transit
   node that a packet traverses to ensure visibility into the entire
   path a packet takes within an IOAM-Domain. I.e., in a typical
   deployment all nodes in an IOAM-Domain would participate in IOAM and
   thus be IOAM transit nodes, IOAM encapsulating 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) will only be collected on those nodes which support
IOAM functionality as defined in this document. Nodes which 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 path MTU 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 which are IOAM capable exceeds the available
space in the packet.

To optimize hardware and software implementations, IOAM tracing is
defined as two separate options. Any deployment MAY 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 Pre-allocated Trace Option-
   Type in the packet and sets corresponding fields in this IOAM-
   Option-Type. The IOAM encapsulating node allocates an array which
   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 which 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 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
   arbitrarily long packets as the MTU allows. The IOAM
   encapsulating node allocates space for the Incremental Trace
   Option-Type. Based on 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, decrease the remaining length
available to subsequent nodes and adjust the lengths and possibly
checksums in outer headers.

A particular implementation of IOAM MAY choose to support only one of
the two trace option types. In the event that both options are
utilized at the same time, the Incremental Trace-Option MUST be
placed before the Pre-allocated Trace-Option. Deployments which 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, 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-Type(s) 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:

o 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.

o Identification of the interface that a packet was received on,
   i.e. ingress interface.

o Identification of the interface that a packet was sent out on,
   i.e. egress interface.

o 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 in-situ OAM domain follow the same definition.

o Generic data: Format-free information where syntax and semantic 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 geo-location 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.

   o Information to detect whether IOAM trace data was added at every
      hop or whether certain hops in the domain weren't IOAM transit
      nodes.

5.4.1. **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, 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.

Pre-allocated and incremental trace option headers:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Namespace-ID          |NodeLen  | Flags | RemainingLen|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                IOAM-Trace-Type                | Reserved     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


The trace option data MUST be 4-octet aligned:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
|                                                              |  |
|                      node data list [0]                      |  |
|                                                              |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  D
|                                                              |  a
|                      node data list [1]                      |  t
|                                                              |  a
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                          ...                                 ~  S
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  p
|                                                              |  a
|                     node data list [n-1]                     |  c
|                                                              |  e
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                                                              |  |
|                      node data list [n]                      |  |
|                                                              |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
```


Namespace-ID: 16-bit identifier of an IOAM-Namespace. The
   Namespace-ID value of 0x0000 is defined as the "Default-Namespace-
   ID" (see Section 5.3) 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 wide, 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, or it can ignore the NodeLen value and calculate the node length from the IOAM-Trace-Type bits (see below).

Flags 4-bit field. Flags are allocated by IANA, as specified in Section 8.3. This document allocates a single flag as follows:

Bit 0 "Overflow" (O-bit) (most significant bit). If there are not enough octets left to record 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. Given that the sender knows the path MTU (PMTU), the sender MAY set the initial value of RemainingLen according to the number of node data bytes allowed before exceeding the MTU. 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 - sizeof(opaque snapshot) in 4 octet units. If RemainingLen in a pre-allocated trace option exceeds the length of the option, as specified in the preceding header, then the node MUST NOT add any fields.

IOAM-Trace-Type: A 24-bit identifier which 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 the Section 5.4.2. 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 presence of Hop_Lim and node_id (short format) in the node data.

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

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

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

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

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

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

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

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

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

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

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

Bit 12-21 Undefined. An IOAM encapsulating node MUST set the value of each of these bits 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 presence of variable length Opaque
State Snapshot field.

Bit 23   Reserved: MUST be set to zero upon transmission and
ignored upon receipt.

Section 5.4.2 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 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.

5.4.2. IOAM node data fields and associated formats

All the IOAM-Data-Fields MUST be 4-octet aligned. If a node which 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" as
well as "wide format". Their use is not 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.

5.4.2.1. Hop_Lim and node_id short format

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

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Hop_Lim     |               node_id                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Hop_Lim: 1-octet unsigned integer. It is set to the Hop Limit value
   in the packet at 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, and 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
   TTL/Hop limit equivalent field.

node_id: 3-octet unsigned integer. 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.

5.4.2.2. ingress_if_id and egress_if_id

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

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     ingress_if_id          |          egress_if_id            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

egress_if_id: 2-octet unsigned integer. 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.

5.4.2.3. timestamp seconds

The "timestamp seconds" field is a 4-octet unsigned integer field.
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 PTP [IEEE1588v2], NTP [RFC5905], or POSIX
[POSIX]. The three timestamp formats are specified in Section 6. In
all three cases, the Timestamp Seconds field contains the 32 most
significant bits of the timestamp format that is specified in
Section 6. 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.

5.4.2.4. timestamp subseconds

The "timestamp subseconds" field is a 4-octet unsigned integer field.
Absolute timestamp in subseconds that specifies the time at which the
packet was received by the node. This field has three possible
formats; based on either PTP [IEEE1588v2], NTP [RFC5905], or POSIX
[POSIX]. The three timestamp formats are specified in Section 6. In
all three cases, the Timestamp Subseconds field contains the 32 least
significant bits of the timestamp format that is specified in
Section 6. 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.

5.4.2.5. 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.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|O|                     transit delay                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.2.6. namespace specific data

   The "namespace specific data" field is a 4-octet field which 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.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    namespace specific data                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.2.7. 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).

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          queue depth                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.2.8. Checksum Complement

   The "Checksum Complement" field is a 4-octet node data which contains
   a 4-octet Checksum Complement field. 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.

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].

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Checksum Complement                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.2.9. Hop_Lim and node_id wide

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

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Hop_Lim     |                 node_id                      ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                        node_id (contd)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Hop_Lim: 1-octet unsigned integer. It is set to the Hop Limit value
   in the packet at 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 for e.g.
   TTL value in IPv4 header or hop limit field from IPv6 header of
   the packet. The semantics of the Hop_Lim field depend on the
   lower layer protocol that IOAM is encapsulated into, and 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 TTL/Hop limit equivalent field.

node_id: 7-octet unsigned integer. 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.

5.4.2.10. 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:

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       ingress_if_id                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       egress_if_id                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

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

5.4.2.11. namespace specific data wide

   The "namespace specific data wide" field is an 8-octet field which
   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.

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                   namespace specific data                    ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                namespace specific data (contd)               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.2.12. 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-id 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 RECOMMEND the implementations to use standard units such as
   Bytes.

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       buffer occupancy                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.2.13. **Opaque State Snapshot**

   The "Opaque State Snapshot" 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 pre-defined 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.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Length      |                  Schema ID                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                                                               |
   |                       Opaque data                             |
   ~                                                               ~
   .                                                               .
   .                                                               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   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.

5.4.3. **Examples of IOAM node data**

   An entry in the "node data list" array can have different formats,
   following the needs of the deployment. Some deployments might only
   be interested in recording the node identifiers, whereas others might
   be interested in recording node identifier and timestamp. The
   section provides example entries of the "node data list".

0xD40000: IOAM-Trace-Type is 0xD40000 (0b1101010000000000000000000)
   then the format of node data is:

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Hop_Lim     |                 node_id                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      ingress_if_id        |            egress_if_id           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     timestamp subseconds                     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     namespace specific data                  |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

0xC00000: IOAM-Trace-Type is 0xC00000 (0b1100000000000000000000000)
   then the format is:

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Hop_Lim     |                 node_id                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      ingress_if_id        |            egress_if_id           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

0x900000: IOAM-Trace-Type is 0x900000 (0b1001000000000000000000000)
   then the format is:

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Hop_Lim     |                 node_id                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     timestamp subseconds                     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

0x840000: IOAM-Trace-Type is 0x840000 (0b1000010000000000000000000)
   then the format is:

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Hop_Lim     |                 node_id                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     namespace specific data                  |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+
       | Hop_Lim | node_id |
       +-+
       | timestamp subseconds |
       +-+
       | namespace specific data |
       +-+

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

        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+
       | timestamp seconds |
       +-+
       | timestamp subseconds |
       +-+
       | Hop_Lim | node_id |
       +-+
       | node_id(contd) |
       +-+
       | Length | Schema Id |
       +-+
       | |
       | |
       | Opaque data |
       ~   ~
       .   .
       .   .
       +-+

5.5. IOAM Proof of Transit Option-Type

   IOAM Proof of Transit Option-Type is to support path or service
   function chain [RFC7665] verification use cases. Proof-of-transit
   leverages mechanisms like Shamir's Secret Sharing Schema (SSSS)
   [SSS]. For further information on Proof-of-transit, please refer to
   [I-D.ietf-sfc-proof-of-transit]. While details on how the IOAM data
   for the Proof-of-transit option is processed at IOAM encapsulating,
   decapsulating and transit nodes are outside the scope of the
   document, all of these 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:

o Random: Unique identifier for the packet (e.g., 64-bits allow for
   the unique identification of 2^64 packets).

o Cumulative: Information which 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:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Namespace-ID          |IOAM POT Type  | IOAM POT flags|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IOAM proof of transit Option-Type IOAM-Data-Fields MUST be
4-octet aligned:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       POT Option data field determined by IOAM-POT-Type      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Namespace-ID: 16-bit identifier of an IOAM-Namespace. The
   Namespace-ID value of 0x0000 is defined as the "Default-Namespace-
   ID" (see Section 5.3) 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
   POT Type 0:

   0: POT data is a 16 Octet field as described below.

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

IOAM POT flags: 8-bit. Following flags are defined:

   Bit 0 "Profile-to-use" (P-bit) (most significant bit). For IOAM
      POT types that use a maximum of two profiles to drive
      computation, indicates which POT-profile is used. The two
      profiles are numbered 0, 1.

   Bit 1-7 Reserved: MUST be set to zero upon transmission and
      ignored upon receipt.

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

5.5.1. IOAM Proof of Transit Type 0

IOAM proof of transit option of IOAM POT Type 0:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Namespace-ID           |IOAM POT Type=0|P|R R R R R R R|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
|                          Random                               |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  P
|                       Random(contd)                          |  O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  T
|                        Cumulative                            |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                     Cumulative (contd)                       |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
```

Namespace-ID: 16-bit identifier of an IOAM-Namespace. The
   Namespace-ID value of 0x0000 is defined as the "Default-Namespace-
   ID" (see Section 5.3) 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 section defines the
   POT data when the IOAM POT Type is set to the value 0.

P bit: 1-bit. "Profile-to-use" (P-bit) (most significant bit).
   Indicates which POT-profile is used to generate the Cumulative.
   Any node participating in POT will have a maximum of 2 profiles
   configured that drive the computation of cumulative. The two

      profiles are numbered 0, 1. This bit conveys whether profile 0 or
      profile 1 is used to compute the Cumulative.

   R (7 bits): 7-bit IOAM POT flags for future use. MUST be set to
      zero upon transmission and ignored upon receipt.

   Random: 64-bit Per packet Random number.

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

   Note: Larger or smaller sizes of "Random" 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".

5.6. IOAM Edge-to-Edge Option-Type

   The IOAM Edge-to-Edge Option-Type is to carry data that is added by
   the IOAM encapsulating node and interpreted by 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:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Namespace-ID           |          IOAM-E2E-Type        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   IOAM Edge-to-Edge Option-Type IOAM-Data-Fields MUST
   be 4-octet aligned:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       E2E Option data field determined by IOAM-E2E-Type       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Namespace-ID: 16-bit identifier of an IOAM-Namespace. The
   Namespace-ID value of 0x0000 is defined as the "Default-Namespace-
   ID" (see Section 5.3) 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, then
   the node MUST NOT change the contents of the IOAM-Data-Fields.

IOAM-E2E-Type: A 16-bit identifier which 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 indicates presence of a
            64-bit sequence number added to a specific "packet group"
            which 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.

   Bit 1   When set indicates presence of a 32-bit sequence number
            added to a specific "packet group" which 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.

   Bit 2   When set indicates 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, 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 [IEEE1588v2], NTP [RFC5905], or POSIX [POSIX]. The
            three timestamp formats are specified in Section 6. In
            all three cases, the Timestamp Seconds field contains the
            32 most significant bits of the timestamp format that is
            specified in Section 6. 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 indicates presence of timestamp subseconds,
                    representing the time at which the packet entered the
                    IOAM domain. This 4-octet field has three possible
                    formats; based on either PTP [IEEE1588v2], NTP [RFC5905],
                    or POSIX [POSIX]. The three timestamp formats are
                    specified in Section 6. In all three cases, the
                    Timestamp Subseconds field contains the 32 least
                    significant bits of the timestamp format that is
                    specified in Section 6. 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 upon
                    receipt.

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

6. Timestamp Formats

   The IOAM-Data-Fields include a timestamp field which 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.

6.1. PTP Truncated Timestamp Format

   The Precision Time Protocol (PTP) [IEEE1588v2] 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.

```
 0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                            Seconds                            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          Nanoseconds                          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
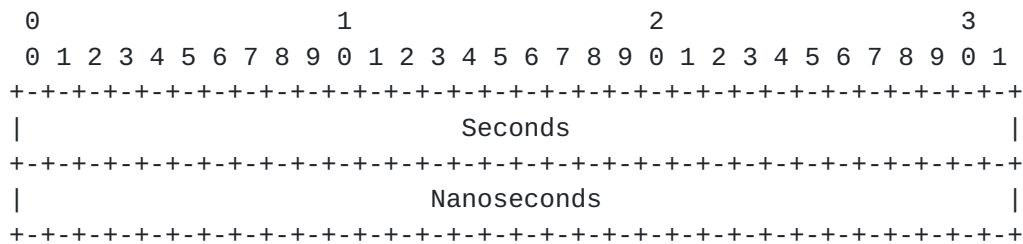
Figure 1: PTP [IEEE1588v2] Truncated Timestamp Format

Timestamp field format:

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

   + Size: 32 bits.

   + Units: seconds.

   Nanoseconds: specifies the fractional portion of the number of
   seconds since the epoch.

   + Size: 32 bits.

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

Epoch:

   The PTP [IEEE1588v2] epoch is 1 January 1970 00:00:00 TAI, which
   is 31 December 1969 23:59:51.999918 UTC.

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 nodes that run this protocol are synchronized
   among themselves. Nodes MAY be synchronized to a global reference
   time. Note that if PTP [IEEE1588v2] 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
   an PTP Grandmaster clock.

   The PTP truncated timestamp format is not affected by leap
   seconds.

6.2. NTP 64-bit Timestamp Format

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


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Seconds                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Fraction                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

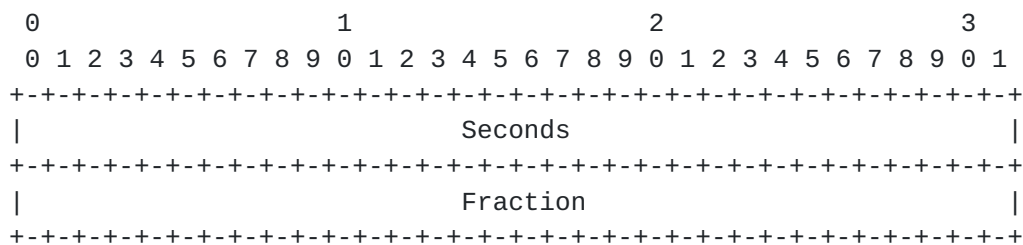   Figure 2: NTP [RFC5905] 64-bit Timestamp Format

   Timestamp field format:

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

      + Size: 32 bits.

      + Units: seconds.

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

      + Size: 32 bits.

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

   Epoch:

      The epoch is 1 January 1900 at 00:00 UTC.

   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.

      The NTP timestamp format is affected by leap seconds; it
      represents the number of seconds since the epoch minus the number
      of leap seconds that have occurred since the epoch. The value of
      a timestamp during or slightly after a leap second could be
      temporarily inaccurate.

6.3. 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.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                            Seconds                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                          Microseconds                        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
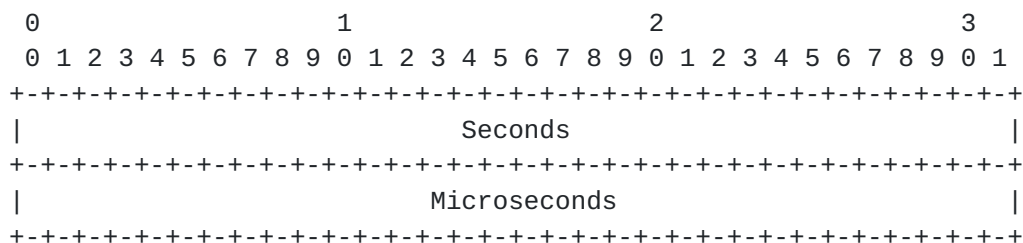
                 Figure 3: POSIX-based Timestamp Format

   Timestamp field format:

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

      + Size: 32 bits.

      + Units: seconds.

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

      + Size: 32 bits.

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

   Epoch:

      The epoch is 1 January 1970 00:00:00 TAI, which is 31 December
      1969 23:59:51.999918 UTC.

   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.

      The POSIX-based timestamp format is affected by leap seconds; it
      represents the number of seconds since the epoch minus the number
      of leap seconds that have occurred since the epoch. The value of
      a timestamp during or slightly after a leap second could be
      temporarily inaccurate.

7. 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., IPFIX. The mechanisms and associated data formats for
   exporting IOAM data is outside the scope of this document.

   Raw data export of IOAM data using IPFIX is discussed in
   [I-D.spiegel-ippm-ioam-rawexport].

8. IANA Considerations

This document requests the following IANA Actions.

IANA is requested to define a registry group named "In-Situ OAM (IOAM) Protocol Parameters".

This group will include the following registries:

    IOAM Option-Type

    IOAM Trace-Type

    IOAM Trace-Flags

    IOAM POT-Type

    IOAM POT-Flags

    IOAM E2E-Type

    IOAM Namespace-ID

New registries in this group can be created via RFC Required process as per [RFC8126].

The subsequent sub-sections detail the registries herein contained.

8.1. 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 5. The following code points are defined in this draft:

0   IOAM Pre-allocated Trace Option-Type

1   IOAM Incremental Trace Option-Type

2   IOAM POT Option-Type

3   IOAM E2E Option-Type

4 - 127 are available for assignment via RFC Required process as per [RFC8126].

8.2. **IOAM Trace-Type Registry**

This registry defines code point for each bit in the 24-bit IOAM-
Trace-Type field for Pre-allocated trace option and Incremental trace
option defined in Section 5.4. The meaning of Bits 0 - 11 for trace
type are defined in this document in Paragraph 5 of Section 5.4.1:

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 subseconds

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

The meaning for Bits 12 - 21 are available for assignment via RFC
Required process as per [RFC8126].

8.3. **IOAM Trace-Flags Registry**

This registry defines code points for each bit in the 4 bit flags for
the Pre-allocated trace option and for the Incremental trace option
defined in Section 5.4. The meaning of Bit 0 (the most significant
bit) for trace flags is defined in this document in Paragraph 3 of
Section 5.4.1:

Bit 0 "Overflow" (O-bit)

Bit 1 - 3 are available for assignment via RFC Required process as per [RFC8126].

8.4. IOAM POT-Type Registry

This registry defines 256 code points to define IOAM POT Type for IOAM proof of transit option Section 5.5. The code point value 0 is defined in this document:

0: 16 Octet POT data

1 - 255 are available for assignment via RFC Required process as per [RFC8126].

8.5. IOAM POT-Flags Registry

This registry defines code points for each bit in the 8 bit flags for IOAM POT option defined in Section 5.5. The meaning of Bit 0 for IOAM POT flags is defined in this document in Section 5.5:

Bit 0 "Profile-to-use" (P-bit)

The meaning for Bits 1 - 7 are available for assignment via RFC Required process as per [RFC8126].

8.6. IOAM E2E-Type Registry

This registry defines code points for each bit in the 16 bit IOAM-E2E-Type field for IOAM E2E option Section 5.6. The meaning of Bit 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 subseconds

The meaning of Bits 4 - 15 are available for assignment via RFC Required process as per [RFC8126].

8.7. IOAM Namespace-ID Registry

IANA is requested to set up an "IOAM Namespace-ID Registry", containing 16-bit values. The meaning of Bit 0 is defined in this document. IANA is requested to reserve the values 0x0001 to 0x7FFF for private use (managed by operators), as specified in Section 5.3

of the current document. Registry entries for the values 0x8000 to
0xFFFF are to be assigned via the "Expert Review" policy defined in
[RFC8126]. Upon a new allocation request, the responsible AD will
appoint a designated expert, who will review the allocation request.
The expert will post the request on the IPPM mailing list, and
possibly on other relevant mailing lists, to allow for community
feedback. Based on the review, the expert will either approve or
deny the request. The intention is that any allocation will be
accompanied by a published RFC. But in order to allow for the
allocation of values prior to the RFC being approved for publication,
the designated expert can approve allocations once it seems clear
that an RFC will be published.

0: default namespace (known to all IOAM nodes)

0x0001 - 0x7FFF: reserved for private use

0x8000 - 0xFFFF: unassigned

9. 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
[I-D.brockners-opsawg-ioam-deployment], which outlines a framework
for IOAM deployment and provides best current practices.

10. 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

The Proof of Transit Option-Type (Section Section 5.5) is used for
verifying the path of data packets. The security considerations of
POT are further discussed in [I-D.ietf-sfc-proof-of-transit].

From a confidentiality perspective, although IOAM options do not
contain user data, they can be used for network reconnaissance,
allowing attackers to collect information about network paths,
performance, queue states, buffer occupancy and other information.

Moreover, if IOAM data leaks from the IOAM domain it could enable
reconnaissance beyond the scope of the IOAM domain. Note that in
case IOAM is used in "Direct Exporting" mode
[I-D.ioamteam-ippm-ioam-direct-export], the IOAM related trace
information would not be available in the customer data packets, but
would trigger export of packet related IOAM information at every
node, thus restricting the potential threat to the management plane
and mitigating the leakage threat. IOAM data exporting and the way
it is secured is outside the scope of this document.

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.

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. Thus, IOAM configuration has to be secured in a way
that authenticates authorized users and verifies the integrity of
configuration procedures.

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, in most cases IOAM is expected to be deployed in specific
network domains, thus confining the potential attack vectors to
within the network 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 network 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.

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.

11. Acknowledgements

The authors would like to thank Eric 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 and Zhenbin (Robin) for the
comments and advice.

This document leverages and builds on top of several concepts
described in [I-D.kitamura-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,
Haoyu Song, Mickey Spiegel and Barak Gafni.

12. References

12.1. Normative References

[IEEE1588v2]
           Institute of Electrical and Electronics Engineers, "IEEE
           Std 1588-2008 - IEEE Standard for a Precision Clock
           Synchronization Protocol for Networked Measurement and
           Control Systems", IEEE Std 1588-2008, 2008,
           <http://standards.ieee.org/findstds/
           standard/1588-2008.html>.

[POSIX]   Institute of Electrical and Electronics Engineers, "IEEE
           Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004) - IEEE
           Standard for Information Technology - Portable Operating
           System Interface (POSIX(R))", IEEE Std 1003.1-2008, 2008,
           <https://standards.ieee.org/findstds/
           standard/1003.1-2008.html>.

   [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>.

12.2. Informative References

   [I-D.brockners-opsawg-ioam-deployment]
              Brockners, F., Bhandari, S., and d.
              daniel.bernier@bell.ca, "In-situ OAM Deployment", draft-
              brockners-opsawg-ioam-deployment-02 (work in progress),
              September 2020.

   [I-D.ietf-nvo3-geneve]
              Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic
              Network Virtualization Encapsulation", draft-ietf-
              nvo3-geneve-16 (work in progress), March 2020.

   [I-D.ietf-nvo3-vxlan-gpe]
              Maino, F., Kreeger, L., and U. Elzur, "Generic Protocol
              Extension for VXLAN (VXLAN-GPE)", draft-ietf-nvo3-vxlan-
              gpe-10 (work in progress), July 2020.

   [I-D.ietf-sfc-proof-of-transit]
              Brockners, F., Bhandari, S., Mizrahi, T., Dara, S., and S.
              Youell, "Proof of Transit", draft-ietf-sfc-proof-of-
              transit-08 (work in progress), November 2020.

   [I-D.ioamteam-ippm-ioam-direct-export]
              Song, H., Gafni, B., Zhou, T., Li, Z., Brockners, F.,
              Bhandari, S., Sivakolundu, R., and T. Mizrahi, "In-situ
              OAM Direct Exporting", draft-ioamteam-ippm-ioam-direct-
              export-00 (work in progress), October 2019.

   [I-D.kitamura-ipv6-record-route]
              Kitamura, H., "Record Route for IPv6 (PR6) Hop-by-Hop
              Option Extension", draft-kitamura-ipv6-record-route-00
              (work in progress), November 2000.

   [I-D.spiegel-ippm-ioam-rawexport]
              Spiegel, M., Brockners, F., Bhandari, S., and R.
              Sivakolundu, "In-situ OAM raw data export with IPFIX",
              draft-spiegel-ippm-ioam-rawexport-04 (work in progress),
              November 2020.

   [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>.

   [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>.

   [SSS]   Wikipedia, "Shamir's Secret Sharing",
              <https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing>.

Contributors' Addresses

        Carlos Pignataro
        Cisco Systems, Inc.
        7200-11 Kit Creek Road
        Research Triangle Park, NC 27709
        United States

        Email: cpignata@cisco.com

        Mickey Spiegel
        Barefoot Networks, an Intel company
        4750 Patrick Henry Drive
        Santa Clara, CA 95054
        US

        Email: mickey.spiegel@intel.com

        Barak Gafni
        Mellanox Technologies, Inc.
        350 Oakmead Parkway, Suite 100
        Sunnyvale, CA 94085
        U.S.A.

        Email: gbarak@mellanox.com

        Jennifer Lemon
        Broadcom
        270 Innovation Drive
        San Jose, CA 95134
        US

        Email: jennifer.lemon@broadcom.com

        Hannes Gredler
        RtBrick Inc.

        Email: hannes@rtbrick.com

        John Leddy
        United States

        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
        US

        Email: petr@fb.com

        Remy Chang
        Barefoot Networks
        4750 Patrick Henry Drive
        Santa Clara, CA 95054
        US

        Email: remy@barefootnetworks.com

        Daniel Bernier
        Bell Canada
        Canada

        Email: daniel.bernier@bell.ca

Authors' Addresses

    Frank Brockners (editor)
    Cisco Systems, Inc.
    Hansaallee 249, 3rd Floor
    DUESSELDORF, NORDRHEIN-WESTFALEN 40549
    Germany

    Email: fbrockne@cisco.com

Shwetha Bhandari (editor)
Cisco Systems, Inc.
Cessna Business Park, Sarjapura Marathalli Outer Ring Road
Bangalore, KARNATAKA 560 087
India

Email: shwethab@cisco.com

Tal Mizrahi (editor)
Huawei
8-2 Matam
Haifa   3190501
Israel

Email: tal.mizrahi.phd@gmail.com