In-band Network Telemetry (INT) Dataplane Specification

Version 2.1

The P4.org Applications Working Group. Contributions from
Alibaba, Arista, CableLabs, Cisco Systems, Dell, Intel, Marvell, Netronome, VMware

2020-05-12

Contents

1. Introduction 3
2. Terminology 4
3. INT Modes of Operation 5
3.1. INT Application Modes . . . 5
3.2. INT Applied to Synthetic Traffic . . . 6
4. What To Monitor 7
4.1. Device-level Information . . . 7
4.2. Ingress Information . . . 7
4.3. Egress Information . . . 8
5. INT Headers 9
5.1. INT Header Types . . . 9
5.2. Per-Hop Header Operations . . . 9
5.2.1. INT Source Node . . . 9
5.2.2. INT Transit Hop Node . . . 10
5.2.3. INT Sink Node . . . 10
5.3. MTU Settings . . . 10
5.4. INT over any encapsulation . . . 11
5.5. Checksum Update . . . 12
5.6. Header Location . . . 13
5.6.1. INT over IPv4/GRE . . . 13
5.6.2. INT over TCP/UDP . . . 15
5.6.3. INT over VXLAN GPE . . . 18
5.6.4. INT over Geneve . . . 19
5.7. INT-MD Metadata Header Format . . . 20
5.8. INT-MX Header Format . . . 24
6. Examples 28
6.1. Example with INT-MD over TCP . . . 28
6.2. Example with INT-MX over TCP . . . 30
6.3. Example with new UDP header and INT-MD inserted before TCP . . . 31
6.4. Example with new UDP header and INT-MX inserted before TCP . . . 32
6.5. Example with INT-MD in-between UDP header and UDP payload . . . 34
6.6. Example with INT-MX in-between UDP header and UDP payload . . . 35
6.7. Example with INT-MD over IPv4/GRE (Original packet IPv4) . . . 36
6.8. Example with INT-MX over IPv4/GRE (Original packet IPv4) . . . 37
6.9. Example with INT-MD over IPv4/GRE (Original packet CE or IP) . . . 38
6.10. Example with INT-MX over IPv4/GRE (Original packet CE or IP) . . . 40
6.11. Example with INT-MD over VXLAN GPE . . . 41
6.12. Example with INT-MX over VXLAN GPE . . . 42
6.13. Example with INT-MD over Geneve . . . 43
6.14. Example with INT-MX over Geneve . . . 44
A. Appendix: An extensive (but not exhaustive) set of Metadata 44
A.1. Node-level . . . 44
A.2. Ingress . . . 45
A.3. Egress . . . 45

A.4. Buffer Information 46
A.5. Miscellaneous 46
B. Domain Specific Examples 47
B.1. Example with INT-MD including source-only metadata 47
C. Acknowledgements 48
D. Change log 48

1. Introduction

Inband Network Telemetry ("INT") is a framework designed to allow the collection and reporting of network state, by the data plane, without requiring intervention or work by the control plane in collecting and delivering the state from the data plane. In the INT architectural model, packets may contain header fields that are interpreted as "telemetry instructions" by network devices. INT traffic sources (applications, end-host networking stacks, hypervisors, NICs, send-side ToRs, etc.) can embed the instructions either in normal data packets, cloned copies of the data packets or in special probe packets. Alternatively, the instructions may be programmed in the network data plane to match on particular network flows and to execute the instructions on the matched flows.

These instructions tell an INT-capable device what state to collect. The network state information may be directly exported by the data plane to the telemetry monitoring system, or can be written into the packet as it traverses the network. When the information is embedded in the packets, INT traffic sinks retrieve (and optionally report) the collected results of these instructions, allowing the traffic sinks to monitor the exact data plane state that the packets "observed" while being forwarded.

Some examples of traffic sink behavior are described below:

- OAM – the traffic sink[1] might simply collect the encoded network state, then export that state to an external controller. This export could be in a raw format, or could be combined with basic processing (such as compression, deduplication, truncation).
- Real-time control or feedback loops – traffic sinks might use the encoded data plane information to feed back control information to traffic sources, which could in turn use this information to make changes to traffic engineering or packet forwarding. (Explicit congestion notification schemes are an example of these types of feedback loops).
- Network Event Detection - If the collected path state indicates a condition that requires immediate attention or resolution (such as severe congestion or violation of certain data-plane invariances), the traffic sinks[1] could generate immediate actions to respond to the network events, forming a feedback control loop either in a centralized or a fully decentralized fashion (a la TCP).

[1]While this will be commonly done by Sink nodes, Transit nodes may also generate OAM's or carry out Network Event Detection

[1]While this will be commonly done by Sink nodes, Transit nodes may also generate OAM's or carry out Network Event Detection

The INT architectural model is intended to be generic and enables a number of interesting high level applications, such as:

- Network troubleshooting and performance monitoring
  - Traceroute, micro-burst detection, packet history (a.k.a. postcards[2])
- Advanced congestion control
- Advanced routing
  - Utilization-aware routing (For example, HULA[3], CLOVE[4])
- Network data plane verification

A number of use case descriptions and evaluations are described in the Millions of Little Minions paper [5].

2. Terminology

Monitoring System:
A system that collects telemetry data sent from different network devices. The monitoring system components may be physically distributed but logically centralized.

INT Header:
A packet header that carries INT information. There are three types of INT Headers – *eMbed data (MD-type)*, *eMbed instruction (MX-type)* and *Destination-type* (See Section 5.1).

INT Packet:
A packet containing an INT Header.

INT Node:
An INT-capable network device that participates in the INT data plane by regularly carrying out at least one of the following: inserting, adding to, removing, or processing instructions from INT Headers in INT packets. Depending on deployment scenarios, examples of INT Nodes may include devices such as routers, switches, and NICs.

INT Instruction:
Instructions indicating which INT Metadata (defined below) to collect at each INT node. The instructions are either configured at each INT-capable node's Flow Watchlist or written into the INT Header.

Flow Watchlist:
A dataplane table that matches on packet headers and inserts or applies INT instructions on each matched flow. A flow is a set of packets having the same values on the selected header fields.

INT Source:
A trusted entity that creates and inserts INT Headers into the packets it sends. A Flow Watchlist is configured to select the flows in which INT headers are to be inserted.

[2] I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks, USENIX NSDI 2014.

[3] HULA: Scalable Load Balancing Using Programmable Data Planes, ACM SOSR 2016

[4] CLOVE: Congestion-Aware Load Balancing at the Virtual Edge, ACM CoNEXT 2017

[5] Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility, ACM SIGCOMM 2014.

INT Sink:
A trusted entity that extracts the INT Headers and collects the path state contained in the INT Headers. The INT Sink is responsible for removing INT Headers so as to make INT transparent to upper layers. (Note that this does not preclude having nested or hierarchical INT domains.) The INT Sink can decide to send the collected information to the monitoring system.

INT Transit Hop:
A trusted entity that collects metadata from the data plane by following the INT Instructions. Based on the instructions, the data may be directly exported to the telemetry monitoring system or embedded into the INT Header of the packet.

Note that one physical device may play multiple roles – INT Source, Transit, Sink – at the same time for the same or different flows. For example, an INT Source node may embed its own metadata into the packet, playing the roles of INT Transit as well.

INT Metadata:
Information that an INT Source or an INT Transit Hop node inserts into the INT Header, or into a telemetry report. Examples of metadata are described in section 4.

INT Domain:
A set of inter-connected INT nodes under the same administration. This specification defines the behavior and packet header formats for interoperability between INT nodes from different vendors in an INT domain. The INT nodes within the same domain must be configured in a consistent way to ensure interoperability between the nodes. Operators of an INT domain should deploy INT Sink capability at domain edges to prevent INT information from leaking out of the domain.

3. INT Modes of Operation

Since INT was first introduced at P4.org in 2015, a number of variations of INT have been evolved and discussed in IETF and industry communities. Also the term 'INT' has been used to broadly indicate data plane telemetry in general, not limited to the original classic INT where both instructions and metadata are embedded in the data packets. Hence we define different modes of INT operation based on the degree of packet modifications, i.e., what to embed in the packets.

The different modes of operation are described in detail below, and summarized in Figure 1.

3.1. INT Application Modes

Original data packets are monitored and may be modified to carry INT instructions and metadata. There are three variations based on the level of packet modifications.

- **INT-XD** (eXport Data): INT nodes directly export metadata from their dataplane to the monitoring system based on the INT instructions configured at their Flow Watchlists. No packet Modification is needed.

  This mode was also known as "Postcard" mode in the previous versions of the Telemetry Report spec, originally inspired by [2].

[2] I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks, USENIX NSDI 2014.

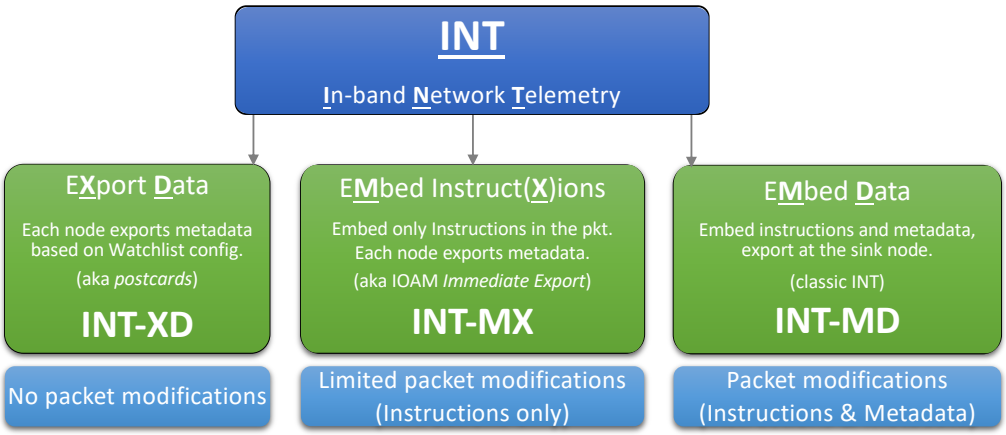


Figure 1. Various modes of INT operation.

- **INT-MX** (eMbed instruct(X)ions): The INT Source node embeds INT instructions in the packet header, then the INT Source, each INT Transit, and the INT sink directly send the metadata to the monitoring system by following the instructions embedded in the packets. The INT Sink node strips the instruction header before forwarding the packet to the receiver. Packet modification is limited to the instruction header, the packet size doesn't grow as the packet traverses more Transit nodes.

  This mode is inspired by IOAM's "Direct Export" [6] [7].

- **INT-MD** (eMbed Data): In this mode both INT instructions and metadata are written into the packets. This is the classic hop-by-hop INT where 1) INT Source embeds instructions, 2) INT Source & Transit embed metadata, and 3) INT Sink strips the instructions and aggregated metadata out of the packet and (selectively) sends the data to the monitoring system. The packet is modified the most in this mode while it minimizes the overhead at the monitoring system to collate reports from multiple INT nodes.

  Since v2.0, INT-MD mode supports 'source-only' metadata as part of Domain Specific Instructions. This allows the INT Source to embed additional metadata for the INT Sink or the monitoring system to consume.

NOTE: the rest of the spec is assuming INT-MD as the default mode, unless specified otherwise.

3.2. INT Applied to Synthetic Traffic

INT Source nodes may generate INT-marked synthetic traffic either by cloning original data packets or by generating special probe packets. INT is applied to this traffic by transit nodes in exactly the same way as all traffic.

The only difference between live traffic and Synthetic traffic is that INT-Sink nodes may need to discard synthetic traffic after extracting the collected INT data as opposed to forwarding the

[6] Data Fields for In-situ OAM, draft-ietf-ippm-ioam-data-09, March 2020.
[7] In-situ OAM Direct Exporting draft-ietf-ippm-ioam-direct-export-00, February 2020.

traffic. This is indicated by using the 'D' bit of the INT-Header to mark relevant packets as being copies/clones or probes, to be 'D'iscarded at the INT-Sink.

All INT modes may be used on these synthetic/probe packets, as decided by the INT-Source node. Specifically the **INT-MD** (eMbed Data) mode applied to Synthetic or probe packets allows functionality similar to **IFA**[8].

It is likely that synthetic traffic created by cloning would be discarded at the Sink, while Probe packets might be marked for forwarding or discarding, depending on the use-case. It is the responsibility of the INT-Source node to mark packets correctly to determine if the INT-Sink will forward or discard packets after extracting the INT Data collected along the path.

4. What To Monitor

In theory, one may be able to define and collect any device-internal information using the INT approach. In practice, however, it seems useful to define a small baseline set of metadata that can be made available on a wide variety of devices: the metadata listed in this section comprises such a set. As the INT specification evolves, we expect to add more metadata to this INT specification.

The exact meaning of the following metadata (e.g., the unit of timestamp values, the precise definition of hop latency, queue occupancy or buffer occupancy) can vary from one device to another for any number of reasons, including the heterogeneity of device architecture, feature sets, resource limits, etc. Thus, defining the exact meaning of each metadata is beyond the scope of this document. Instead we assume that the semantics of metadata for each device model used in a deployment is communicated with the entities interpreting/analyzing the reported data in an out-of-band fashion.

4.1. Device-level Information

Node id

The unique ID of an INT node. This is generally administratively assigned. Node IDs must be unique within an INT domain.

4.2. Ingress Information

Ingress interface identifier

The interface on which the INT packet was received. A packet may be received on an arbitrary stack of interface constructs starting with a physical port. For example, a packet may be received on a physical port that belongs to a link aggregation port group, which in turn is part of a Layer 3 Switched Virtual Interface, and at Layer 3 the packet may be received in a tunnel. Although the entire interface stack may be monitored in theory, this specification allows for monitoring of up to two levels of ingress interface identifiers. The first level of ingress interface identifier would typically be used to monitor the physical port on which the packet was received, hence a 16-bit field (half of a 4-Byte metadata) is deemed adequate. The second level of ingress interface identifier occupies a full 4-Byte metadata field, which may be used to monitor a logical interface on which the packet was received. A 32-bit space at the second level allows for an adequately large number of logical interfaces at each network element. The semantics of interface identifiers

[8] Inband Flow Analyzer, draft-kumar-ippm-ifa-02, April 2020.

may differ across devices, each INT hop chooses the interface type it reports at each of the two levels.

Ingress timestamp
The device local time when the INT packet was received on the ingress physical or logical port.

4.3. Egress Information

Egress interface identifier
The interface on which the INT packet was sent out. A packet may be transmitted on an arbitrary stack of interface constructs ending at a physical port. For example, a packet may be transmitted on a tunnel, out of a Layer 3 Switched Virtual Interface, on a Link Aggregation Group, out of a particular physical port belonging to the Link Aggregation Group. Although the entire interface stack may be monitored in theory, this specification allows for monitoring of up to two levels of egress interface identifiers. The first level of egress interface identifier would typically be used to monitor the physical port on which the packet was transmitted, hence a 16-bit field (half of a 4-Byte metadata) is deemed adequate. The second level of egress interface identifier occupies a full 4-Byte metadata field, which may be used to monitor a logical interface on which the packet was transmitted. A 32-bit space at the second level allows for an adequately large number of logical interfaces at each network element. The semantics of interface identifiers may differ across devices, each INT hop chooses the interface type it reports at each of the two levels.

Egress timestamp
The device local time when the INT packet was processed by the egress physical or logical port.

Hop latency
Time taken for the INT packet to be switched within the device.

Egress interface TX Link utilization
Current utilization of the egress interface via which the INT packet was sent out. Again, devices can use different mechanisms to keep track of the current rate, such as bin bucketing or moving average. While the latter is clearly superior to the former, the INT framework does not stipulate the mechanics and simply leaves those decisions to device vendors.

Queue occupancy
The build-up of traffic in the queue (in bytes, cells, or packets) that the INT packet observes in the device while being forwarded. The format of this 4-octet metadata field is implementation specific and the metadata semantics YANG model shall describe the format and units of this metadata field in the metadata stack.

Buffer occupancy
The build-up of traffic in the buffer (in bytes, cells, or packets) that the INT packet observes in the device while being forwarded. Use case is when the buffer is shared between multiple queues. The format of this 4-octet metadata field is implementation specific and the metadata semantics YANG model shall describe the format and units of this metadata field in the metadata stack.

Details of the metadata semantics YANG model can be accessed at the link below:
https://github.com/p4lang/p4-applications/blob/master/telemetry/code/models/p4-dtel-metadata-semantics.yang

5. INT Headers

This section specifies the format and location of INT Headers. INT Headers and their locations are relevant for INT-MX and INT-MD modes where the INT instructions (and metadata stack in case of MD mode) are written into the packets.

5.1. INT Header Types

There are three types of INT Headers: MD-type, MX-type and Destination-type. A given INT packet may carry either of MD or MX type headers, and/or a Destination-type header. When Destination-type and MD-type or MX-type headers are present, the MD-type header or MX-type header must precede the Destination-type header.

- MD-type (**INT Header type 1**)
  - Intermediate nodes (INT Transit Hops) must process this type of INT Header. The format of this header is defined in section 5.7.
- Destination-type (**INT Header type 2**)
  - Destination headers must only be consumed by the INT Sink. Intermediate nodes must ignore Destination headers.
  - Destination headers can be used to enable Edge-to-Edge communication between the INT Source and INT Sink. For example:
    * INT Source can add a sequence number to detect loss of INT packets.
    * INT Source can add the original values of IP TTL and INT Remaining Hop Count, thus enabling the INT sink to detect network devices on the path that do not support INT by comparing the IP TTL decrement against INT Remaining Hop Count decrement (assuming each network device is an L3 hop)
  - The format of Destination-type headers will be defined in a future revision. Note some Edge-to-Edge INT use cases can be supported by 'source-only' metadata, part of Domain Specific Instructions in the MD-type header.
- MX-type (**INT Header type 3**)
  - Intermediate nodes (INT Transit Hops) must process this type of INT Header and generate reports to the monitoring system as instructed. The format of this header is defined in section 5.8.

5.2. Per-Hop Header Operations

5.2.1. INT Source Node

In the INT-MD and INT-MX modes, the INT Source node in the packet forwarding path creates the INT-MD or INT-MX Header.

In INT-MD, the source node add its own INT metadata after the header. To avoid exhausting header space in the case of a forwarding loop or any other anomalies, it is strongly recommended to limit the number of total INT metadata fields added by Transit Hop nodes by setting the *Remaining Hop Count* field in INT header appropriately.

The INT-MD and INT-MX headers are described in detail in the subsequent sections.

5.2.2. INT Transit Hop Node

In the INT-MD mode, each node in the packet forwarding path creates additional space in the INT-MD Header on-demand to add its own INT metadata. To avoid exhausting header space in the case of a forwarding loop or any other anomalies, each INT Transit Hop must decrement the *Remaining Hop Count* field in the INT header appropriately.

In the INT-MX mode, each node in the packet forwarding path follows the intructions in the INT-MX Header, gathers the device specific metadata and exports the device metadata using the Telemetry Report.

INT Transit Hop nodes may update the *DS Flags* field in the INT-MD or INT-MX header. The *Hop ML*, *Instruction Bitmap*, *Domain Specific ID* and *DS Instruction* fields must not be modified by Transit Hop nodes.

5.2.3. INT Sink Node

In INT-MD mode, the INT Sink node removes the INT Headers and Metadata stack from the packet, and decides whether to report the collected information.

In INT-MX mode, the INT Sink node removes the INT-MX header, gathers the device specific metadata and decides whether to report that metadata.

5.3. MTU Settings

In both INT-MX and INT-MD modes, it is possible that insertion of the INT header at the INT Source node may cause the egress link MTU to be exceeded.

In INT-MD mode, as each hop creates additional space in the INT header to add its metadata, the packet size increases. This can potentially cause egress link MTU to be exceeded at an INT node.

This may be addressed in the following ways -

- It is recommended that the MTU of links between INT sources and sinks be configured to a value higher than the MTU of preceding links (server/VM NIC MTUs) by an appropriate amount. Configuring an MTU differential of [Per-hop Metadata Length*4*INT Hop Count + Fixed INT Header Length] bytes (just [Fixed INT Header Length] for INT-MX mode), based on conservative values of total number of INT hops and Per-hop Metadata Length, will prevent egress MTU being exceeded due to INT metadata insertion at INT hops. The Fixed INT Header Length is the sum of INT metadata header length (12B) and the size of encapsulation-specific shim/option header (4B) as defined in section 5.6.

- An INT source/transit node may optionally participate in dynamic discovery of Path MTU for flows being monitored by INT by transmitting ICMP message to the traffic source as per Path MTU Discovery mechanisms of the corresponding L3 protocol (RFC 1191 for IPv4, RFC 1981 for IPv6). An INT source or transit node may report a conservative MTU in the ICMP message, assuming that the packet will go through the maximum number of allowed INT hops (i.e. *Remaining Hop Count* will decrement to zero), accounting for cumulative metadata insertion at all INT hops, and assuming that the egress MTU at all downstream INT hops is the same as its own egress link MTU. This will help the path MTU discovery source to converge to a path MTU estimate faster, although this would be a conservative path MTU estimate. Alternatively, each INT hop may report an MTU only accounting for

the metadata it inserts. This would enable the path MTU discovery source converge to a precise path MTU, at the cost of receiving more ICMP messages, one from each INT hop.

Regardless of whether or not an INT transit node participates in Path MTU discovery, if it cannot insert all requested metadata because doing so will cause the packet length to exceed egress link MTU, it must either:

- not insert any metadata and set the M bit in the INT header, indicating that egress MTU was exceeded at an INT hop, or
- report the metadata stack collected from previous hops (setting the Intermediate Report bit if a Telemetry Report 2.0 packet is generated) and remove the reported metadata stack from the packet, including the metadata from this transit hop in either the report or embedding in the INT-MD metadata header.

An INT source inserts 12 bytes of fixed INT headers, and may also insert Per-hop Metadata Length*4 bytes of its own metadata. If inserting the fixed headers causes egress link MTU to be exceeded, INT cannot not be initiated for such packets. If an INT source is programmed to insert its own INT metadata, and there is enough room in a packet to insert fixed INT headers, but no additional room for its INT metadata, the source must initiate INT and set the M bit in the INT header.

In theory, an INT transit node can perform IPv4 fragmentation to overcome egress MTU limitation when inserting its metadata. However, IPv4 fragmentation can have adverse impact on applications. Moreover, IPv6 packets cannot be fragmented at intermediate hops. Also, fragmenting packets at INT transit hops, with or without copying preceding INT metadata into fragments imposes extra complexity of correlating fragments in the INT monitoring engine. Considering all these factors, this specification requires that an INT node must not fragment packets in order to append INT information to the packet.

5.4. INT over any encapsulation

The specific location for INT Headers is intentionally not specified: an INT Header can be inserted as an option or payload of any encapsulation type. The only requirements are that the encapsulation header provides sufficient space to carry the INT information and that all INT nodes (Sources, transit hops and Sinks) agree on the location of the INT Headers. The following choices are potential encapsulations using common protocol stacks, although a deployment may choose a different encapsulation format if better suited to their needs and environment.

- INT over VXLAN (as VXLAN payload, per GPE extension)
- INT over Geneve (as Geneve option)
- INT over NSH (as NSH payload)
- INT over TCP (as payload)
- INT over UDP (as payload)
- INT over GRE (as a shim between GRE header and encapsulated payload)

5.5. Checksum Update

As described above in section 5.4, INT headers and metadata may be carried in an L4 protocol such as TCP or UDP, or in an encapsulation header that includes an L4 header, such as VXLAN. The checksum field in the TCP or UDP L4 header needs to be updated as INT nodes modify the L4 payload via insertion/removal of INT headers and metadata. However, there are certain exceptions. For example, when UDP is transported over IPv4, it is possible to assign a zero checksum, causing the receiver to ignore the value of the checksum field (as defined in RFC 768). For UDP over IPv6, there are specific use cases in which it is possible to assign a zero Checksum (as defined in RFC 6936).

INT source, transit and sink nodes must comply with IETF standards for Layer 4 transport protocols with respect to whether or not Layer 4 checksum is to be updated upon modification of Layer 4 payload. For example, if an INT source/transit/sink hop receives UDP traffic with zero L4 checksum, it must not update the L4 checksum in conformance with the behavior defined in relevant IETF standards such as RFC 768 and RFC 6936.

When L4 checksum update is required, an INT source/transit node may update the checksum in one of two ways:

- Update the L4 Checksum field such that the new value is equal to the checksum of the new packet, after the INT-related updates (header additions/removals, field updates), or
- If the INT source indicates that Checksum-neutral updates are allowed by setting an instruction bit corresponding to the Checksum Complement metadata, then the INT source/transit nodes may assign a value to the Checksum Complement metadata which guarantees that the existing L4 Checksum is the correct value of the packet after the INT-related updates.

The motivation for the Checksum Complement is that some hardware implementations process data packets in a serial order, which may impose a problem when INT fields and metadata that reside after the L4 Checksum field are inserted or modified. Therefore, the Checksum Complement metadata, if present, is the last metadata field in the stack.

Note that when the Checksum Complement metadata is present source/transit nodes may choose to update the L4 Checksum field instead of using the Checksum Complement metadata. In this case the Checksum Complement metadata must be assigned the reserved value 0xFFFFFFFF. A host that verifies the L4 Checksum will be unaffected by whether some or all of the nodes chose not to use the Checksum Complement, since the value of the L4 Checksum should fit the Checksum of the payload in either of the cases.

INT sink cannot perform a Checksum-neutral update using Checksum Complement metadata as it removes all INT headers from the packet. Thus, an INT sink when performing a checksum update has to do so by updating the L4 Checksum field.

Regardless of whether checksum update is performed via modifying the L4 checksum field or via use of Checksum Complement metadata, performing the update based on an incremental checksum calculation (as is typically done) will ensure that any potential corruption is detected at the point of checksum validation. If full checksum computation is performed at an INT node, it should be preceded by checksum validation so as to not mask out any corruption at preceding hops.

5.6. Header Location

We describe four encapsulation formats in this specification, covering different deployment scenarios, with and without network virtualization:

1. *INT over IPv4/GRE* - INT headers are carried between the GRE header and the encapsulated GRE payload.
2. *INT over TCP/UDP* - A shim header is inserted following TCP/UDP header. INT Headers are carried between this shim header and TCP/UDP payload. Since v2.0, the spec also supports an option to insert a new UDP header (followed by INT headers) before the existing L4 header. This approach doesn't rely on any tunneling/virtualization mechanism and is versatile to apply INT to both native and virtualized traffic.
3. *INT over VXLAN* - VXLAN generic protocol extensions [9] are used to carry INT Headers between the VXLAN header and the encapsulated VXLAN payload.
4. *INT over Geneve* - Geneve is an extensible tunneling framework, allowing Geneve options to be defined for INT Headers.

5.6.1. INT over IPv4/GRE

In case the traffic being monitored is not encapsulated by any virtualization header, INT over VXLAN or INT over Geneve is not helpful. Instead, a GRE encapsulation as defined in RFC 2784 [10] can be utilized. The INT metadata header and INT metadata follows the GRE header. In an administrative domain where INT is used, insertion of the INT metadata header and metadata in GRE is enabled at the INT source and deletion of INT metadata header and metadata is enabled at the INT sink by means of configuration.

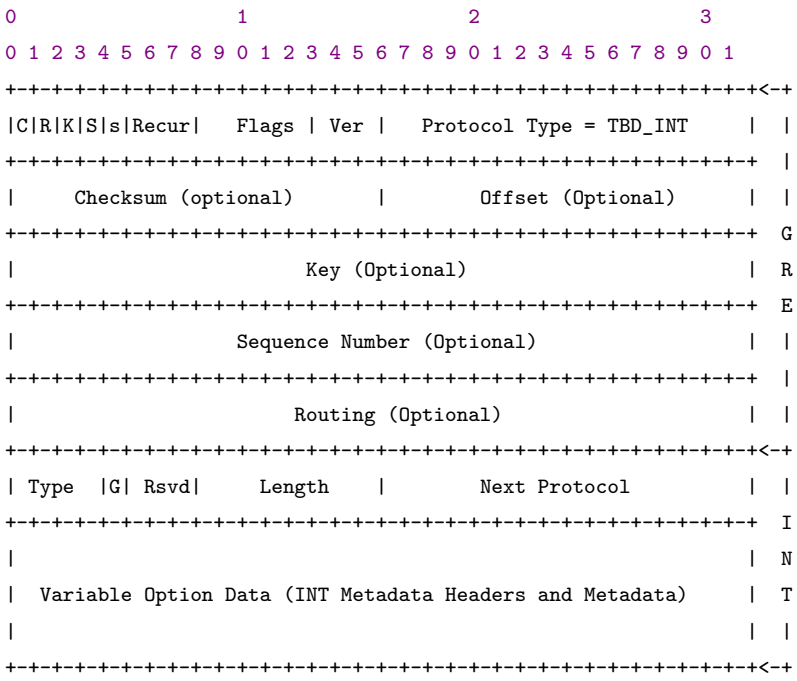
There are two scenarios when utilizing GRE encapsulation to support INT:

1. If the incoming packet at the source node of the INT domain is GRE encapsulated, then the source node should add the INT Metadata Header and Metadata following the GRE header. The sink node of the INT domain should remove the INT Metadata Header and Metadata stack before forwarding the GRE encapsulated packet to the destination.

2. If the incoming packet at the source node of the INT domain is not GRE encapsulated, then the source node should add a GRE encapsulation and insert the INT Metadata Header and Metadata following the GRE header. The sink node of the INT domain should remove the GRE encapsulation along with removing the INT Metadata Header and the Metadata stack before forwarding the packet to the destination.

[9] Generic Protocol Extension for VXLAN, draft-ietf-nvo3-vxlan-gpe-09, December 2019.
[10] Generic Routing Encapsulation (GRE), RFC 2784, March 2000.

IPv4 GRE Option format for carrying INT Header and Metadata:



The GRE header and fields are defined in RFC 2784 [10]. The GRE Protocol Type value is TBD_INT.

The INT Shim header for GRE option is defined as follows:

- **Type (4b):** This field indicates the type of INT Header following the shim header. The Type values are defined in Section 5.1.
- **G (1b):** Indicates whether the GRE headers were inserted to transport INT by the INT source.
  - **0:** Original packet (before insertion of INT headers and metadata) had GRE encapsulation.
  - **1:** Original packet had no GRE encapsulation, hence the INT source inserted GRE.
  - This is a hint that helps the INT sink (when it is not the GRE tunnel endpoint) determine whether to remove the GRE headers part of INT decapsulation (if G=1).
- **Rsvd (3b):** reserved for future use, set to zero upon transmission and ignored upon reception.
- **Length (8b):** This is the total length of INT metadata header, INT stack excluding the shim header in 4-byte words. A non-INT device may read this field and skip over INT headers.
- **Next Protocol (16b):** this field contains an EtherType value (defined in the IANA registry [11]) indicating the type of the protocol following the INT stack. An implementation receiving a packet containing a type value which is not listed in the registry should discard the packet.

[10] Generic Routing Encapsulation (GRE), RFC 2784, March 2000.
[11] IANA Ethernet Numbers.

5.6.2. INT over TCP/UDP

In case the traffic being monitored is not encapsulated by any virtualization header, one can also put the INT metadata just after layer 4 headers (TCP/UDP). The scheme assumes that the non-INT devices between the INT source and the INT sink either do not parse beyond layer-4 headers or can skip through the INT stack using the Length field in the INT shim header. If TCP has any options, the INT stack may come before or after the TCP options but the decision must be consistent within an INT domain.

Note that INT over UDP can be used even when the packet is encapsulated by VXLAN, Geneve, or GUE (Generic UDP Encapsulation). INT over TCP/UDP also makes it easier to add INT stack into outer, inner, or even both layers. In such cases both INT header stacks carry information for respective layers and need not be considered interfering with each other.

A field in Ethernet, IP, or TCP/UDP should indicate if the INT header exists after the TCP/UDP header. We propose three options.

1. UDP destination port field: a new UDP port number (INT_TBD) will be assigned by IANA to indicate the existence of INT after UDP. This option supports two cases:

   - The original packet already has UDP header either as user application protocol or as part of another UDP-based encapsulation such as VXLAN, GENEVE, RoCEv2. INT is inserted after the UDP header with the UDP destination port number changed to INT_TBD. The original destination port number is carried in the shim header for the INT sink to restore, when it removes the INT stack from the packet.
   - A new UDP header for INT is inserted between IP and the existing L4 header. The protocol field of IP header is set to 17 for UDP and the original IP protocol value is carried in the INT shim header. In the new UDP header, INT_TBD is used as the destination port number. It is recommended that the source port number of the new UDP header be calculated using a hash of fields from the original packet, for example the original outer 5 tuple or the original L4 header fields. This is to enable a level of entropy for ECMP/LAG load balancing logic. It is recommended that the checksum in the new UDP header be set to zero. For IPv6 packets, this falls under the case of tunnel protocols, which are allowed to use zero UDP checksums as specified in RFC 6936. The existing L4 header will typically include a checksum computed using the encapsulating IPv6 header fields, thus offering some protection against IPv6 header corruption.

   In both cases, traffic with INT headers is likely to be hashed to a different path in the network as the new UDP destination port (INT_TBD) becomes part of the outer 5 tuple used by ECMP.

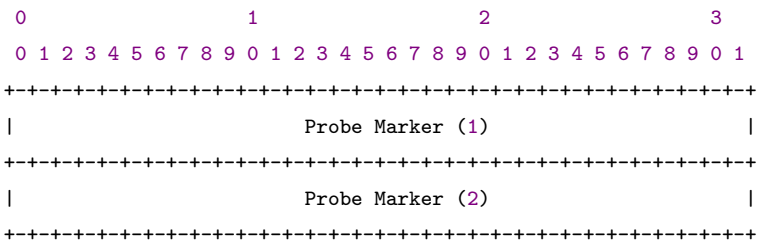
   The INT shim header for UDP has a field NPT (Next Protocol Type) that indicates which of the two cases are applied to a given INT packet. In case a new UDP header was inserted, INT sink must copy the original IP protocol number from the shim header to IP header, and strip the newly added UDP header with all INT headers. For the case that original packet already had UDP header, INT sink must restore the original destination port number from the shim header into the UDP header and strip the INT headers.

2. IPv4 DSCP or IPv6 Traffic Class field: A value or a bit can be used to indicate the existence of INT after TCP/UDP. When the INT source inserts the INT header into a packet, it sets the reserved value in the field or sets the bit. The INT source may write the original DSCP value in the INT headers so that the INT sink can restore the original value. Restoring the original value is optional.

    - Allocating a bit, as opposed to a value codepoint, will allow the rest of DSCP field to be used for QoS, hence allowing the coexistence of DSCP-based QoS and INT. If the traffic being monitored is subjected to QoS services such as rate limiting, shaping, or differentiated queueing based on DSCP field, QoS classification in the network must be programmed to ignore the designated bit position to ensure that the INT-enabled traffic receives the same treatment as the original traffic being monitored.
    - In brownfield scenarios, however, the network operator may not find a bit available to allocate for INT but may still have a fragmented space of 32 unused DSCP values. The operator can allocate an INT-enabled DSCP value for every QoS DSCP value, map the INT-enabled DSCP value to the same QoS behavior as the corresponding QoS DSCP value. This may double the number of QoS rules but will allow the co-existence of DSCP-based QoS and INT even when a single DSCP bit is not available for INT.
    - Within an INT domain, DSCP values used for INT must exclusively be used for INT. INT transit and sink nodes must not receive non-INT packets marked with DSCP values used for INT. Any time a node forwards a packet into the INT domain and there is no INT header present, it must ensure that the DSCP/Traffic class value is not the same as any of the values used to indicate INT.

3. Probe Marker fields: If DSCP field or values cannot be reserved for INT, probe marker option could be used. A specific 64-bit value can be inserted after the TCP/UDP header to indicate the existence of INT after TCP/UDP. These fields must be interpreted as unsigned integer values in network byte order. This approach is a variation of an early IETF draft with existing implementation[12].

    INT probe marker for TCP/UDP:



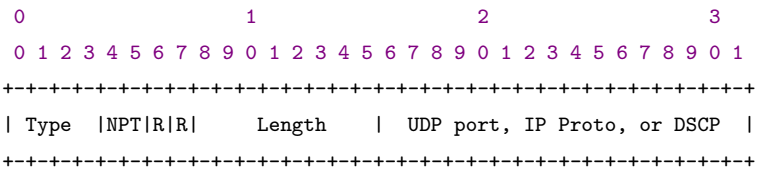
    With arbitrary values being inserted after TCP/UDP header as probe markers, the likelihood of conflicting with user traffic in a data center is low, but cannot be completely eliminated. To further reduce the chance of conflict, a deployment could choose to also examine TCP/UDP port numbers to validate INT probe marker.

[12] Data-plane probe for in-band telemetry collection, draft-lapukhov-dataplane-probe-01, June 2016.

Any of the above options may be used in an INT domain, provided that the INT transit and sink nodes in the INT domain comply with the mechanism chosen at the INT sources, and are able to correctly identify the presence and location of INT headers. The above approaches are not intended to interoperate in a mixed environment, for example it would be incorrect to mark a packet for INT using both DSCP and probe marker, as INT nodes that only understand DSCP marking and do not recognize probe markers may incorrectly interpret the first four bytes of the probe marker as INT shim header. It is strongly recommended that only one option be used within an INT domain.

We introduce an INT shim header for TCP/UDP. The INT metadata header and INT metadata stack will be encapsulated between the shim header and the TCP/UDP payload.

INT shim header for TCP/UDP:



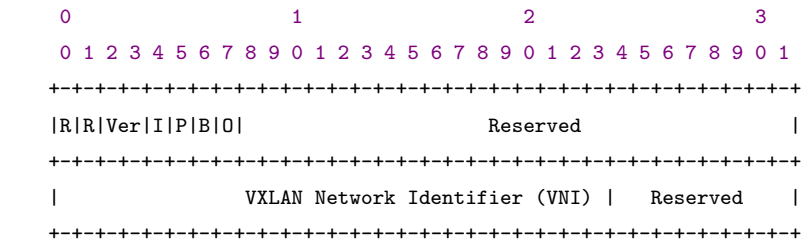
- **Type (4b):** This field indicates the type of INT Header following the shim header. The Type values are defined in Section 5.1.
- **NPT (Next Protocol Type, 2b):** This field is meaningful only when the UDP destination port number (INT_TBD) is used to indicate the existence of INT. In the other cases, this field must be zero. When UDP destination port is INT_TBD, this field may have one of the two values:
  - **one (1):** indicates that the original UDP payload follows the INT stack, and the last two bytes of the shim header carry the original UDP destination port.
  - **two (2):** indicates that another (the original) L4 header follows the INT stack, and the last byte of the shim header carries the IP protocol value for the L4 layer.
- **Length (8b):** This is the total length of INT metadata header and INT stack in 4-byte words. The length of the shim header (1 word) is NOT counted since INT version 2.0. A non-INT device may read this field and skip over INT headers.
- **UDP port, IP proto, or DSCP (16b):** The contents of this field differ depending on the value of NPT.
  - **NPT=0:** The first byte and the last two bits of this 16b field are reserved, set to zero upon transmission and ignored upon reception. The first 6 bits of the second byte may optionally carry the original DSCP value.
  - **NPT=1:** The original UDP destination port value.
  - **NPT=2:** The first byte is reserved, set to zero upon transmission and ignored upon reception. The second byte carries the original IP protocol value.

The other bits in the shim header are reserved (R) for future use, set to zero upon transmission and ignored upon reception.

5.6.3. INT over VXLAN GPE

VXLAN is a common tunneling protocol for network virtualization and is supported by most software virtual switches and hardware network elements. The VXLAN header as defined in RFC 7348 is a fixed 8-byte header as shown below.

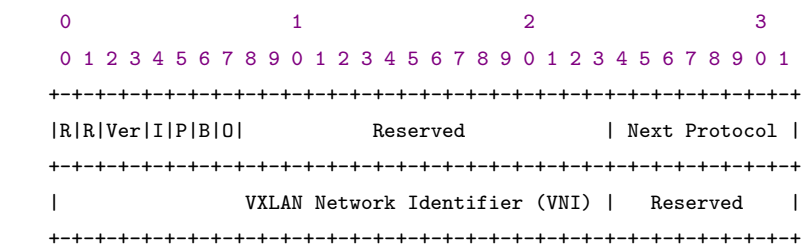
VXLAN Header:



The amount of free space in the VXLAN header allows for carrying minimal network state information. Hence, we embed INT metadata in a shim header between the VXLAN header and the encapsulated payload.

The VXLAN header as defined in RFC 7348 does not specify the protocol being encapsulated and assumes that the payload following the VXLAN header is an Ethernet payload. Internet draft draft-ietf-nvo3-vxlan-gpe[9] proposes changes to the VXLAN header to allow for multi-protocol encapsulation. We use this VXLAN generic protocol extension draft and propose a new "Next Protocol" field value for INT.

VXLAN GPE Header:



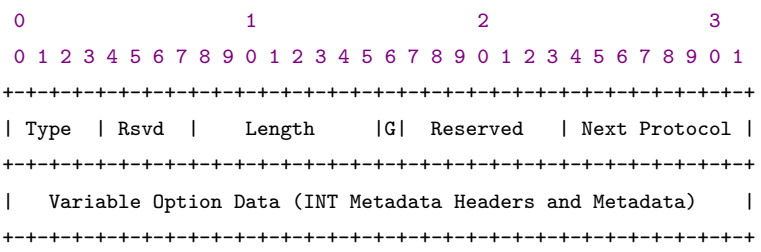
- **P bit:** Flag bit 5 is defined as the Next Protocol bit. The P bit MUST be set to 1 to indicate the presence of the 8-bit next protocol field.
- **Next Protocol Values:**
  - 0x01: IPv4
  - 0x02: IPv6
  - 0x03: Ethernet
  - 0x04: Network Service Header (NSH)
  - 0x05 to 0x7F: Unassigned
  - 0x80 to 0xFF: Unassigned (shim headers)
  - **0x82:** In-band Network Telemetry Header (This value has not been reserved by VXLAN GPE specification yet, and is hence subject to change)

[9] Generic Protocol Extension for VXLAN, draft-ietf-nvo3-vxlan-gpe-09, December 2019.

When there is one INT Header in the VXLAN GPE stack, the VXLAN GPE header for the INT Header will have a next protocol value other than INT Header indicating the payload following the INT Header - typically Ethernet. If there are multiple INT Headers in the VXLAN GPE stack (for example if both MD and destination type INT headers are being carried), then all VXLAN GPE shim headers for the INT Headers other than the last one will carry 0x82 for their next protocol values, and the VXLAN GPE header for the last INT Header will carry next protocol value of the original VXLAN payload (e.g., Ethernet).

To embed a variable-length data (i.e., INT metadata) in the VXLAN GPE stack, we introduce the INT shim header. This header follows each VXLAN GPE header for INT.

INT shim header for VXLAN GPE encapsulation:

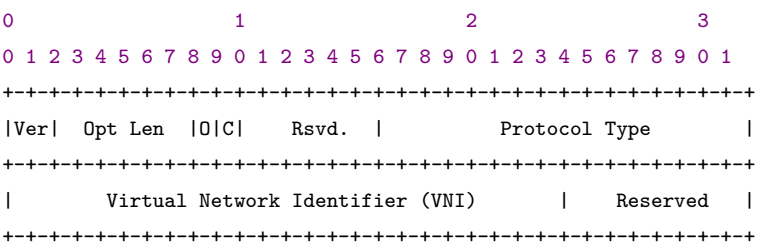


- **Type (4b):** This field indicates the type of INT Header following the shim header. The Type values are defined in Section 5.1.
- **Rsvd (4b):** These 4 bits must be set to zero in order to allign with the shim header format recommended by Internet draft draft-ietf-nvo3-vxlan-gpe, which allocates 8 bits for the Type field.
- **Length:** This is the total length of the variable INT option data, not including the shim header, in 4-byte words.
- **G:** Indicates whether the original packet (before insertion of INT headers and metadata) used a VXLAN or VXLAN GPE encapsulation.
  - **0:** Original packet used VXLAN GPE encapsulation.
  - **1:** Original packet used VXLAN encapsulation.
  - This may be used as a hint that helps the INT sink (when it is not the VTEP) determine whether to progress the packet using a VXLAN GPE encapsulation, or whether to convert the VXLAN GPE encapsulation back to a VXLAN (without GPE) encapsulation.

5.6.4. INT over Geneve

Geneve is a generic and extensible tunneling framework, allowing for INT metadata to be carried in TLV format as “Option headers” in the tunnel header.

Geneve Header:



```
|                  Variable Length Options                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Geneve Option for INT:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Option Class         |      Type     |R|R|R| Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Variable Option Data (INT Metadata Headers and Metadata)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note:

- We do not need to reserve any special values for fields in the base Geneve header for INT.
- Users may or may not use INT with Geneve along with VNI (network virtualization), though using INT with Geneve without network virtualization would be a bit wasteful.
- The Geneve **Option Class** codepoint 0x0103 has been tentatively assigned for INT [13].
- The Geneve Option **Type** field indicates the type of INT Header in the Geneve Option. The Type values are defined in Section 5.1.
- The variable length option data following the Geneve Option Header carries the actual INT metadata header and metadata.
- The Length field of the Geneve Option header is 5-bits long, which limits a single Geneve option instance to no more than 124 bytes long (31 *4)*. Remaining Hop Count* in INT-MD type header has to be set accordingly at the INT source to ensure that the Geneve option does not overflow. The entire INT-MD header must fit in a single Geneve option.

5.7. INT-MD Metadata Header Format

In this section, we define the format of the INT-MD metadata header, and the metadata itself.

INT-MD Metadata Header and Metadata Stack:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver = 2|D|E|M|       Reserved        | Hop ML  |RemainingHopCnt|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Instruction Bitmap       |        Domain Specific ID     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         DS Instruction        |            DS Flags           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| INT Metadata Stack (Each hop inserts Hop ML * 4B of metadata) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             . . .                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Last INT metadata                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

[13] IANA Network Virtualization Overlay (NVO3), Geneve Option Class

- INT-MD metadata header is 12 bytes long followed by a stack of INT metadata. Each metadata is either 4 bytes or 8 bytes in length. Each INT hop adds the same length of metadata, except for the source node if there is any 'source-only' metadata. The total length of the metadata stack is variable as different packets may traverse different paths and hence different number of INT hops.
- **Ver (4b):** INT metadata header version. Should be 2 for this version.
- **D (1b):** Discard.
  - INT Sink must Discard the packet after Extracting INT-MD metadata.
- **E (1b):** Max Hop Count exceeded.
  - This flag must be set if a node cannot prepend its own metadata due to the *Remaining Hop Count* reaching zero.
  - E bit must be set to 0 by INT source
- **M (1b):** MTU exceeded
  - This flag must be set if a node cannot add all of the requested metadata because doing so will cause the packet length to exceed egress link MTU. In this case, the node must not add any metadata to the packet, and set the M bit in the INT header. Note that it is possible for egress MTU limitation to prevent INT metadata insertion at multiple hops along a path. The M bit simply serves as an indication that INT metadata was not inserted at one or more hops and corrective action such as reconfiguring MTU at some links may be needed, particularly when INT nodes are not participating in path MTU discovery. The M bit is not aimed at readily identifying which node(s) did not insert INT metadata due to egress MTU limitation. In theory, if this does not occur at consecutive hops, it may be possible for the monitoring system to derive which node(s) set the M bit based on knowledge of the network topology and "Node ID, Ingress interface ID, Egress interface ID" tuples in the INT metadata stack.
- **R (12b):** Reserved bits, should be set to 0 by the INT source and ignored by other nodes.
- **Hop ML (5b):** Per-hop Metadata Length. This is the length of metadata including the Domain Specific Metadata in 4-Byte words to be inserted at each INT transit hop.
  - *Hop ML* is set by the INT source for transit and sink hops to abide by. If an INT domain uses 'source-only' Domain Specific Metadata, defined below, the length of the source-only Domain Specific Metadata is *excluded* from the *Hop ML*.
  - The largest value of *Hop ML* for baseline and domain specific metadata is 31.
  - An INT-capable node may be limited in the maximum number of instructions it can process and/or maximum length of metadata it can insert in data packets. An INT hop that cannot process all instructions must still insert *Hop ML* * 4 bytes, with all-ones reserved value (4 or 8 bytes of 0xFF depending on the length of metadata) for the metadata corresponding to instructions it cannot process. An INT hop that cannot insert Per-hop Metadata Length * 4 bytes must skip INT processing altogether and not insert any metadata in the packet.

- **Remaining Hop Count (8b):** The remaining number of hops that are allowed to add their metadata to the packet.

  - Upon creation of an INT metadata header, the INT Source must set this value to the maximum number of hops that are allowed to add metadata instance(s) to the packet. Each INT node on the path, including the INT Source as well as INT Transit Hops, must decrement the *Remaining Hop Count* if and when it pushes its local metadata onto the stack.
  - When a packet is received with the *Remaining Hop Count* equal to 0, the node must ignore the INT instructions in the *Instruction Bitmap* and *DS Instruction*, pushing no new metadata onto the stack, and the node must set the E bit.

- **Instruction Bitmap:** Each bit corresponds to a specific standard metadata as specified in Section 3.

  - bit0 (MSB): Node ID
  - bit1: Level 1 Ingress Interface ID (16 bits) + Egress Interface ID (16 bits)
  - bit2: Hop latency
  - bit3: Queue ID (8 bits) + Queue occupancy (24 bits)
  - bit4: Ingress timestamp (8 bytes)
  - bit5: Egress timestamp (8 bytes)
  - bit6: Level 2 Ingress Interface ID + Egress Interface ID (4 bytes each)
  - bit7: Egress interface Tx utilization
  - bit8: Buffer ID (8 bits) + Buffer occupancy (24 bits)
  - bit15: Checksum Complement
  - The remaining bits are reserved.

  Semantics of Queue occupancy and Buffer occupancy is the default semantics of those two metadata. Additional semantics as needed for different implementation can be defined in the metadata semantics YANG model.

  Details of the metadata semantics YANG model can be accessed at the link below:
  https://github.com/p4lang/p4-applications/blob/master/telemetry/code/models/p4-dtel-metadata-semantics.yang

  Bits 0 - 14 are Baseline INT Instructions. Each instruction bit that is set requests 4 bytes of metadata to be inserted at each hop, except for bits 4-6, each requires 8 bytes of metadata. Per-hop metadata length (*Hop ML*) is set accordingly at the INT source.

- **Domain Specific ID (16b):** The unique ID of the INT Domain. If the *Domain Specific ID* matches any Domain ID known to this node, then additional processing of the Domain Specific Flags (*DS Flags*) and Domain Specific Instruction (*DS Instruction*) is required.

- **DS Instruction (16b):** Instruction bitmap specific to the INT domain identified by the *Domain Specific ID*. Each bit that is set requests that Domain Specific Metadata be appended to the Baseline Metadata before the Checksum Complement is inserted.

  Some instruction bits can be defined as 'source-only' metadata by the INT domain. Those metadata will be inserted only by the INT source, not by INT transit or sink nodes. In a

sense, 'source-only' bits do not serve as instructions for downstream INT nodes to follow. The INT source sets the bits to indicate which source-only DS metadata it's adding such that the monitoring system (or any consumer of the metadata) knows how to parse and use the data.

The amount of Domain Specific Metadata added by each hop must be a multiple of 4 bytes, determined from the *DS Instruction*. In case of INT transit, the amount must be consistent with the per-hop metadata length (*Hop ML*) set by the INT source. The amount of Domain Specific Metadata added by the INT source can be larger than the amount added by a transit hop and the delta must match the total size of 'source-only' Domain Specific Metadata. Although the delta is excluded in *Hop ML*, it must be counted in the INT length field of the INT shim header.

If the *Domain Specific ID* does not match any Domain ID known to this node, then the transit node is required to either:

  - Pad the node's INT Metadata stack with the special all-ones reserved value for a Domain Specific Metadata length, calculated by subtracting from the *Hop ML* a length computed from all bits in the 16-bit *Instruction Bitmap*, or
  - Skip INT processing altogether and not insert any metadata into the packet.

- Each INT Transit node along the path that supports INT adds its own metadata values as specified in the *Instruction Bitmap* and *DS Instruction* immediately after the INT metadata header.

  - When adding a new metadata, each node must prepend its metadata in front of the metadata that are already added by the upstream nodes. This is similar to the push operation on a stack data structure. Hence, the most recently added metadata appears at the top of the stack. The node must add metadata in the order of bits set in the instruction bitmap.
  - If a node is unable to provide a metadata value specified in the instruction bitmap because its value is not available, it must add a special all-ones reserved value indicating "invalid" (4 or 8 bytes of 0xFF depending on metadata length).
  - If a node cannot add all the metadata required by the instruction bitmap (irrespective of the availability of the metadata values that are asked for), it must skip processing that particular INT packet entirely. This ensures that each INT Transit node adds either zero bytes or *Hop ML* * 4 bytes to the packet.
  - Reserved bits in the *Instruction Bitmap* are to be handled similarly. If an INT transit hop receives a reserved bit set in the *Instruction Bitmap* (e.g. set by a INT source that is running a newer version), the transit hop must either add corresponding metadata filled with the reserved value 0xFFFFFFFF or must not add any INT metadata to the packet. This means that an instruction bit marked reserved in this specification may be used for a 4B metadata in a subsequent minor version while still being backward compatible with this specification. However, an instruction bit marked reserved in this specification may be used for a 8B metadata only in the next major version, breaking backward compatibility and requiring all INT nodes to be upgraded to the new major version. For example a version 2.0 INT node cannot operate alongside version 3.0 INT nodes if a new 8B metadata is introduced in version 3.0, as the version 2.0 INT node could insert 0xFFFFFFFF reserved value for a 8B metadata field.

  - If an INT transit hop does not add metadata to a packet due to any of the above reasons, it must not decrement the *Remaining Hop Count* in the INT metadata header.

- Summary of the field usage

  - The INT Source must set the following fields:

    * *Ver*, *D*, *M*, *Hop ML*, *Remaining Hop Count*, and *Instruction Bitmap*.
    * INT Source should set all reserved bits to zero.
    * INT Source may set the Domain-specific fields.

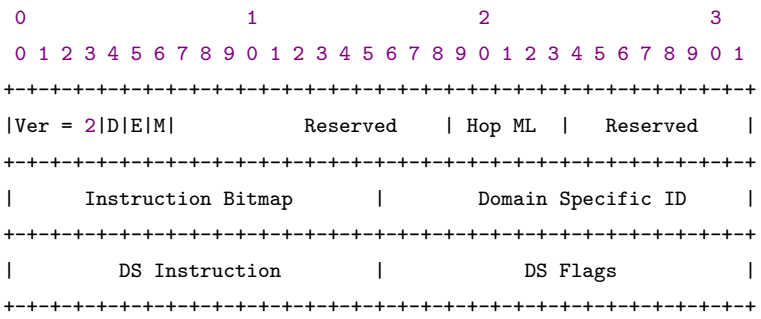
  - Intermediate transit nodes can set the following fields:

    * *E*, *M*, *Remaining Hop Count*, and *DS Flags* fields.
    * Intermediate transit nodes must not modify the *Hop ML*, *Instruction Bitmap*, *Domain Specific ID* and *DS Instruction* in the INT-MD header.

- The length (in bytes) of the INT metadata stack must always be a multiple of (*Hop ML* * 4), plus the size of 'source-only' Domain Specific Metadata if added by the source. The total stack length can be determined by subtracting the total INT fixed header sizes (12 bytes) from (shim header length * 4).

5.8. INT-MX Header Format

In this section, we define the format of the INT-MX header.
INT-MX Header:



- The INT-MX header is 12 bytes long. Each metadata requested in the INT-MX Header instruction is either 4 bytes or 8 bytes in length. Each INT node in the forwarding path will send the requested metadata to the monitoring system in the Telemetry Report.

  Details of the metadata semantics and format of the Telemetry Report can be accessed at the link below:
  https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v2.0.pdf

- **Ver (4b):** INT-MX header version. Should be 2 for this version.

- **D (1b):** Discard.

  - INT Sink must Discard the packet after sending the metadata requested in the INT-MX header to the monitoring system.

- **E (1b):** Max Hop Count exceeded.
  - Not applicable in INT-MX mode. Must be set to 0 by the INT Source.
- **M (1b):** MTU exceeded.
  - Not applicable in INT-MX mode. Must be set to 0 by the INT Source.
- **R (12b):** Reserved bits.
  - Should be set to 0 by INT Source and ignored by other nodes.
- **Hop ML (5b):** Hop Metadata Length. This is the length of metadata including the Domain Specific Metadata in 4-Byte words to be sent to monitoring systems at each INT transit hop.
  - *Hop ML* is set by the INT source. Transit hops and INT sink node may refer to it. If an INT domain uses 'source-only' Domain Specific Metadata, defined below, the length of the source-only Domain Specific Metadata is *excluded* from the *Hop ML*.
  - The largest value of *Hop ML* for baseline and domain specific metadata is 31.
  - An INT node may be limited in the maximum number of instructions it can process and/or maximum length of metadata it can gather for each data packet. An INT hop that cannot process all instructions has the following alternatives:
    * Send to the monitoring system *Hop ML* * 4 bytes, with all-ones reserved value (4 or 8 bytes of 0xFF depending on the length of metadata) for the metadata corresponding to instructions it cannot process, or
    * Send to the monitoring system the metadata it can process, update the *RepMdBits*, *DSMdBits* and *MD Length* fields appropriately in the Telemetry Report.
- **R (8b):** Reserved bits.
  - Should be set to 0 by INT Source and ignored by other nodes.
- **Instruction Bitmap:** Each bit corresponds to a specific standard metadata as specified in Section 3.
  - bit0 (MSB): Node ID
  - bit1: Level 1 Ingress Interface ID (16 bits) + Egress Interface ID (16 bits)
  - bit2: Hop latency
  - bit3: Queue ID (8 bits) + Queue occupancy (24 bits)
  - bit4: Ingress timestamp (8 bytes)
  - bit5: Egress timestamp (8 bytes)
  - bit6: Level 2 Ingress Interface ID + Egress Interface ID (4 bytes each)
  - bit7: Egress interface Tx utilization
  - bit8: Buffer ID (8 bits) + Buffer occupancy (24 bits)
  - The remaining bits are reserved.

Semantics of Queue occupancy and Buffer occupancy is the default semantics of those two metadata. Additional semantics as needed for different implementation can be defined in the metadata semantics YANG model.

Details of the metadata semantics YANG model can be accessed at the link below: https://github.com/p4lang/p4-applications/blob/master/telemetry/code/models/p4-dtel-metadata-semantics.yang

Bits 0 - 14 are Baseline INT Instructions. Each instruction bit that is set requests 4 bytes of metadata be sent to monitoring system at each hop, except for bits 4-6, each requires 8 bytes of metadata.

- **Domain Specific ID (16b):** The unique ID of the INT Domain. If the *Domain Specific ID* matches any Domain ID known to this node, then additional processing of the Domain Specific Flags (*DS Flags*) and Domain Specific Instruction (*DS Instruction*) is required.
- **DS Instruction (16b):** Instruction bitmap specific to the INT domain identified by the *Domain Specific ID*. Each bit that is set requests that metadata be sent to the monitoring system.

  Some metadata can be defined as 'source-only' metadata by the INT domain. Those metadata will be sent only by the INT source, not by INT transit or sink nodes. In a sense, 'source-only' bits do not serve as instructions for downstream INT nodes to follow. In INT-MX mode, the source node must not set the bits corresponding to "source-only" metadata in the *DS Instruction* bitmap. However, the source node must ensure that it sets the bits corresponding to "source-only" metadata in *DSMdBits* bitmap of the Telemetry Report. This is needed for the monitoring system (or any consumer of the metadata) to know how to parse and use the data.

  The amount of Domain Specific Metadata sent by each hop must be a multiple of 4 bytes, either determined from the *DS Instruction* or consistent with the per-hop metadata length (*Hop ML*) set by the INT source (see *Hop ML* above).

  If the *Domain Specific ID* does not match any Domain ID known to this node, then the transit node is required to either:

  - Send the metadata corresponding to *Instruction Bitmap* and ensure that *DSMdbits* is not set in the Telemetry Report and *MD Length* field reflects the length of the metadata sent in the Telemetry Report, or
  - Send the metadata corresponding to *Instruction Bitmap* and pad the node's INT Metadata with the special all-ones reserved value for a Domain Specific Metadata length, calculated by subtracting from the *Hop ML* a length computed from all bits in the 16-bit *Instruction Bitmap*, or
  - Skip INT processing altogether and not send any metadata the monitoring systems.

  Details of the metadata semantics and format of the Telemetry Report can be accessed at the link below:

  https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v2.0.pdf

- Each INT Transit node along the path that supports INT-MX sends its own metadata values as specified in the *Instruction Bitmap* and *DS Instruction*.
  - If a node is unable to provide a metadata value specified in the instruction bitmap because its value is not available, it must either:
    * ensure that the corresponding bit in *RepMdbits* and/or *DSMdbits* is not set in the Telemetry Report, or
    * send the all-ones reserved value (4 or 8 bytes of 0xFF depending on the length of metadata).

    Details of the metadata semantics and format of the Telemetry Report can be accessed at the link below:
    https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v2.0.pdf
  - Reserved bits in the instruction bitmap are to be handled as described. If an INT transit hop receives a reserved bit set in the instruction bitmap (e.g. set by a INT source that is running a newer version), the transit hop must either:
    * ensure that the corresponding reserved bits in *RepMdbits* and/or *DSMdbits* are not set in the Telemetry Report sent to the monitoring system, or
    * for *RepMdBits*, send the all-ones reserved value (4 bytes of 0xFF); for *DSMdbits*, pad with the special all-ones reserved value in lieu of reserved DS Metadata, out to *MD Length*.

    Details of the metadata semantics and format of the Telemetry Report can be accessed at the link below:
    https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v2.0.pdf
- Summary of the field usage
  - The INT Source must set the following fields:
    * *Ver*, *D* and *Instruction Bitmap*.
    * INT Source should set all reserved bits to zero.
    * INT Source may set the Domain-specific fields.
  - Intermediate transit nodes may set bits in the *DS Flags* field. Intermediate transit nodes must not modify the *Hop ML*, *Instruction Bitmap*, *Domain Specific ID* and *DS Instruction* in the INT-MX header.
- The length (in bytes) of the INT metadata sent to the monitoring system must always be a multiple of 4B.

6. Examples

This section shows example INT Headers with two hosts (Host1 and Host2), communicating over a network path composed of three network switches (Switch1, Switch2 and Switch3) as shown below.

```
==> packet P travels from Host1 to Host2 ==>
Host1 --------> Switch1 ---------> Switch2 ---------> Switch3 --------> Host2
```

Detailed assumptions made for this example are as follows

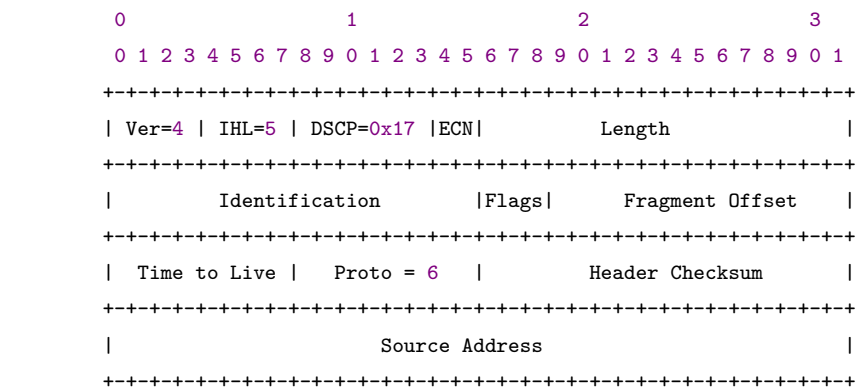
- INT source requests each INT hop to insert node ID and queue occupancy (For the sake of illustration we only consider node ID and queue occupancy being inserted at each hop. Queue IDs are typically defined per port, hence in a real use-case queue occupancy is likely to be collected along with egress interface ID)
- There are three INT nodes (hops) on the path, and all the nodes expose both metadata (node ID and queue occupancy).
- The maximum number of hops (network diameter) is 8.
- The values of INT metadata header fields in this example are as follows:
  - *Ver* = 2
  - $D = 0$ (Packet is not a clone/copy, hence the Sink must not Discard)
  - $E = 0$ (Max Hop Count not exceeded)
  - $M = 0$ (MTU not exceeded at any node)
  - Per-hop Metadata Length = 2 (for node id & queue occupancy)
  - *Remaining Hop Count* starts at 8, decremented by 1 at each hop that inserts INT metadata

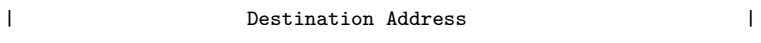
6.1. Example with INT-MD over TCP

We consider a scenario where host1 sends a TCP packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It adds INT-MD headers and its own metadata in the packet. Switch2 prepends its metadata. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes INT-MD headers before forwarding the packet to host2.

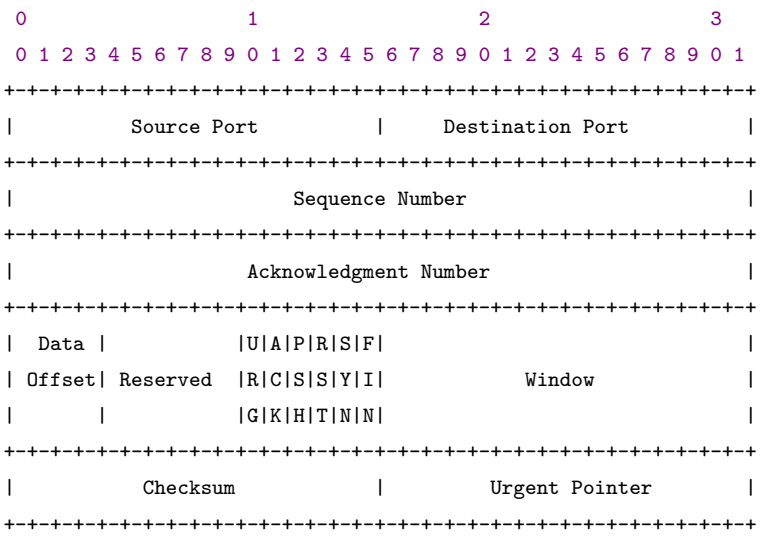
Below is the packet received by INT sink Switch3, starting from the IPv4 header. We use the value of 0x17 for IPv4.DSCP to indicate the existence of INT headers.

IP Header:

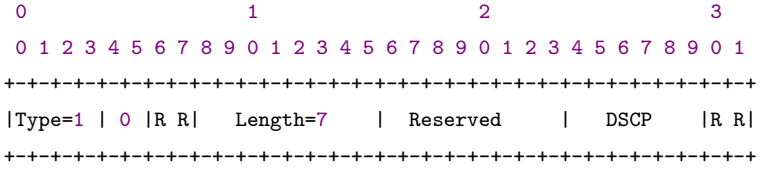




TCP Header:



INT Shim Header for TCP/UDP, INT type is INT-MD (1) and NPT (Next Protocol Type) is zero:



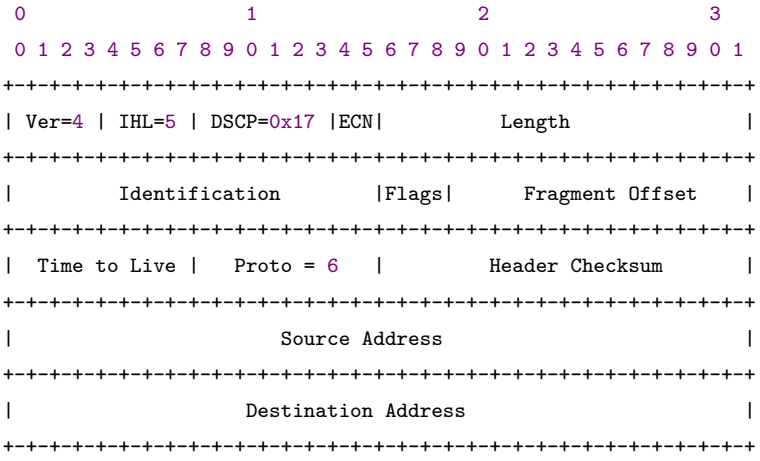
INT-MD Metadata Header and Metadata Stack, followed by TCP payload:

6.2. Example with INT-MX over TCP

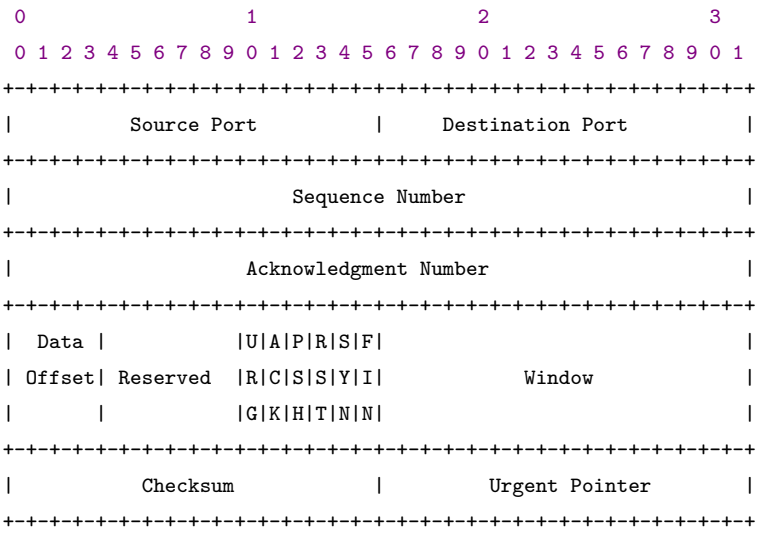
We consider a scenario where host1 sends a TCP packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It adds INT-MX header in the packet. Switch2 processes the INT-MX header. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes INT-MX header before forwarding the packet to host2.

Below is the packet received by INT sink Switch3, starting from the IPv4 header. We use the value of 0x17 for IPv4.DSCP to indicate the existence of INT headers.

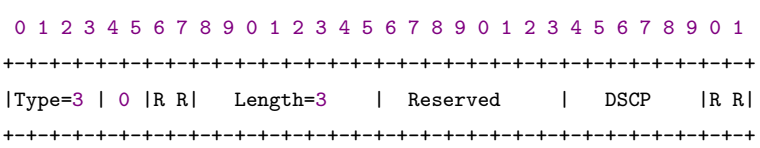
IP Header:



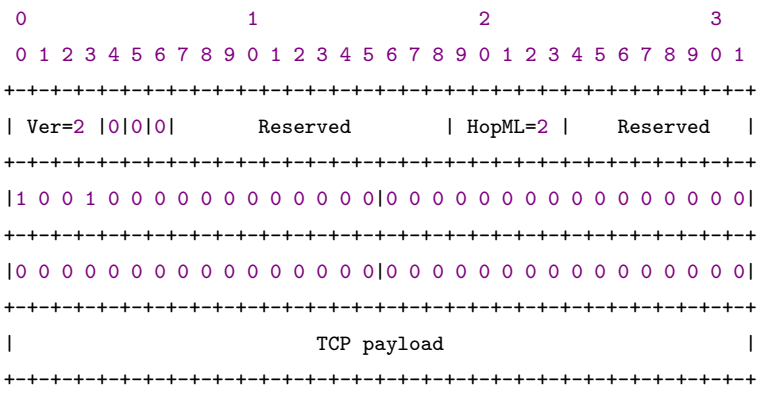
TCP Header:



INT Shim Header for TCP/UDP, INT type is INT-MX (3) and NPT (Next Protocol Type) is zero:



INT-MX Header, followed by TCP payload:

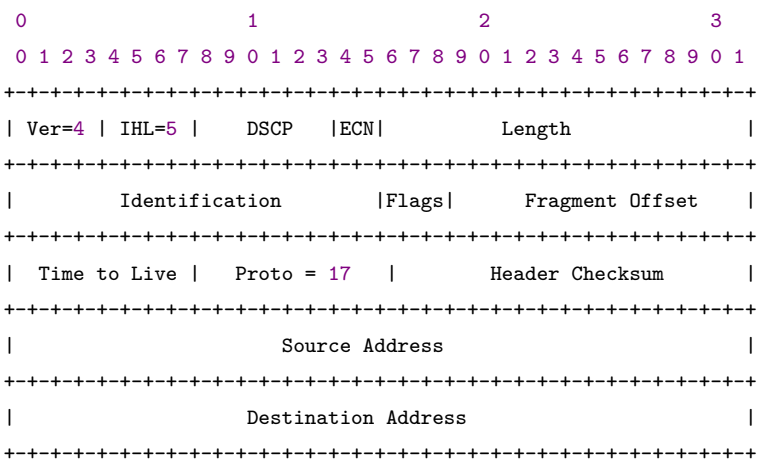


6.3. Example with new UDP header and INT-MD inserted before TCP

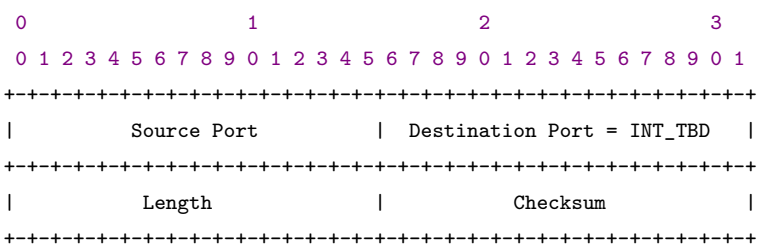
As before we consider a scenario where host1 sends a TCP packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It adds a new UDP header, INT-MD headers and its own metadata in the packet. Switch2 prepends its metadata. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes the UDP and INT-MD headers before forwarding the packet to host2.

Below is the packet received by INT sink Switch3, starting from the IPv4 header. We use INT_TBD for UDP.Destination_Port to indicate the existence of INT headers.

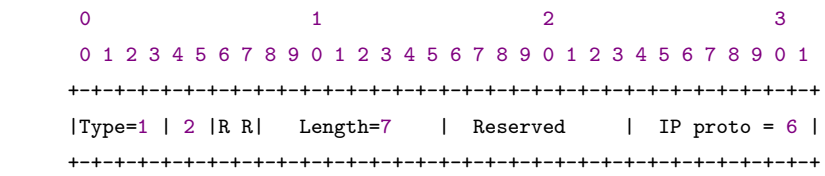
IP Header:



UDP Header:



INT Shim Header for UDP, INT type is INT-MD (1) and NPT (Next Protocol Type) is 2 indicating another L4 header follows INT. IP proto is 6 to indicate that TCP follows INT:



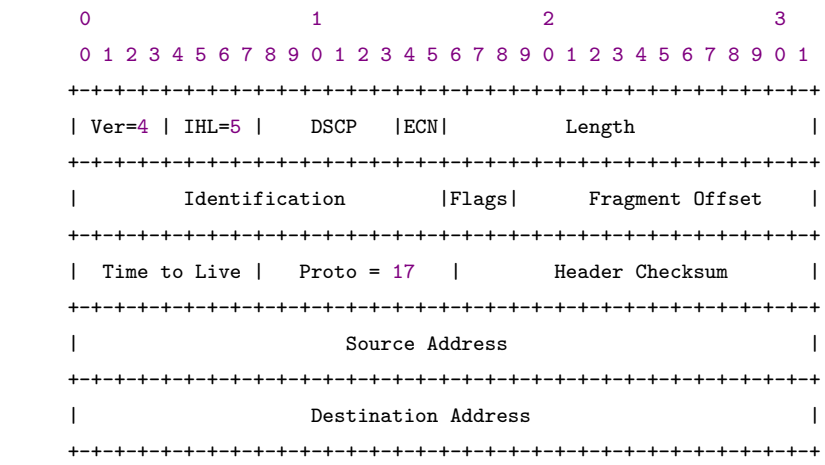
INT-MD Metadata Header and Metadata Stack, followed by TCP header:

6.4. Example with new UDP header and INT-MX inserted before TCP

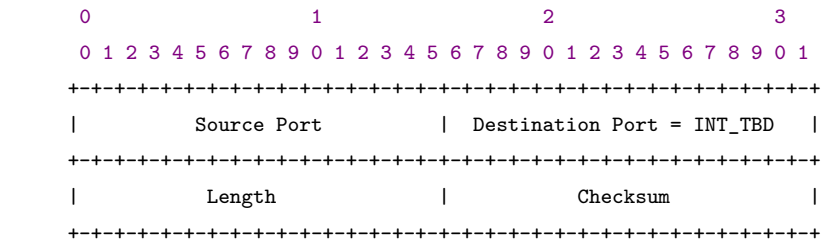
As before we consider a scenario where host1 sends a TCP packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It adds a new UDP header and a new INT-MX header in the packet. Switch2 processes the INT-MX Header. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes the UDP and INT-MX headers before forwarding the packet to host2.

Below is the packet received by INT sink Switch3, starting from the IPv4 header. We use INT_TBD for UDP.Destination_Port to indicate the existence of INT headers.

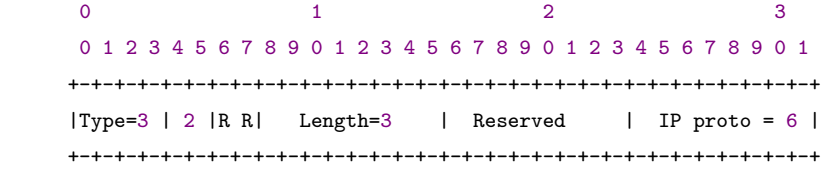
IP Header:



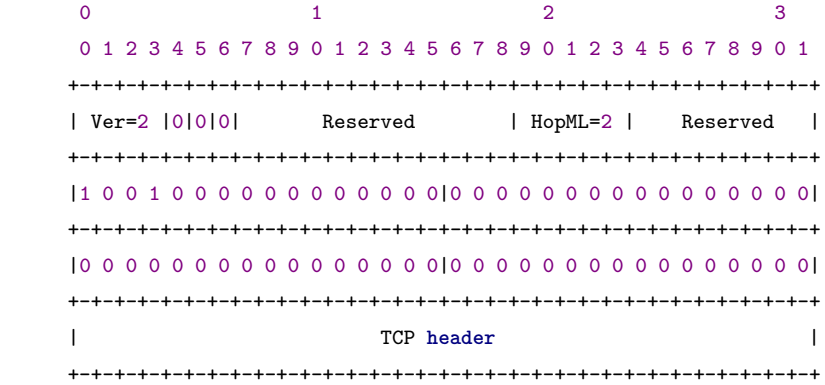
UDP Header:



INT Shim Header for UDP, INT type is INT-MX (3) and NPT (Next Protocol Type) is 2 indicating another L4 header follows INT. IP proto is 6 to indicate that TCP follows INT:



INT-MX Header, followed by TCP header:

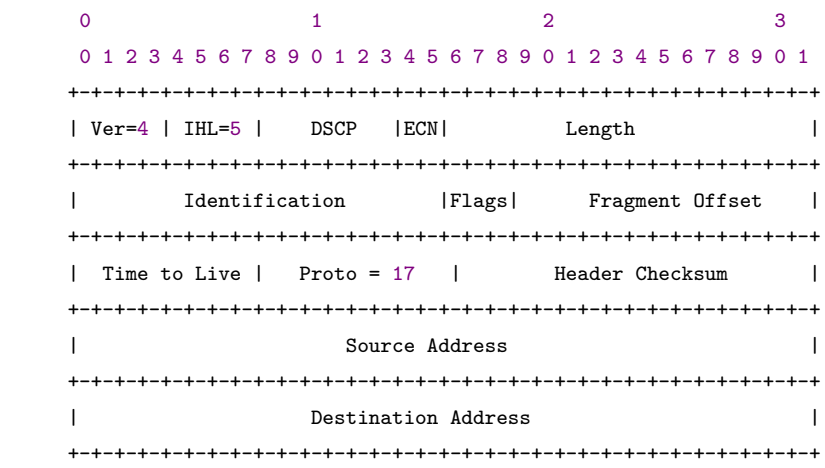


6.5. Example with INT-MD in-between UDP header and UDP payload

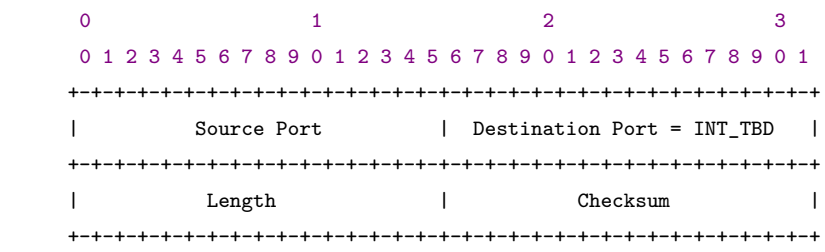
In this scenario host1 sends a UDP packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It alters the UDP destination port to INT_TBD, inserts INT-MD headers before the UDP payload. Switch2 prepends its metadata. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes the INT-MD headers and restores the original UDP destination port before forwarding the packet to host2.

Below is the packet received by INT sink Switch3, starting from the IPv4 header. We use INT_TBD for UDP.Destination_Port to indicate the existence of INT headers.

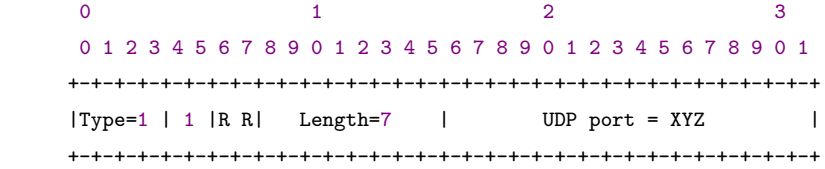
IP Header:



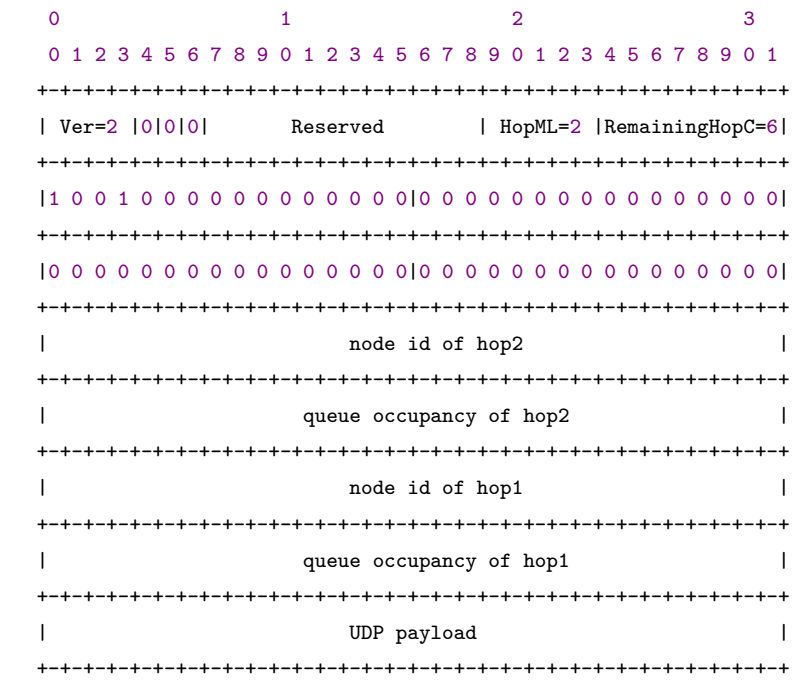
UDP Header:



INT Shim Header for UDP, INT type is INT-MD (1) and NPT (Next Protocol Type) is 1 indicating UDP payload follows the INT. The original port number XYZ is stored in the shim header:



INT-MD Metadata Header and Metadata Stack, followed by UDP payload:

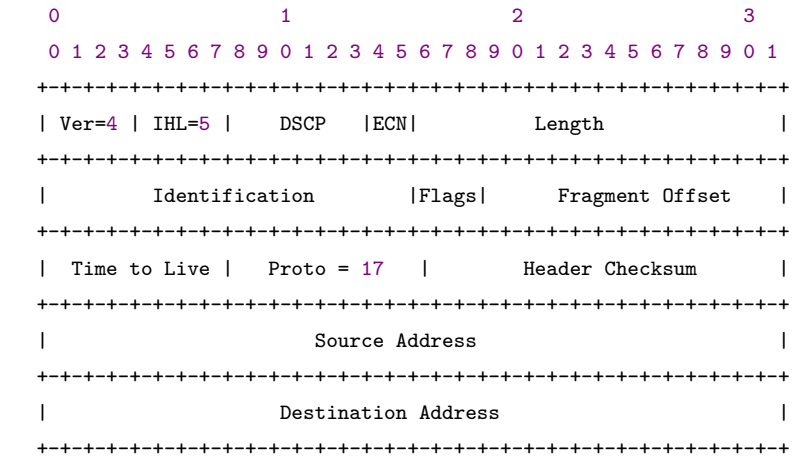


6.6. Example with INT-MX in-between UDP header and UDP payload

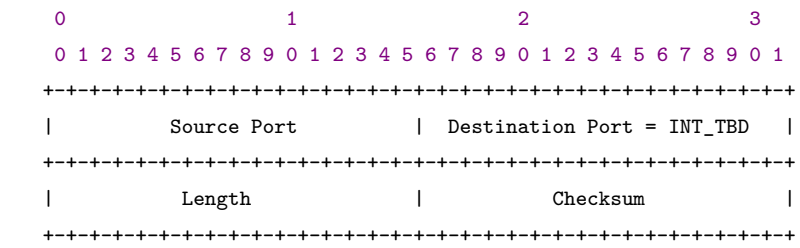
In this scenario host1 sends a UDP packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It alters the UDP destination port to INT_TBD, inserts INT-MX header before the UDP payload. Switch2 processes the INT-MX header. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes the INT-MX headers and restores the original UDP destination port before forwarding the packet to host2.

Below is the packet received by INT sink Switch3, starting from the IPv4 header. We use INT_TBD for UDP.Destination_Port to indicate the existence of INT headers.

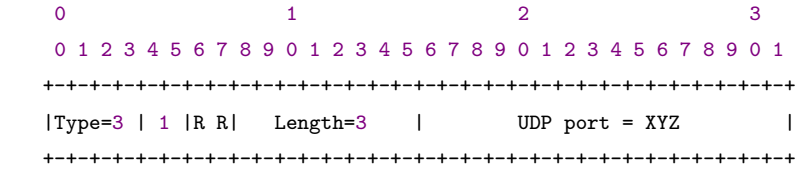
IP Header:



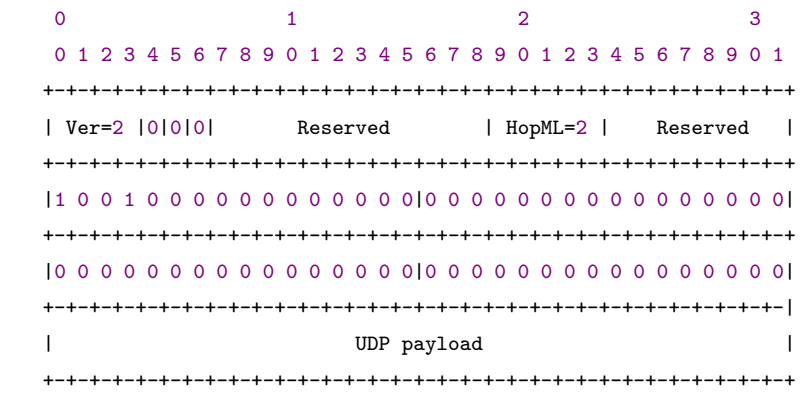
UDP Header:



INT Shim Header for UDP, INT type is INT-MX (3) and NPT (Next Protocol Type) is 1 indicating UDP payload follows the INT. The original port number XYZ is stored in the shim header:



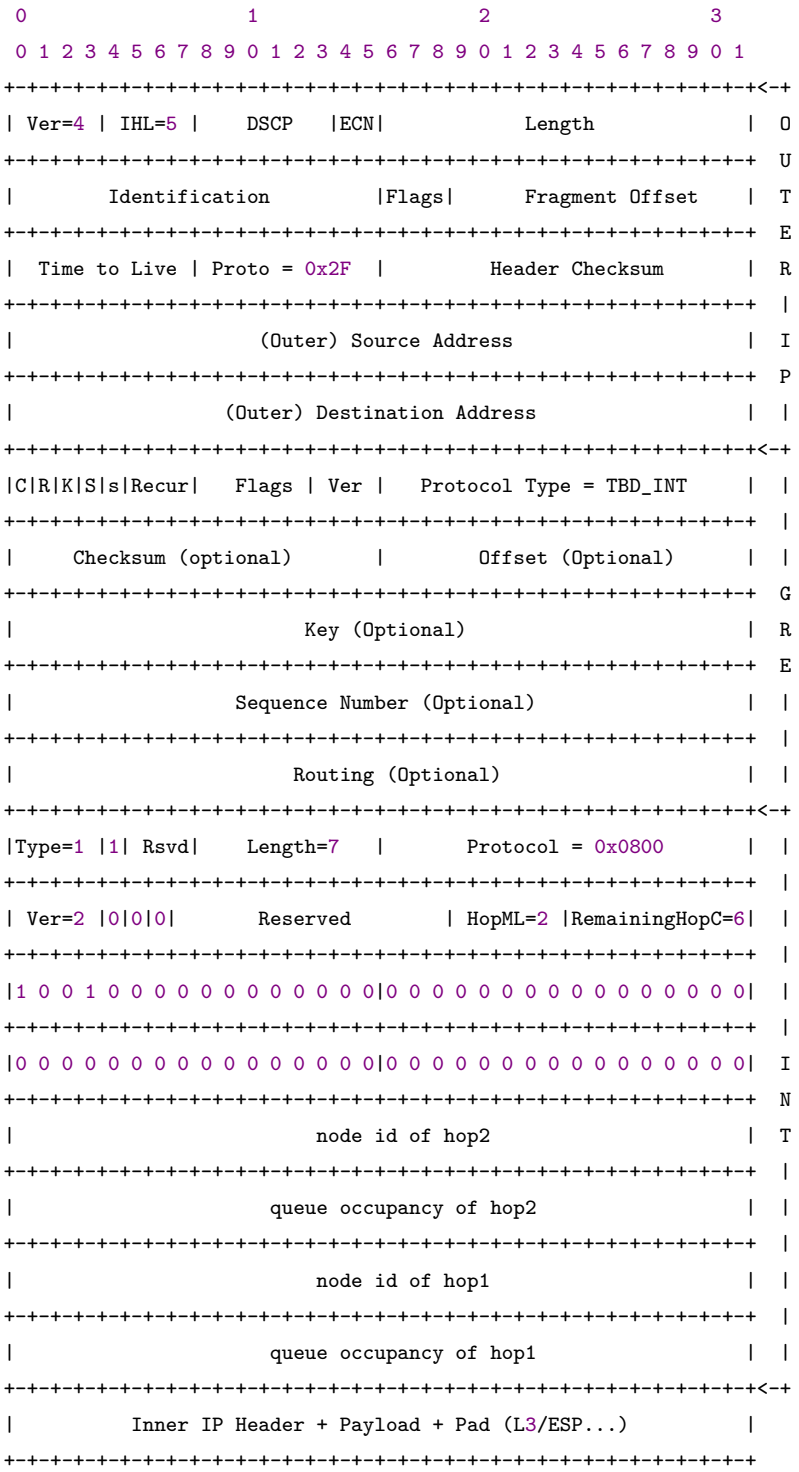
INT-MX Header, followed by UDP payload:



6.7. Example with INT-MD over IPv4/GRE (Original packet IPv4)

In this scenario host1 sends an IPv4 packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It does IPv4/GRE encapsulation and inserts INT-MD headers before the inner (original) IPv4. Switch2 prepends its metadata. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes the INT-MD headers and decapsulates outer IPv4/GRE before forwarding the packet to host2.

Below is the packet received by INT sink Switch3, starting from the outer IPv4 header. The G bit of INT shim is set to 1 to indicate that GRE was inserted by the INT source.



6.8. Example with INT-MX over IPv4/GRE (Original packet IPv4)

In this scenario host1 sends an IPv4 packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It does IPv4/GRE encapsulation and inserts INT-MX header before the inner (original) IPv4. Switch2 processes the INT-MX header. Finally, the ToR switch of host2 (Switch3)

acts as the INT sink and removes the INT-MX header and decapsulates outer IPv4/GRE before forwarding the packet to host2.

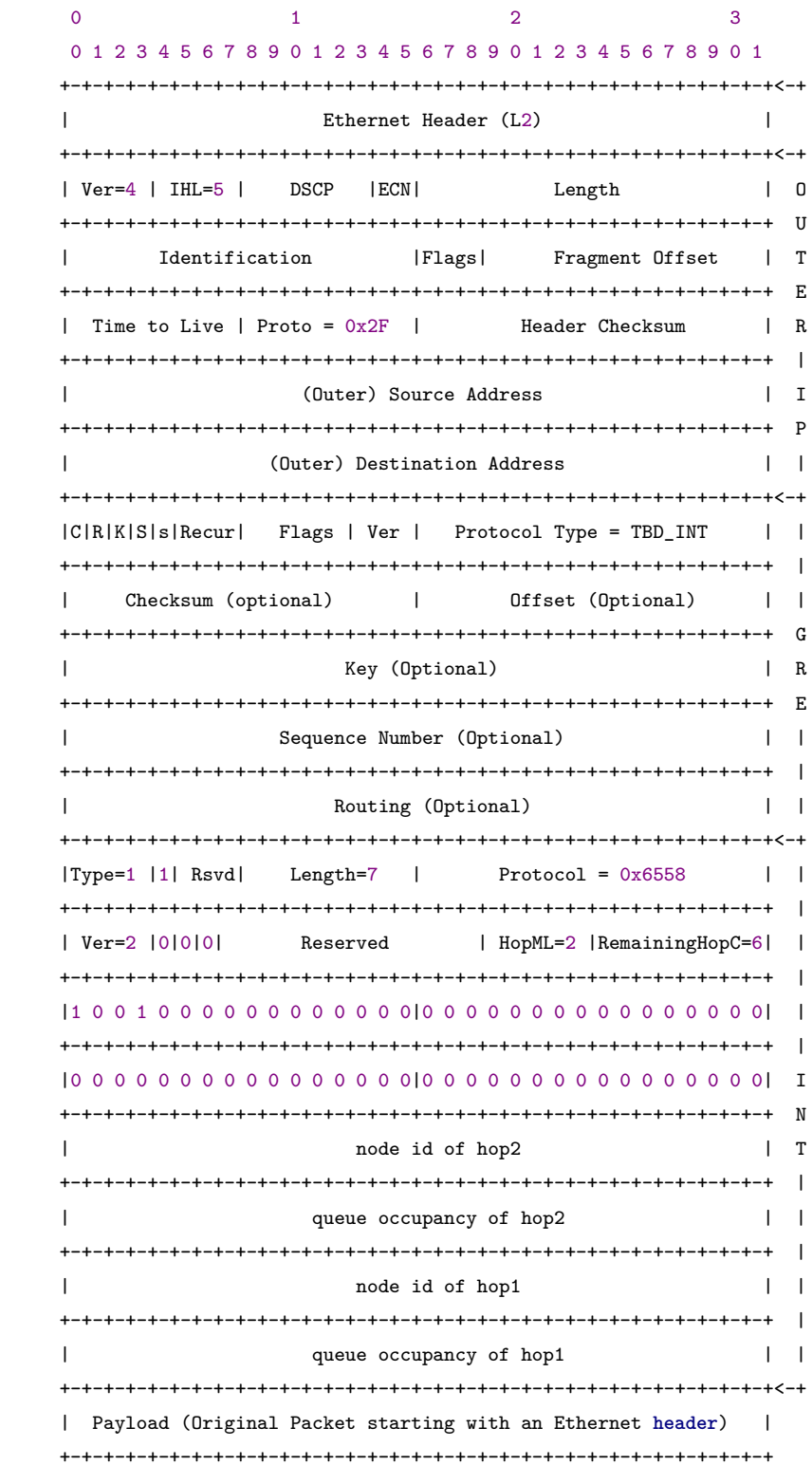
Below is the packet received by INT sink Switch3, starting from the outer IPv4 header. The G bit of INT shim is set to 1 to indicate that GRE was inserted by the INT source.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
| Ver=4 | IHL=5 |    DSCP   |ECN|            Length             |  O
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  U
|        Identification         |Flags|      Fragment Offset    |  T
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  E
|  Time to Live | Proto = 0x2F  |         Header Checksum       |  R
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                     (Outer) Source Address                    |  I
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  P
|                  (Outer) Destination Address                  |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
|C|R|K|S|s|Recur|   Flags | Ver |   Protocol Type = TBD_INT     |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|     Checksum (optional)       |        Offset (Optional)      |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  G
|                         Key (Optional)                        |  R
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  E
|                   Sequence Number (Optional)                  |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                       Routing (Optional)                      |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
|Type=3 |1| Rsvd|    Length=3   |       Protocol = 0x0800       |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
| Ver=2 |0|0|0|       Reserved        | HopML=2 |    Reserved   |  I
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  N
|1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|  T
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
|          Inner IP Header + Payload + Pad (L3/ESP...)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

6.9. Example with INT-MD over IPv4/GRE (Original packet CE or IP)

In this scenario host1 sends an IPv4 packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It adds an Ethernet (L2) header, does IPv4/GRE encapsulation and inserts INT-MD headers before the inner (original) packet that starts with an Ethernet header. Switch2 prepends its metadata. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes the INT-MD headers and decapsulates outer L2 header, outer IPv4/GRE before forwarding the packet to host2.

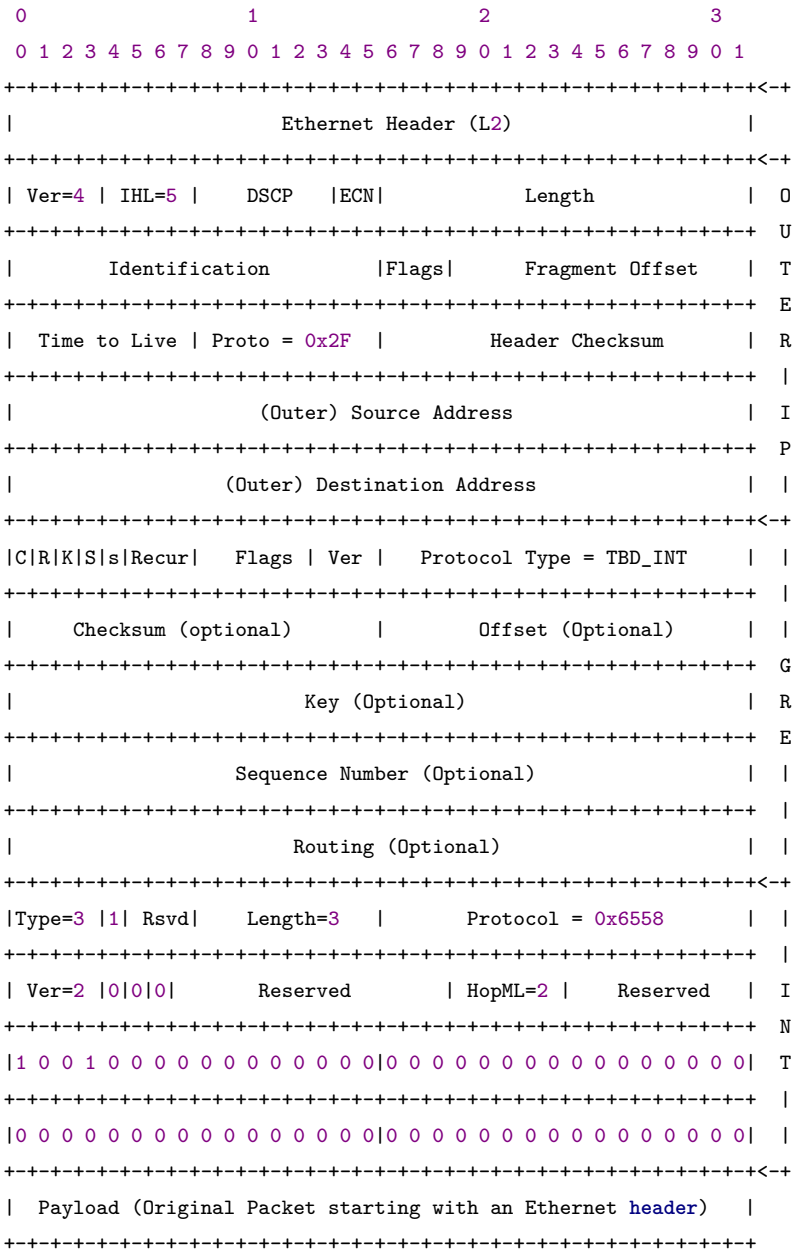
Below is the packet received by INT sink Switch3, starting from the outer Ethernet header. The G bit of INT shim is set to 1 to indicate that GRE was inserted by the INT source.



6.10. Example with INT-MX over IPv4/GRE (Original packet CE or IP)

In this scenario host1 sends an IPv4 packet to host2. The ToR switch of host1 (Switch1) acts as the INT source. It adds an Ethernet (L2) header, does IPv4/GRE encapsulation and inserts INT-MX header before the inner (original) packet that starts with an Ethernet header. Switch2 processes the INT-MX header. Finally, the ToR switch of host2 (Switch3) acts as the INT sink and removes the INT-MX header and decapsulates outer L2 header, outer IPv4/GRE before forwarding the packet to host2.

Below is the packet received by INT sink Switch3, starting from the outer Ethernet header. The G bit of INT shim is set to 1 to indicate that GRE was inserted by the INT source.

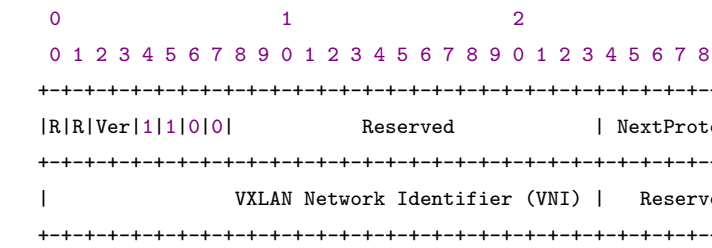


6.11. Example with INT-MD over VXLAN GPE

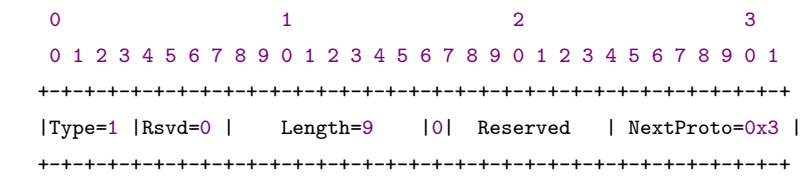
We now consider a scenario where Host1 and Host2 use VXLAN encapsulation. Host1 acts as VXLAN tunnel endpoint and INT source, inserts VXLAN and INT-MD headers with instruction bits corresponding to the network state to be reported at intermediate switches. In this example, Host1 itself does not insert any INT metadata. Intermediate switches parse through VXLAN header and populate the INT metadata. Host2 acts as INT sink and VXLAN tunnel endpoint, removes INT-MD and VXLAN headers.

The packet headers received at Host 2 are as follows, starting with the VXLAN GPE header (encapsulating ethernet, IP and UDP headers are not shown here):

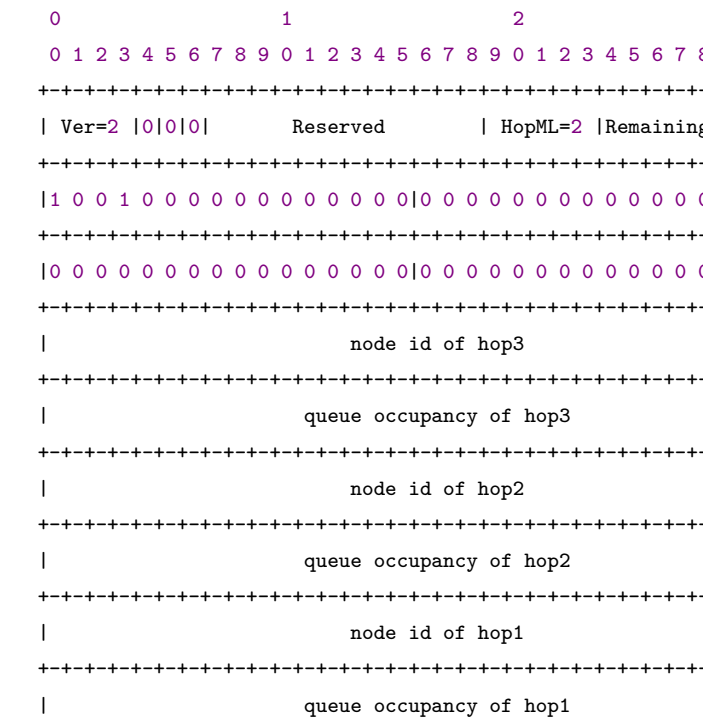
VXLAN GPE Header:



INT Shim Header for VXLAN-GPE, INT-MD type:



INT-MD Metadata Header and Metadata Stack:

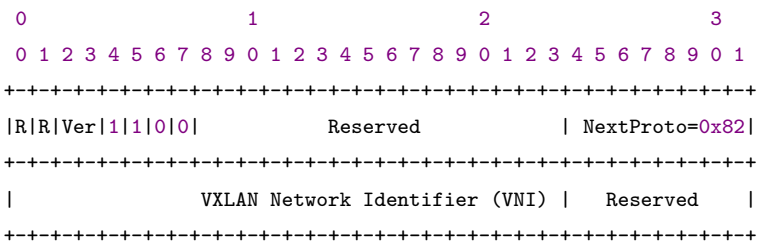


6.12. Example with INT-MX over VXLAN GPE

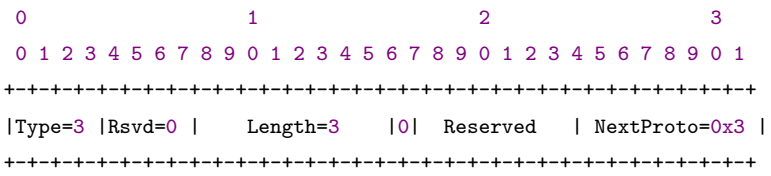
We now consider a scenario where Host1 and Host2 use VXLAN encapsulation. Host1 acts as VXLAN tunnel endpoint and INT source, inserts VXLAN and INT-MX header with instruction bits corresponding to the network state to be reported at intermediate switches. In this example, Host1 itself does not send any metadata to the monitoring system. Intermediate switches parse through VXLAN header, process the INT-MX header and send the INT metadata requested to the monitoring system. Host2 acts as INT sink and VXLAN tunnel endpoint, removes INT-MX and VXLAN headers.

The packet headers received at Host 2 are as follows, starting with the VXLAN GPE header (encapsulating ethernet, IP and UDP headers are not shown here):

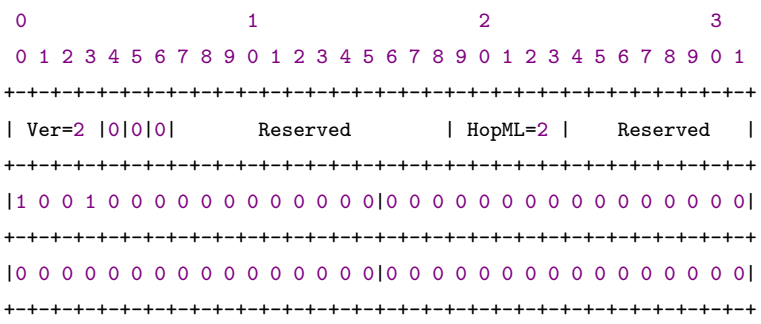
VXLAN GPE Header:



INT Shim Header for VXLAN-GPE, INT-MX type:



INT-MX Header:

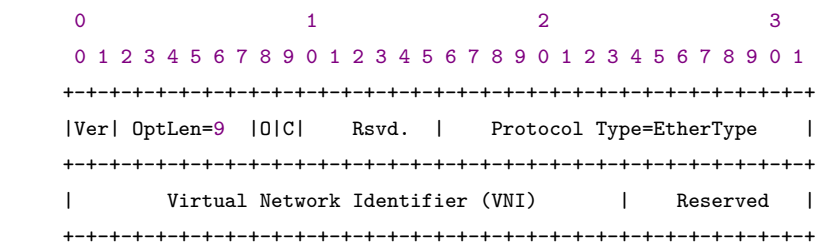


6.13. Example with INT-MD over Geneve

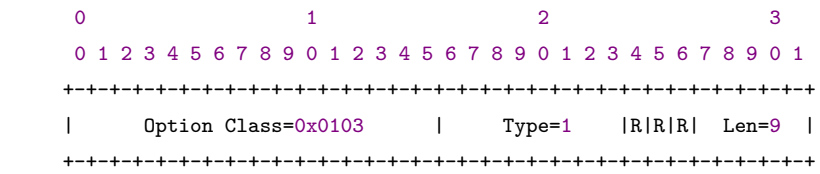
Finally, we consider a scenario where Host1 and Host2 use Geneve encapsulation. Host1 acts as Geneve tunnel endpoint and INT source, inserts Geneve and INT-MD headers with instruction bits corresponding to the network state to be reported at intermediate switches. In this example, Host1 itself does not insert any INT metadata. Intermediate switches parse through Geneve header and populate the INT metadata. Host2 acts as INT sink and Geneve tunnel endpoint, removes INT-MD and Geneve headers.

Following are the Geneve and INT Headers attached to the packet received by Host2.

Geneve Header:



Geneve Option for INT-MD type:



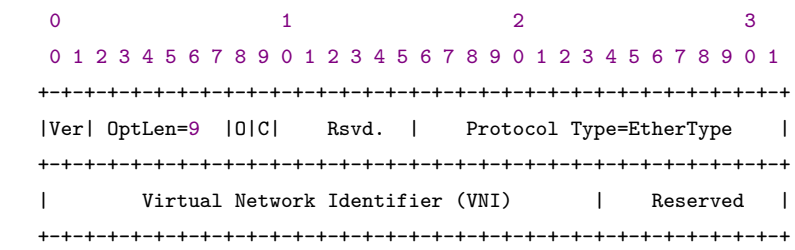
INT-MD Metadata Header and Metadata Stack:

6.14. Example with INT-MX over Geneve

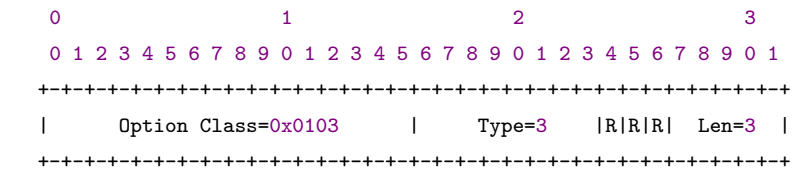
Finally, we consider a scenario where Host1 and Host2 use Geneve encapsulation. Host1 acts as Geneve tunnel endpoint and INT source, inserts Geneve and INT-MX headers with instruction bits corresponding to the network state to be reported at intermediate switches. In this example, Host1 does not send INT metadata to the monitoring system. Intermediate switches parse through Geneve header, process the INT-MX header and send the INT metadata requested to the monitoring system. Host2 acts as INT sink and Geneve tunnel endpoint, removes INT-MX and Geneve headers.

Following are the Geneve and INT Headers attached to the packet received by Host2.

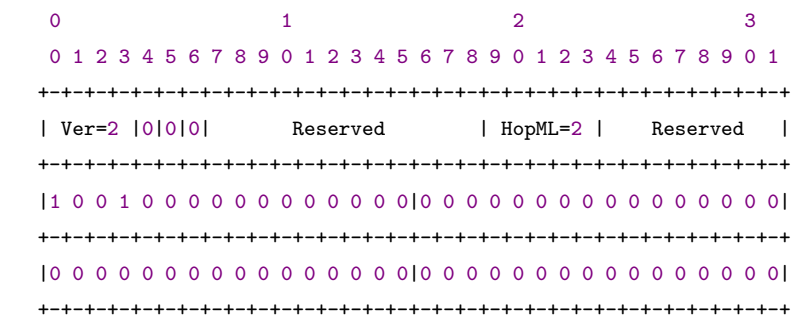
Geneve Header:



Geneve Option for INT-MX type:



INT-MX Header:



A. Appendix: An extensive (but not exhaustive) set of Metadata

Here we list a set of exemplary metadata that future versions of the spec may support as well as those are supported in the current spec.

A.1. Node-level

Node id

The unique ID of an INT node. This is generally administratively assigned. Node IDs must be unique within an INT domain.

Control plane state version number
Whenever a control-plane state changes (e.g., IP FIB update), the node's control plane can also update this version number in the data plane. INT packets may use these version numbers to determine which control-plane state was active at the time packets were forwarded.

A.2. Ingress

Ingress interface identifier
The interface on which the INT packet was received. A packet may be received on an arbitrary stack of interface constructs starting with a physical port. For example, a packet may be received on a physical port that belongs to a link aggregation port group, which in turn is part of a Layer 3 Switched Virtual Interface, and at Layer 3 the packet may be received in a tunnel. Although the entire interface stack may be monitored in theory, this specification allows for monitoring of up to two levels of ingress interface identifiers. The semantics of interface identifiers may differ across devices, each INT hop chooses the interface type it reports at each of the two levels.

Ingress timestamp
The device local time when the INT packet was received on the ***ingress*** physical or logical port.

Ingress interface RX pkt count
Total # of packets received so far (since device initialization or counter reset) on the ingress physical port or logical interface where the INT packet was received.

Ingress interface RX byte count
Total # of bytes received so far on the ingress physical port or logical interface where the INT packet was received.

Ingress interface RX drop count
Total # of packet drops occurred so far on the ingress physical port or logical interface where the INT packet was received.

Ingress interface RX utilization
Current utilization of the ingress physical port or logical interface where the INT packet was received. The exact mechanism (bin bucketing, moving average, etc.) is device specific and while the latter is clearly superior to the former, the INT framework leaves those decisions to device vendors.

A.3. Egress

Egress interface identifier
The interface on which the INT packet was sent out. A packet may be transmitted on an arbitrary stack of interface constructs ending at a physical port. For example, a packet may be transmitted on a tunnel, out of a Layer 3 Switched Virtual Interface, on a Link Aggregation Group, out of a particular physical port belonging to the Link Aggregation Group. Although the entire interface stack may be monitored in theory, this specification allows for monitoring of up to two levels of egress interface identifiers. The semantics of interface identifiers may differ across devices, each INT hop chooses the interface type it reports at each of the two levels.

Egress timestamp
The device local time when the INT packet was processed by the egress physical port or logical interface.

Egress interface TX pkt count
Total # of packets forwarded so far (since device initialization or counter reset) through the egress physical port or logical interface where the INT packet was also forwarded.

Egress interface TX byte count
Total # of bytes forwarded so far through the egress physical port or logical interface where the INT packet was forwarded.

Egress interface TX drop count
Total # of packet drops occurred so far on the egress physical port or logical interface where the INT packet was forwarded.

Egress interface TX utilization
Current utilization of the egress interface via which the INT packet was sent out.

A.4. Buffer Information

Queue id
The id of the queue the device used to serve the INT packet.

Instantaneous queue length
The instantaneous length (in bytes, cells, or packets) of the queue the INT packet has observed in the device while being forwarded. The units used need not be consistent across an INT domain, but care must be taken to ensure that there is a known, consistent mapping of {device, queue} values to their respective unit {packets, bytes, cells}.

Average queue length
The average length (in bytes, cells, or packets) of the queue via which the INT packet was served. The calculation mechanism of this value is device specific.

Queue drop count
Total # of packets dropped from the queue.

The metadata below are introduced to capture the buffer occupancy INT packet observes in the device while being forwarded. Use case is when buffer is shared between multiple queues.

Buffer id
The id of the buffer the device used to serve the INT packet.

Instantaneous buffer occupancy
The instantaneous value (in bytes, or cells) of the buffer occupancy the INT packet has observed in the device while being forwarded. The units used need not be consistent across an INT domain, but care must be taken to ensure that there is a known, consistent mapping of {device, buffer} values to their respective unit {bytes, cells}.

Average buffer occupancy
The average value (in bytes or cells) of the buffer occupancy that the INT packet was observed. The calculation mechanism of this value is device specific.

A.5. Miscellaneous

Checksum Complement
This field enables a Checksum-neutral update when INT is encapsulated over an L4 protocol that uses a Checksum field, such as TCP or UDP.

B. Domain Specific Examples

Here we list a set of exemplary domain specific implementations.

B.1. Example with INT-MD including source-only metadata

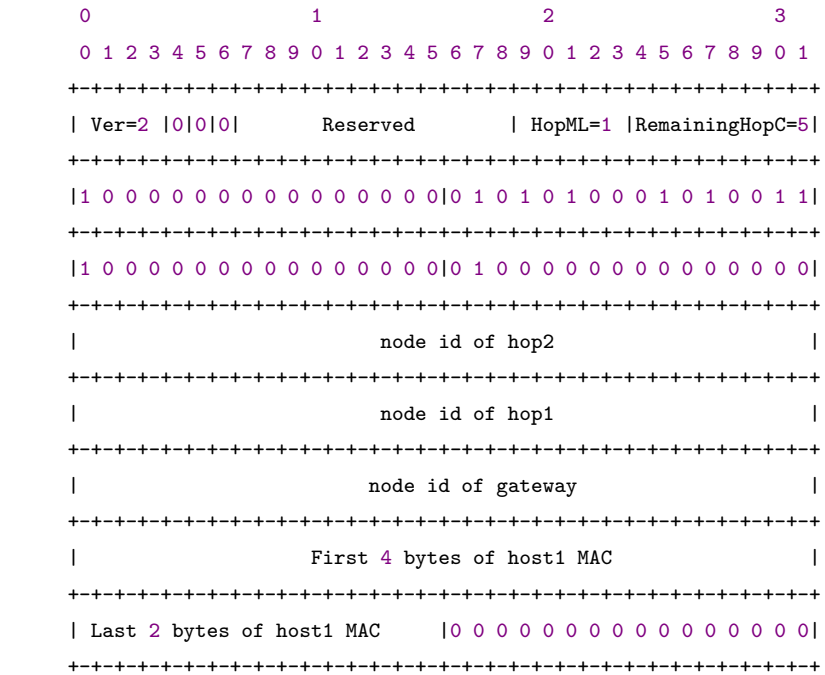
We consider a case where Host1 is on a wireless network behind a NAT and its identity can not be confirmed by port location. Host1 does not support INT. The gateway acts at the INT source, inserting the header and the MAC address of Host1. Intermediate switches populate the INT metadata while preserving the source-only domain specific metadata. This example can work with any of the above encapsulation methods such as UDP encapsulation.

The MAC address from host1 is copied from the source address in the Ethernet frame. The MAC is a 6 byte id. Since the INT metadata header is measured in blocks of 4 bytes, the last 2 bytes are reserved. For example if a source device's MAC address is "a6:1a:f6:b1:64:7d", the source-only INT metadata would be 0xA61AF6B1647d0000.

Following is the INT-MD Header attached to the packet transmitted by the hop2 switch.

Details of the Transparent Security domain specific model can be accessed at the link below: https://github.com/cablelabs/transparent-security/blob/master/docs/int_header/INT_header.md

INT Metadata Header and Metadata Stack. The *Domain Specific ID* is 0x5453 ('TS' in ascii), bit 0 is set in the domain bitmap to indicate the 16 bit source-only device source and bit 1 is set in the *DS Flags* to indicate that this was set by the gateway:



C. Acknowledgements

We thank the following individuals for their contributions to the design, specification and implementation of this spec.

- Daniel Alvarez
- Parag Bhide
- Dennis Cai
- Dan Daly
- Bruce Davie
- Ed Doe
- Senthil Ganesan
- Anoop Ghanwani
- Mukesh Hira
- Hugh Holbrook
- Raja Jayakumar
- Changhoon Kim
- Jeongkeun Lee
- Randy Levensalor
- Tal Mizrahi
- Masoud Moshref
- Michael Orr
- Heidi Ou
- Ramesh Sivakolundu
- Mickey Spiegel
- Bapi Vinnakota

D. Change log

- 2015-09-28
  - Initial release
- 2016-06-19
  - Updated section 5.6.3, the Length field definition of VXLAN GPE shim header, to be consistent with the example in section 6.
- 2017-10-17
  - Introduced INT over TCP/UDP (section 5.6.2 and new example)
  - Removed BOS (Bottom-Of-Stack) bit at each 4B metadata, from the header definition and examples
  - Updated the INT instruction bitmap and the meaning of a few instructions (section 5.7)
  - Moved the INT transit P4 program from Appendix to the main section. Re-wrote the program in p4_16.
- 2017-12-11
  - Increaed the size of Version field from 2b to 4b in INT Metadata Header

  - Improved the header presentation of the examples and clarified the assumptions in section 6
  - Formatted the spec as a Madoko file
  - **Tag v0.5 spec**

- 2018-02-13
  - Elaborated on interactions between INT and MTU settings. Defined switch behavior when inserting INT metadata in a packet would result in egress link MTU to be exceeded.
  - Defined behavior of INT transit switch when it receives reserved bits set in the INT header

- 2018-02-14
  - Replaced Max Hop Count and Total Hop Count with Remaining Hop Count

- 2018-02-28
  - Added Probe Marker approach as another way to indicate the existence of INT over TCP/UDP (section 5.6.2).

- 2018-03-08
  - Added support for monitoring of two levels of ingress and egress port identifiers

- 2018-03-13
  - Defined INT domain in section 2.
  - Described a possible allocation of non-contiguous DSCP codepoints for INT over TCP/UDP in section 5.6.2.
  - Relaxed the location of INT stack relative to TCP options in section 5.6.2.

- 2018-03-14
  - Added the Checksum Complement metadata.

- 2018-03-29
  - Removed queue congestion status from the list of metadata.
  - Removed Section 4.2 (Handling INT Packets) on slow path processing using follow-up packets.
  - Removed the examples of piggybacked metadata for closed loop control.
  - The expectation is that any of these may be reintroduced in future versions of INT. They could benefit from a better understanding of use cases and some preliminary implementation experience.

- 2018-03-31
  - Defined checksum update behavior more precisely
  - Miscellaneous editorial changes in preparation for v1.0

- 2018-04-02

  - Revised the example transit code to be compliant with spec v1.0, perform incremental TCP/UDP checksum updates, and against PSA architecture instead of v1model.

- 2018-04-03
  - Some more editorial changes for v1.0

- 2018-04-10
  - Removed the option to modify L4 destination port to indicate INT over TCP/UDP.
  - Removed INT Tail header from INT over TCP/UDP encapsulation.
  - Added DSCP to the INT over TCP/UDP shim header.

- 2018-04-20
  - Some more editorial changes for v1.0
  - **Tag v1.0 spec**

- 2018-05-08
  - Fixed checksum subtract/add calls in the reference code

- 2018-08-17
  - Fixed INT DSCP mask in the reference code

- 2018-12-07
  - Added instruction bit for buffer occupancy

- 2019-07-03
  - Added INT modes of operation: INT-XD/MX/MD and INT-CLONE/PROBE-MD.
  - Removed the reference code

- 2020-01-15
  - Added GPE bit to INT shim header for VXLAN GPE encapsulation
  - Swapped the length and reserved bytes in the shim header to align with other INT transports, and to align with the VXLAN GPE shim header format.
  - Changed the length definition in the shim header to exclude the shim header itself, in order to align with the VXLAN GPE shim header format.
  - Increased ingress and egress timestamp size to 8 bytes.

- 2020-01-16
  - Changed the meaning of the length field in every INT shim header. The length of the shim header is NOT included any more.
  - Revised INT over UDP encap, using a new UDP destination port number (INT_TBD).

- 2020-01-28
  - Added Domain ID, Domain Specific (DS) Instructions, and DS Flags.
  - As a result, the INT common header size is increased from 8B to 12B.

  - Removed Rep bits and 'C' bit, introduced 'D' bit for Discarding Copy/Clone at INT Sink.

- 2020-02-11
  - Added 'source-only' metadata as part of DS instruction.
  - Changed the Hop ML to be required only by Transit and Sink devices.

- 2020-02-14
  - Added IPv4/GRE transport for INT.
  - **Tag V2.0 spec**

- 2020-03-04
  - Added example with 'source-only' domain-specific metadata.

- 2020-04-06
  - Added INT-MX Header format, per-hop header operations, and examples

- 2020-04-17
  - Changed 'port identifier' terminology to 'interface identifier'

- 2020-04-30
  - Added INT-MD alternative MTU processing, generating 'Intermediate Report'
  - Changed 'switch id' terminology to 'node id'

- 2020-05-12
  - Changed Geneve Option Class codepoint to value assigned by IANA
  - **Tag V2.1 spec**