ippm
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2020

F. Brockners
S. Bhandari
C. Pignataro
Cisco
H. Gredler
RtBrick Inc.
J. Leddy

S. Youell
JPMC
T. Mizrahi
Huawei Network.IO Innovation Lab
D. Mozes

P. Lapukhov
Facebook
R. Chang
Barefoot Networks
D. Bernier
Bell Canada
J. Lemon
Broadcom

July 04, 2019

Data Fields for In-situ OAM
draft-ietf-ippm-ioam-data-06

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 embedded into a variety of transports such as NSH, Segment Routing, Geneve, native 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.

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 is being sent within packets specifically dedicated to OAM. IOAM is to complement mechanisms such as Ping or Traceroute, or more recent active probing mechanisms as described in [I-D.lapukhov-dataplane-probe]. In terms of "active" or "passive" OAM, "in-situ" OAM can be considered a hybrid OAM type. While no extra packets are sent, IOAM adds information to the packets therefore cannot be considered passive. In terms of the classification given in [RFC7799] IOAM could be portrayed as Hybrid Type 1. "In-situ" mechanisms do not require extra packets to be sent and hence don't change the packet traffic mix within the network. 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.

2. 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

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]

3. 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 transported by a variety of transport protocols, including NSH, Segment Routing, Geneve, IPv6, or IPv4. Specification details for these different transport 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 carrier protocols for IOAM must 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, e.g. using for example packet filtering methods. The operator should consider 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 a native IPv6 transport, IOAM

support for ICMPv6 Echo Request/Reply could 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 within an IOAM domain can update and/or add IOAM data-fields. Domain edge devices 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. It SHOULD be possible to enable 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.) The selected set of traffic can also be all traffic.

Encapsulation independence: Data formats for IOAM SHOULD be defined in a transport-independent manner. IOAM applies to a variety of encapsulating protocols. A definition of how IOAM data fields are carried by different transport protocols 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-records could be present at every layer. The behavior follows the ships-in-the-night model, i.e. IOAM data in one layer is independent from IOAM data 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 and decapsulation.

IOAM implementation: The IOAM data-field definitions take the specifics of devices with hardware data-plane and software data-plane into account.

4. IOAM Data Types and Formats

This section defines IOAM data types and data fields and associated data types required for IOAM.

To accommodate the different uses of IOAM, IOAM data fields fall into different categories, as specified below. In IOAM these categories are referred to as IOAM-Types. A common registry is maintained for IOAM-Types, see Section 7.2 for details. Corresponding to these IOAM-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-Types, as specified in this section:

- Pre-allocated Trace Option
- Incremental Trace Option
- Proof of Transit (POT) Option
- Edge-to-Edge (E2E) Option

IOAM is expected to be deployed in a specific domain rather than on the overall Internet. The part of the network which employs IOAM is referred to as the "IOAM-domain". IOAM data is added to a packet upon entering the IOAM-domain and is removed from the packet when exiting the domain. Within the IOAM-domain, the IOAM data may be updated by network nodes that the packet traverses. The device which adds an IOAM data container to the packet to capture IOAM data is called the "IOAM encapsulating node", whereas the device which removes the IOAM data container 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. 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-Types (from the list of four IOAM-Types above) into packets that IOAM is enabled for. If IOAM is enabled for a selected subset of the traffic, the 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 options are present in the packet, each IOAM transit node will update at most one of these options. A transit node cannot add new IOAM options to a packet, and cannot change an IOAM Edge-to-Edge Option.

An IOAM decapsulating node removes all the IOAM-Types from packets.

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.

4.1. IOAM Namespaces

A subset or all of the IOAM option types and associated IOAM data fields can be associated to an IOAM namespace. 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. 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 are defined within an IOAM namespace.

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

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

- 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 default and known to all the nodes implementing IOAM.

Namespace identifiers allow devices which are IOAM capable to determine:

- whether IOAM option header(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.

- which IOAM option headers need to be processed/updated in case there are multiple IOAM option headers present in the packet. Multiple option headers can be present in a packet in case of overlapping IOAM domains or in case of a layered IOAM deployment.

- whether IOAM option header(s) should be removed from the packet, e.g. at a domain edge or domain boundary.

IOAM namespaces support several different uses:

- 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.

- Namespaces provide additional context for IOAM data fields and thus ensure that IOAM data is unique and can be interpreted properly by management stations or network controllers. While, for example, the IOAM node identifier (Node-ID) does not need to be unique in a deployment (e.g. an operator may wish to use different Node-IDs for different IOAM layers, even within the same device; or Node-IDs 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, 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 namespace.

- 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 based on the device, the devices could be grouped into multiple 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. This 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 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 namespaces, in a way that at 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 namespaces need to be correlated.

  * Assigning different Namespace-IDs to different sets of nodes or network partitions and using a separate IOAM header for each Namespace-ID, a full trace for a flow could be collected and constructed via partial traces from each IOAM header in each of the packets in the flow. Example: An operator could choose to group the devices of a domain into two namespaces, in a way

that each namespace is represented by one of two IOAM headers in the packet. Each node would record data only for the IOAM namespace that it belongs to, ignoring the other IOAM header with a namespace to which it doesn't belong. To retrieve a full view of the deployment, the captured IOAM data fields of the two namespaces need to be correlated.

4.2. IOAM Tracing Options

"IOAM tracing data" is expected to be collected at every 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 in-situ OAM-domain would participate in IOAM and thus be IOAM transit nodes, IOAM encapsulating or IOAM decapsulating nodes. If not all nodes within a domain are IOAM capable, IOAM tracing information (i.e., node data) will only be collected on those nodes which are IOAM capable. Nodes which are not IOAM capable 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.

To optimize hardware and software implementations tracing is defined as two separate options. Any deployment MAY choose to configure and support one or both of the following options. An implementation of the transport protocol that carries these in-situ OAM data MAY choose to support only one of the options. 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. 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 enabled for a particular domain.

Pre-allocated Trace Option: This trace option is defined as a container of node data fields with pre-allocated space for each node to populate its information. This option is useful for software implementations where it is efficient to allocate the space once and index into the array to populate the data during transit. The IOAM encapsulating node allocates the option header and sets the fields in the option header. The in situ OAM 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.

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 this 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 in-situ OAM encapsulating node allocates the option header. The in-situ OAM encapsulating node based on operational state and configuration sets the fields in the header that control what node data fields should be collected, and how large the node data list can grow. The in-situ OAM 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.

Every node data entry is to hold information for a particular IOAM transit node that is traversed by a packet. The in-situ OAM decapsulating node removes the IOAM data and processes and/or exports the metadata. IOAM data uses its own name-space for information such as node identifier or interface identifier. This allows for a domain-specific definition and interpretation. For example: In one case an interface-id could 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).

The following IOAM data is defined for IOAM tracing:

- 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. 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.

- Generic data: Format-free information where syntax and semantic of the information is defined by the operator in a specific deployment. For a specific deployment, all IOAM nodes should 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.

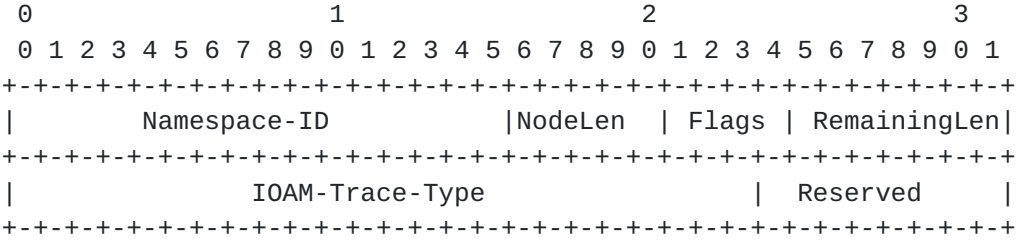
- A mechanism to detect whether IOAM trace data was added at every hop or whether certain hops in the domain weren't in-situ OAM transit nodes.

The "node data list" array 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.

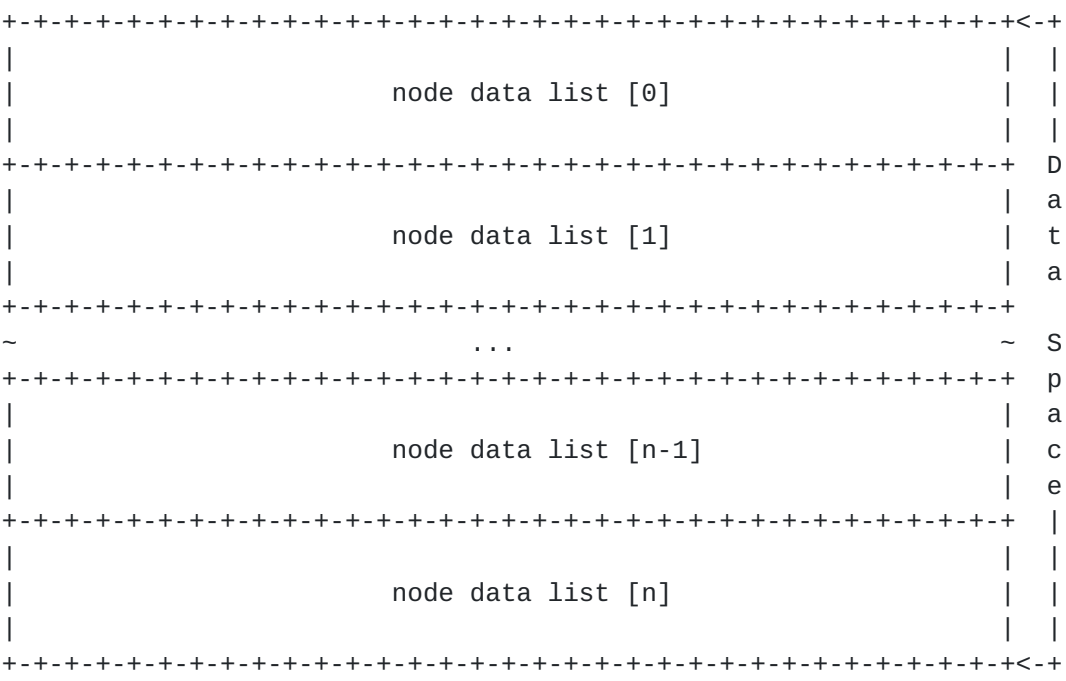
4.2.1. Pre-allocated and Incremental Trace Options

The in-situ OAM pre-allocated trace option and the in-situ OAM incremental trace option have similar formats. Except where noted below, the internal formats and fields of the two trace options are identical.

Pre-allocated and incremental trace option headers:



The trace option data MUST be 4-octet aligned:



Namespace-ID: 16-bit identifier of an IOAM namespace. The Namespace-ID value of 0x0000 is defined as the default value 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 7 is not set, then NodeLen specifies the actual length added by each node. If IOAM-Trace-Type bit 7 is

set, then the actual length added by a node would be (NodeLen + Opaque Data Length).

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 may rely on the NodeLen value, or it may ignore the NodeLen value and calculate the node length from the IOAM-Trace-Type bits.

Flags 4-bit field. Flags are allocated by IANA, as specified in Section 7.4. The current document allocates a single flag as follows:

Bit 0 "Overflow" (O-bit) (most significant bit). This bit is set by the network element if there are not enough octets left to record node data, no field is added and the overflow "O-bit" must be set to "1" in the 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. When RemainingLen reaches 0, nodes are no longer allowed to add node data. Given that the sender knows the minimum path MTU, 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, this is used as an offset in data space to record the node data element.

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 bit fields are defined in this document, with details on each field described in the Section 4.2.2. 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 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 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 variable length Opaque State Snapshot field.

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 namespace specific data in wide format in the node data.

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

Bit 12-22 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 23 When set indicates presence of the Checksum Complement node data.

Section 4.2.2 describes the IOAM data types and their formats. Within an in-situ OAM domain possible combinations of these bits making the IOAM-Trace-Type can be restricted by configuration knobs.

Reserved: 8-bits. Must be zero.

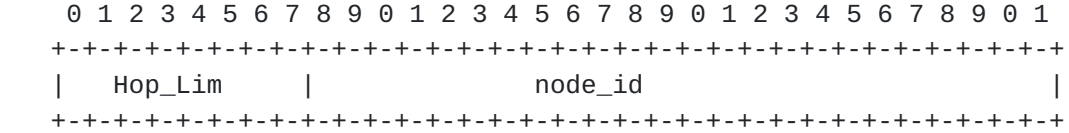
Node data List [n]: Variable-length field. The type of which is determined by the IOAM-Trace-Type bit representing the n-th node data in the node data list. The node data list is encoded starting from the last node data of the path. The first element of the node data list (node data list [0]) contains the last node of the path while the last node data of the node data list (node data list[n]) contains the first node data of the path traced. 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.

4.2.2. IOAM node data fields and associated formats

All the 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.

Data field and associated data type for each of the data field is shown below:

Hop_Lim and node_id: 4-octet field defined as follows:

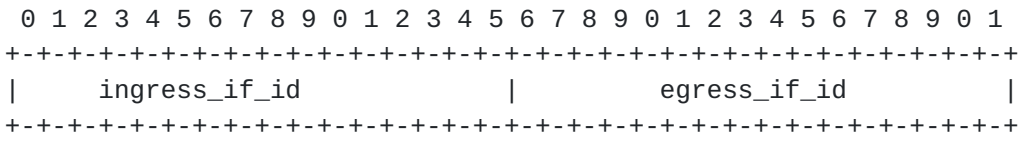


Hop_Lim: 1-octet unsigned integer. It is set to the Hop Limit value in the packet at the node that records this data. Hop Limit information is used to identify the location of the node

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 over, and therefore its specific semantics are outside the scope of this memo.

node_id: 3-octet unsigned integer. Node identifier field to uniquely identify a node within in-situ OAM domain. The procedure to allocate, manage and map the node_ids is beyond the scope of this document.

ingress_if_id and egress_if_id: 4-octet field defined as follows:



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

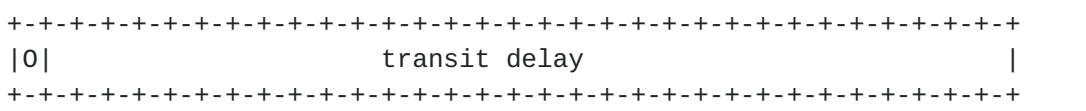
egress_if_id: 2-octet unsigned integer. Interface identifier to record the egress interface the packet is forwarded out of.

Note that due to the fact that IOAM uses its own 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. an IOAM interface ID could represent a physical interface, a virtual or logical interface, or even a queue.

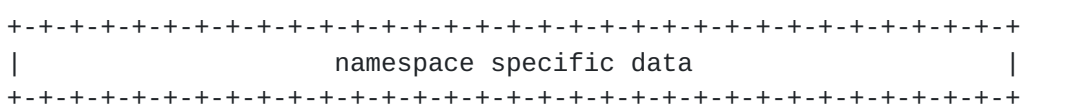
timestamp seconds: 4-octet unsigned integer. Absolute timestamp in seconds that specifies the time at which the packet was received by the node. This field has three possible formats; based on either PTP [IEEE1588v2], NTP [RFC5905], or POSIX [POSIX]. The three timestamp formats are specified in Section 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 should correlate several packets or compare the timestamp value to its own time-of-day in order to detect the error indication.

timestamp subseconds: 4-octet unsigned integer. 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 5. In all three cases, the Timestamp Subseconds 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 should correlate several packets in order to detect the error indication.

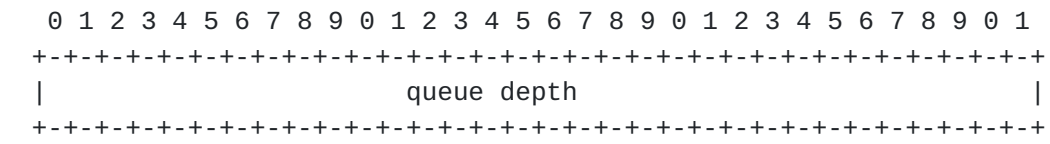
transit delay: 4-octet unsigned integer in the range 0 to 2^31-1. It is the time in nanoseconds the packet spent in the transit node. This can serve as an indication of the queuing delay at the node. If the transit delay exceeds 2^31-1 nanoseconds then the top bit 'O' is set to indicate overflow and 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.



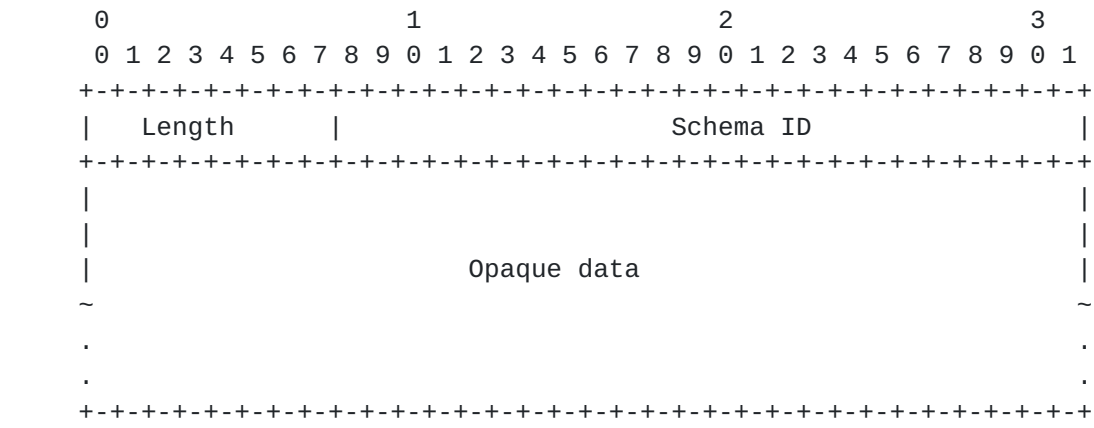
namespace specific data: 4-octet field which can be used by the node to add namespace specific data. This represents a "free-format" 4-octet bit field with its semantics defined in the context of a specific namespace.



queue depth: 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 number of memory buffers used by the queue (a packet may consume one or more memory buffers, depending on its size).



Opaque State Snapshot: Variable length field. 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 a 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 a namespace, indicates which particular schema is used, and should be configured on the network element by the operator.



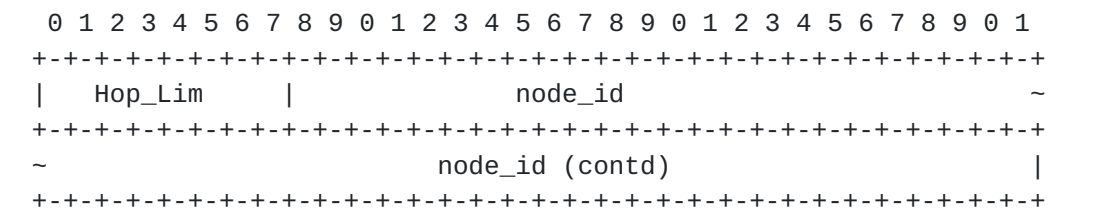
Length: 1-octet unsigned integer. It is the length in multiples of 4-octets of the Opaque data field that follows Schema Id.

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.

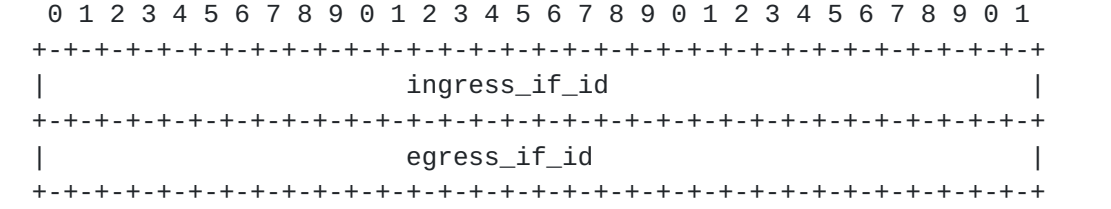
Hop_Lim and node_id wide: 8-octet field defined as follows:



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 over, and therefore its specific semantics are outside the scope of this memo.

node_id: 7-octet unsigned integer. Node identifier field to uniquely identify a node within in-situ OAM domain. The procedure to allocate, manage and map the node_ids is beyond the scope of this document.

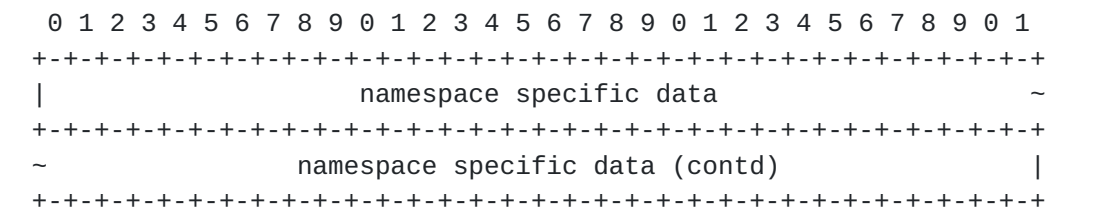
ingress_if_id and egress_if_id wide: 8-octet field defined as follows:



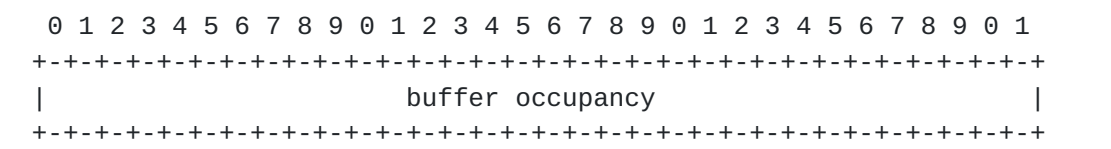
ingress_if_id: 4-octet unsigned integer. Interface identifier to record the ingress interface the packet was received on.

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

namespace specific data wide: 8-octet field which can be used by the node to add namespace specific data. This represents a "free-format" 8-octet bit field with its semantics defined in the context of a specific namespace.



buffer occupancy: 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 depend on the equipment type and deployment and has to be interpreted within the context of a namespace and/or node-id if used.



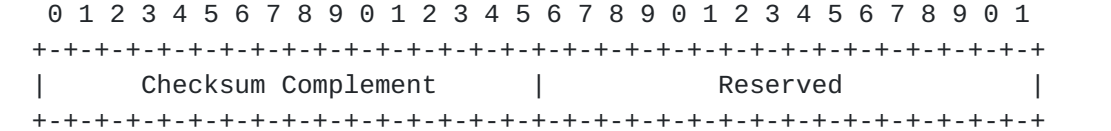
Checksum Complement: 4-octet node data which contains a two-octet Checksum Complement field, and a 2-octet reserved field. The Checksum Complement is useful when IOAM is transported over encapsulations that make use of a UDP transport, such as VXLAN-GPE or Geneve. Without the Checksum Complement, nodes adding IOAM node data must update the UDP Checksum field. 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.

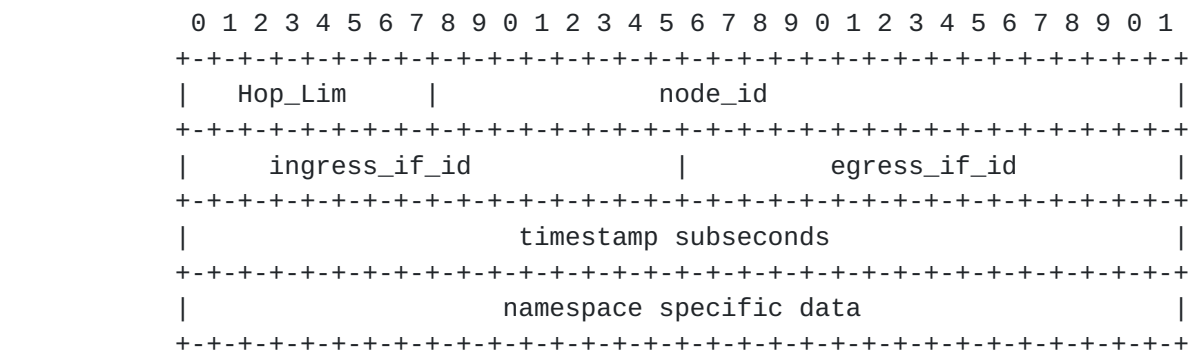
Checksum Complement fields are used in a similar manner in [RFC7820] and [RFC7821].



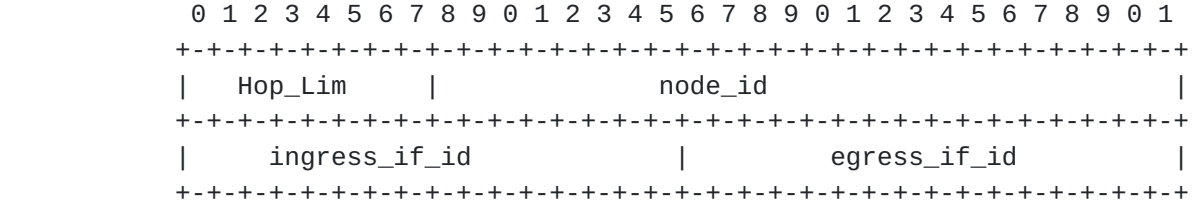
4.2.3. Examples of IOAM node data

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 defines different types that an entry in "node data list" can take.

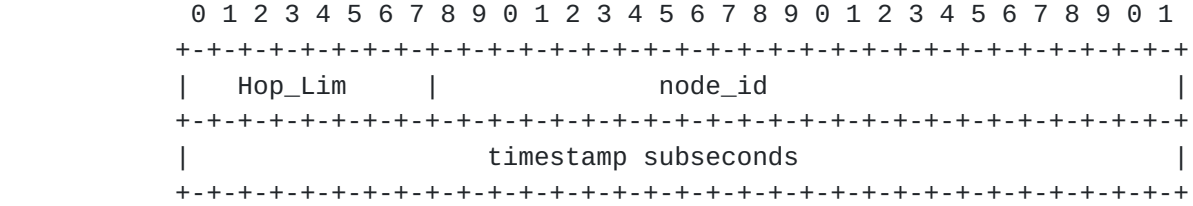
0xD40000: IOAM-Trace-Type is 0xD40000 then the format of node data is:



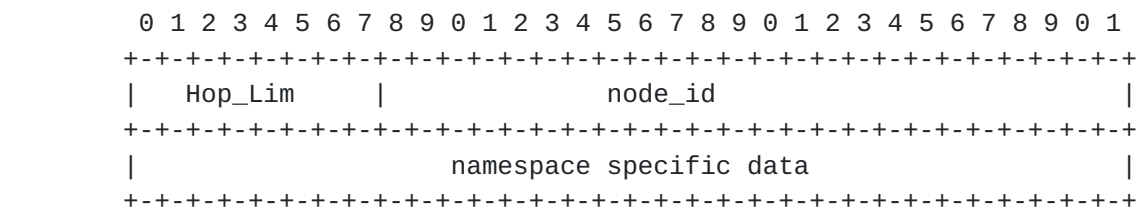
0xC00000: IOAM-Trace-Type is 0xC00000 then the format is:



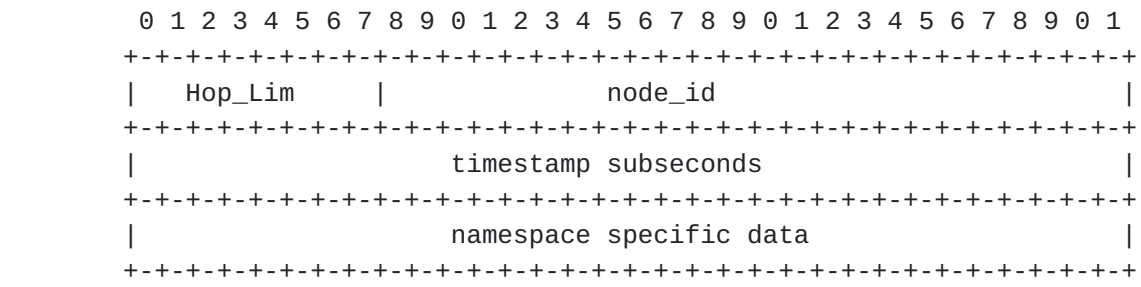
0x900000: IOAM-Trace-Type is 0x900000 then the format is:



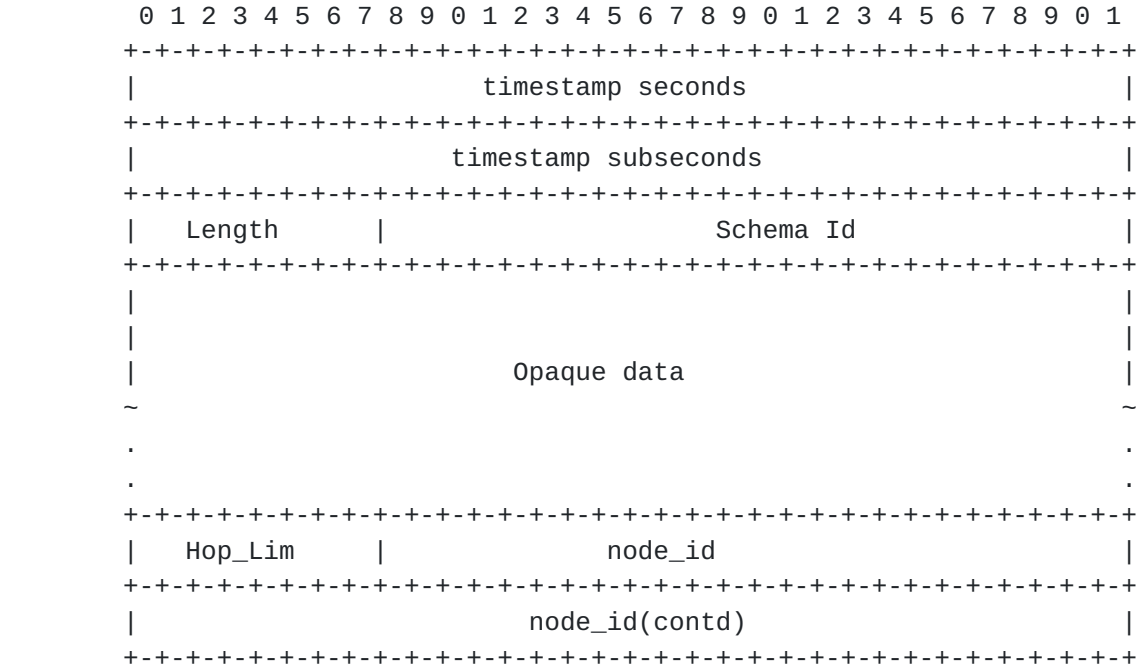
0x840000: IOAM-Trace-Type is 0x840000 then the format is:



0x940000: IOAM-Trace-Type is 0x940000 then the format is:



0x318000: IOAM-Trace-Type is 0x318000 then the format is:



4.3. IOAM Proof of Transit Option

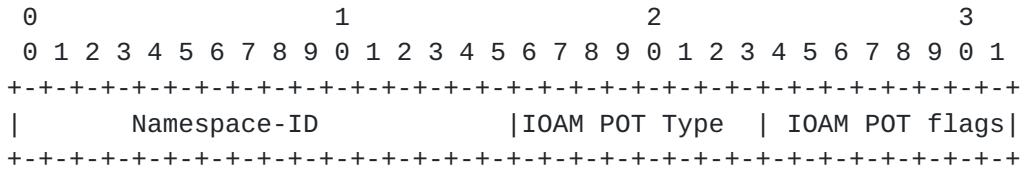
IOAM Proof of Transit data is to support the path or service function chain [RFC7665] verification use cases. Proof-of-transit uses methods like nested hashing or nested encryption of the IOAM data or

mechanisms such as Shamir's Secret Sharing Schema (SSSS). 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 to the packet:

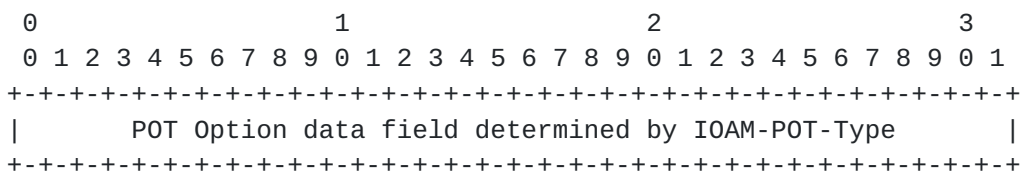
- Random: Unique identifier for the packet (e.g., 64-bits allow for the unique identification of 2^64 packets).

- Cumulative: Information which is handed from node to node and updated by every node according to a verification algorithm.

IOAM proof of transit option header:



IOAM proof of transit option data MUST be 4-octet aligned.:



Namespace-ID: 16-bit identifier of an IOAM namespace. The Namespace-ID value of 0x0000 is defined as the default value 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.

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

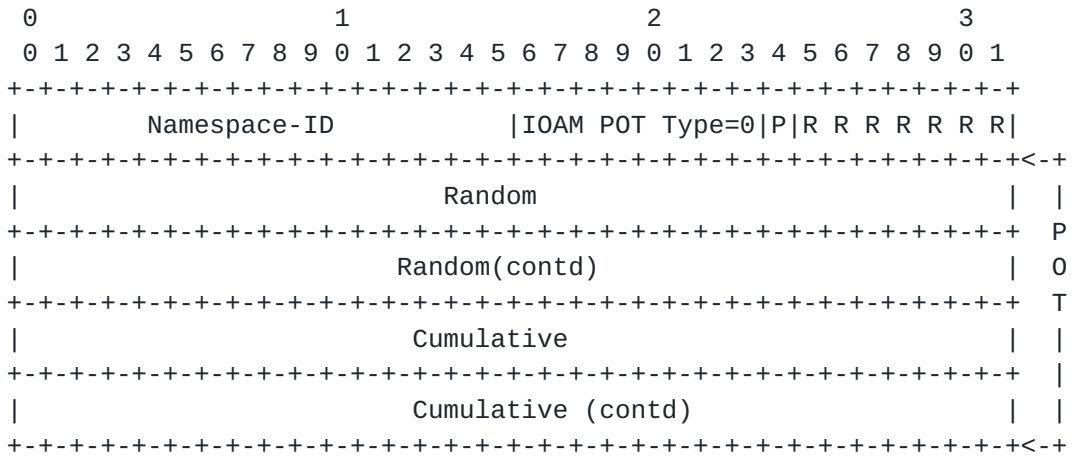
Bit 0 "Profile-to-use" (P-bit) (most significant bit). For IOAM POT types that use a maximum of two profiles to drive computation, indicates which POT-profile is used. The two profiles are numbered 0, 1.

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.

4.3.1. 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. The Namespace-ID value of 0x0000 is defined as the default value 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.

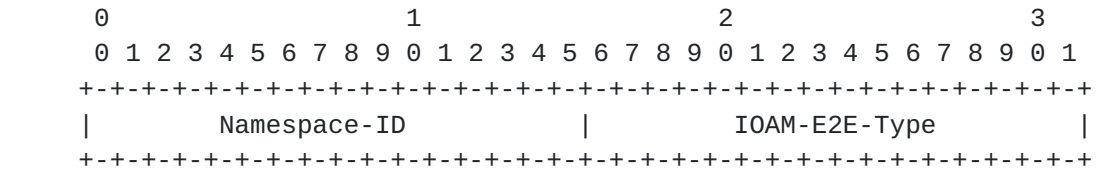
Cumulative: 64-bit Cumulative that is updated at specific nodes by processing per packet Random number field and configured parameters.

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 transport protocol used). Future versions of this document will address different sizes of data for "proof of transit".

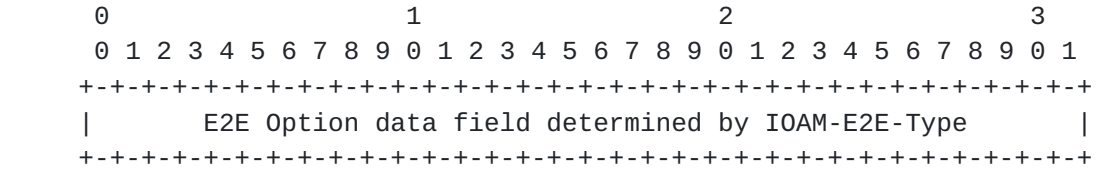
4.4. IOAM Edge-to-Edge Option

The IOAM edge-to-edge option 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 without modifying it.

IOAM edge-to-edge option header:



IOAM edge-to-edge option data MUST be 4-octet aligned:



Namespace-ID: 16-bit identifier of an IOAM namespace. The Namespace-ID value of 0x0000 is defined as the default value 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 for the transmission of the frame. 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 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 should 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 for the transmission of the frame. 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 5. In all three cases, the Timestamp Subseconds 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 should 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.

5. 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.

5.1. 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 [I-D.ietf-ntp-packet-timestamps], and the details are presented below for the sake of completeness.

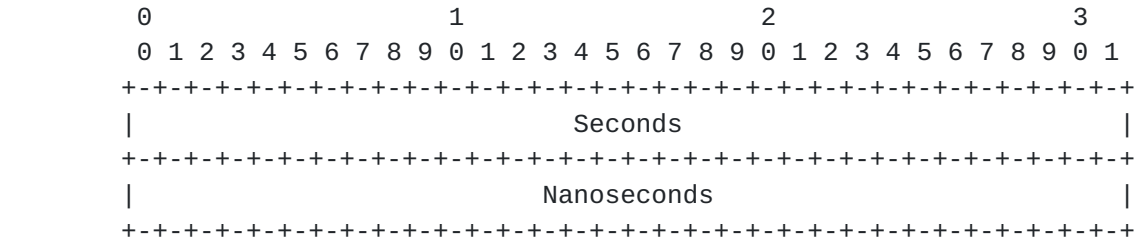


Figure 1: PTP [IEEE1588v2] Truncated Timestamp Format

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.

5.2. 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 [I-D.ietf-ntp-packet-timestamps], and the details are presented below for the sake of completeness.

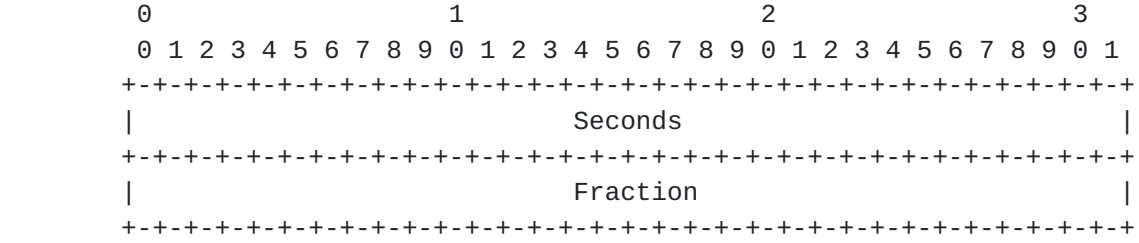


Figure 2: NTP [RFC5905] 64-bit Timestamp Format

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 may be temporarily inaccurate.

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.

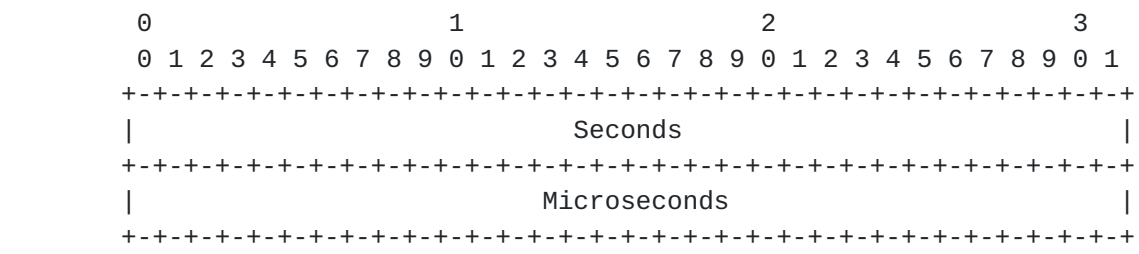


Figure 3: POSIX-based Timestamp Format

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 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 may be temporarily inaccurate.

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., 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].

7. IANA Considerations

This document requests the following IANA Actions.

7.1. Creation of a new In-Situ OAM Protocol Parameters Registry (IOAM) Protocol Parameters IANA registry

IANA is requested to create a new protocol registry for "In-Situ OAM (IOAM) Protocol Parameters". This is the common registry that will include registrations for all IOAM namespaces. Each Registry, whose names are listed below:

IOAM Type

IOAM Trace Type

IOAM Trace flags

IOAM POT Type

IOAM POT flags

IOAM E2E Type

IOAM Namespace-ID

will contain the current set of possibilities defined in this document. New registries in this name space are created via RFC Required process as per [RFC8126].

The subsequent sub-sections detail the registries herein contained.

7.2. IOAM Type Registry

This registry defines 128 code points for the IOAM-Type field for identifying IOAM options as explained in Section 4. 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].

7.3. 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 4.2. The meaning of Bits 0 - 11 for trace type are defined in this document in Paragraph 5 of Section 4.2.1:

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 variable length Opaque State Snapshot

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 23 checksum complement

The meaning for Bits 12 - 22 are available for assignment via RFC Required process as per [RFC8126].

7.4. 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 4.2. The meaning of Bit 0 (the most significant bit) for trace flags is defined in this document in Paragraph 3 of Section 4.2.1:

Bit 0 "Overflow" (O-bit)

7.5. IOAM POT Type Registry

This registry defines 256 code points to define IOAM POT Type for IOAM proof of transit option Section 4.3. 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].

7.6. IOAM POT Flags Registry

This registry defines code points for each bit in the 8 bit flags for IOAM POT option defined in Section 4.3. The meaning of Bit 0 for IOAM POT flags is defined in this document in Section 4.3:

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

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

7.7. 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 4.4. 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].

7.8. 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 4.1 of the current document. Registry entries for the values 0x8000 to 0xFFFF are to be assigned via the "Expert Review" policy defined in [RFC8126].

0: default namespace (known to all IOAM nodes)

0x0001 - 0x7FFF: reserved for private use

0x8000 - 0xFFFF: unassigned

8. 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.

The Proof of Transit option (Section Section 4.3) is used for verifying the path of data packets. The security considerations of POT are further discussed in [I-D.brockners-proof-of-transit].

The data elements of IOAM can be used for network reconnaissance, allowing attackers to collect information about network paths, performance, queue states, buffer occupancy and other information. Note that in case IOAM is used in "immediate export" mode (reference to be added in a future revision), 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. IOAM data export and securing IOAM data export 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 collectors that analyze the IOAM data. Another example is a packet length attack, in which an attacker pushes IOAM headers into data packets, causing these packets to be increased beyond the MTU size, resulting in fragmentation or in packet drops.

Since IOAM options may 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 may be implemented by misconfiguring or by maliciously configuring IOAM-enabled nodes in a way that enables other attacks. Thus, IOAM configuration should be secured in a way that authenticates authorized users and verifies the integrity of configuration procedures.

Notably, IOAM is expected to be deployed in specific network domains, thus confining the potential attack vectors to within the network domain. Indeed, in order to limit the scope of threats 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. Note that the Immediate Export mode (reference to be added in a future revision) can mitigate the potential threat of IOAM data leaking through data packets.

9. 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, Haoyu song and Robin <lizhenbin@huawei.com> 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, and Mickey Spiegel.

The authors would like to acknowledge the contribution of "Immediate export" of IOAM trace by Barak Gafni.

10. References

10.1. 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>.

10.2. Informative References

[I-D.brockners-proof-of-transit] Brockners, F., Bhandari, S., Dara, S., Pignataro, C., Leddy, J., Youell, S., Mozes, D., and T. Mizrahi, "Proof of Transit", draft-brockners-proof-of-transit-05 (work in progress), May 2018.

[I-D.ietf-ntp-packet-timestamps]
Mizrahi, T., Fabini, J., and A. Morton, "Guidelines for Defining Packet Timestamps", draft-ietf-ntp-packet-timestamps-06 (work in progress), February 2019.

[I-D.ietf-nvo3-geneve]
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-13 (work in progress), March 2019.

[I-D.ietf-nvo3-vxlan-gpe]
Maino, F., Kreeger, L., and U. Elzur, "Generic Protocol Extension for VXLAN", draft-ietf-nvo3-vxlan-gpe-07 (work in progress), April 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.lapukhov-dataplane-probe]
Lapukhov, P. and r. remy@barefootnetworks.com, "Data-plane probe for in-band telemetry collection", draft-lapukhov-dataplane-probe-01 (work in progress), June 2016.

[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-01 (work in progress), October 2018.

[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>.

Authors' Addresses

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

Email: fbrockne@cisco.com

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

Email: shwethab@cisco.com

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

Email: cpignata@cisco.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

Tal Mizrahi
Huawei Network.IO Innovation Lab
Israel

Email: tal.mizrahi.phd@gmail.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

Daniel Bernier
Bell Canada
Canada

Email: daniel.bernier@bell.ca

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

Email: john.lemon@broadcom.com