NV03 working group
Internet Draft
Category: Standards Track
Expires: November 2015

L. Dunbar
D. Eastlake
Huawei
Tom Herbert
Google

March 3, 2015

NVA Address Mapping Distribution (NAMD) Protocol

draft-dunbar-nvo3-nva-mapping-distribution-01.txt

Status of this Memo

Abstract

This draft describes the protocol for NVA to promptly and incrementally distribute the inner (TS) to outer (NVE) mapping and VN Context to relevant NVEs in a timely manner.

Table of Contents

1. Introduction

Section 4.5 of [nvo3-problem-statement] describes the back-end Network Virtualization Authority (NVA) that is responsible for distributing the mapping information for entire overlay system. [nvo3-nve-nva-cp-req] defines the requirement for the control plane between NVA and NVE.

This draft describes a mechanism for NVA to promptly and incrementally distribute the inner (TS) to outer (NVE) mapping and VN Context to relevant NVEs in a timely manner.

The mechanism described in this document is based on IS-IS protocol with CSNP messages to distribute NVA data base content to all the NVEs and to maintain the database consistency between NVA and NVEs.

For ease of description, the term "NAMD" is used to represent the NVA Address Mapping Distribution protocol.

2. Terminology

The following terms are used interchangeably in this document:

- The terms "Subnet" and "VLAN" because it is common to map one subnet to one VLAN.
- The term "Directory" and "Network Virtualization Authority (NVA)"
- The term "NVE" and "Edge"

Bridge: IEEE Std 802.1Q-2011 compliant device [802.1Q]. In this draft, Bridge is used interchangeably with Layer 2 switch.

CSNP: Complete Sequence Number PDU

CSNP Timeout: The time interval that an NVE can assume NVA is not reachable if the NVE hasn’t received any CNSP updates from NVA during this time. CSNP Timeout is an unsigned byte that gives the amount of time in seconds during which the NVA will send at least three CSNP PDUs. It defaults to 30 seconds.

DA: Destination Address

DC: Data Center

EoR: End of Row switches in data center. Also known as aggregation switches.

End Station: Guest OS running on a physical server or on a virtual machine. An end station in this document has at least one IP address and at least one MAC address, which could be in DA or SA field of a data frame.

LISP: Locator/ID Separation Protocol

RBridge: "Routing Bridge", an alternative name for a TRILL switch.

NVA: Network Virtualization Authority

NVE: Network Virtualization Edge

SA: Source Address

Station: A node, or a virtual node, with IP and/or MAC addresses, which could be in the DA or SA of a data frame.

ToR: Top of Rack Switch in data center. It is also known as access switches in some data centers.

TRILL: Transparent Interconnection of Lots of Links [RFC6325]

TRILL switch: A device implementing the TRILL protocol [RFC6325]

TS: Tenant System

VM: Virtual Machines

VN: Virtual Network

VNID: Virtual Network Instance Identifier

3. Overall Requirement for NVE<->NVA Control Plane

Section 3.1 of [nvo3-cp-req] describes the basic requirement of inner address to outer address mapping for NVO3. A NVE needs to know the mapping of the Tenant System destination (inner) address to the (outer) address (IP) on the Underlying Network of the egress NVE.

Section 3.1 of [nvo3-cp-req] states that a protocol is needed to provide this inner to outer mapping and VN Context to each NVE that requires it and keep the mapping updated in a timely manner. Timely updates are important for maintaining connectivity between Tenant Systems.

4. Terminologies and Assumptions

NVAs can be centralized or distributed with each NVA holding the mapping information for a subset of VNs. By saying that an NVA holds mapping information for a VN, it means that the NVA has mapping information for all the TSs in the VN.

Centralized NVA means that the NVA holds mapping information for all the VNs in the administrative domain. There could be multiple instances of centralized NVA for redundancy purpose.

A NVA could be instantiated on a server/VM attached to a NVE, very much like a TS attached to a NVE, or could be integrated within an NVE. When a NVA is a standalone server/VM attached to a NVE, it has to be reachable via the attached NVE by other NVEs. A NVA can also be instantiated on a NVE that doesn't have any TSs attached. The NVE-NVA control plane for NVA being attached to NVE (like a VM) will require additional functions on NVEs than NVA being embedded in a NVE.

NVA should have at least the following information for each TS:

- Inner Address: TS (host) Address family (IPv4/IPv6, MAC, virtual network Identifier MPLS/VLAN, etc)
- Outer Address: The list of locally attached edges (NVEs); normally one TS is attached to one edge, TS could also be attached to 2 edges for redundancy (dual homing). One TS is rarely attached to more than 2 edges, though it could be possible;
- VN Context (VN ID and/or VN Name)
- Timer for NVEs to keep the entry when pushed down to or pulled from NVEs.
- Optionally the list of interested remote edges (NVEs). This information is for NVA to promptly update relevant edges (NVEs) when there is any change to this TS' attachment to

edges (NVEs). However, this information doesn't have to be kept per TS. It can be kept per VN.

By saying that a NVE is participating in a VN or the VN is active on the NVE, it means that the VN is enabled on the NVE and there is at least one TS of the VN being attached to the NVE.

5. Overview of NVA Address Mapping Distribution (NAMD) Protocol

The mapping entries in NVA could change as TSs move around and changes could be only on some specific TSs. Therefore, it is important to have a mechanism for NVA to send incremental updates to NVEs for the changes instead of entire database of the mapping entries. This document uses CSNP messages of IS-IS to achieve incremental updates from NVAs to NVEs, and to maintain data consistency between NVAs and NVEs.

The mechanism described in this document is based on IS-IS protocol with CSNP messages to distribute NVA content to all the NVEs, inform the incremental changes to the relevant NVEs, and maintain the database consistency between NVA and NVEs.

A NVA can offer services in a Push, Pull model, or the combination of the two.

In Push model, the NVE, upon restart or initialization, sends requests for all the interested VNs as a multicast to all the NVAs. NVAs with the requested VNs use CSNP messages to distribute the mapping entries to the requested NVEs. Whenever, there are changes in the mapping entries, NVA uses CSNP messages to only send the changed portion of the entries.

In the Pull model, an NVA periodically sends VN scoped broadcast messages to all NVEs. An NVE, upon receiving a unknown unicast or ARP/ND with unknown target NVE, sends the pull request to the NVA that supports the VN that the targets belongs to.

6. TLV for NVE reachable addresses

The Reachable Interface Addresses (IA) TLV is used to advertise a set of addresses within a VN being attached to (or reachable by) a specific NVE, and optionally the NVE Virtual Access Point.

These addresses can be in different address families. For example, it can be used to declare that a particular interface

with specified IPv4, IPv6, and 48-bit MAC addresses in some particular VN is reachable from a particular NVE.

This document suggests using the Interface Addresses APPsub-TLV defined by [IA] except using NVE address subTLV in the fourth field shown below:

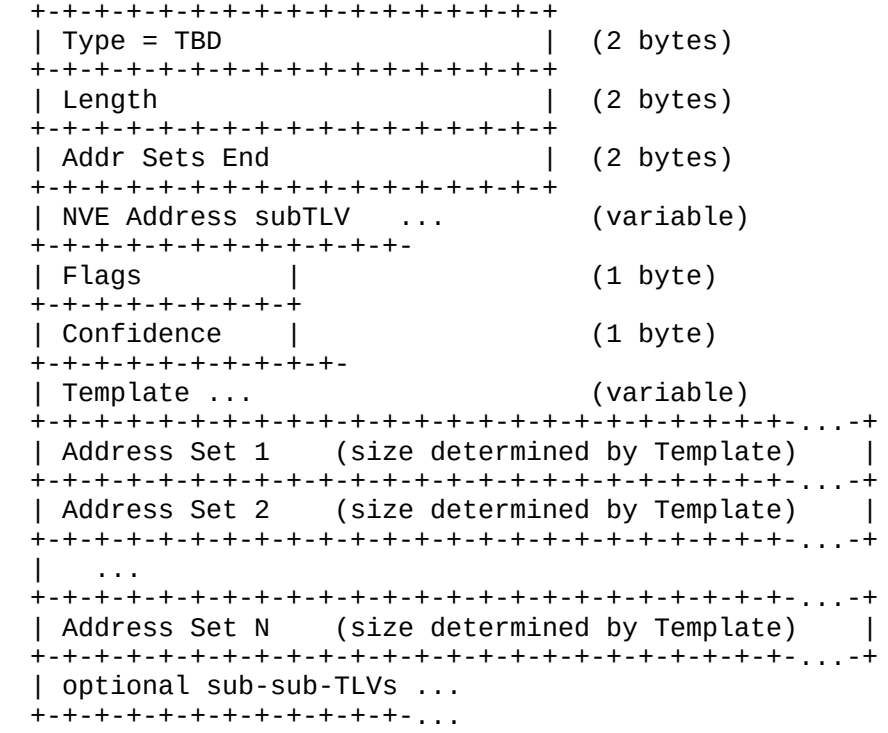


Figure 1. The Interface Addresses APPsub-TLV

Addr Sets End: The unsigned integer offset of the byte, within the IA APPsub-TLV [IA] value part, of the last byte of the last Address Set. This will be the byte just before the first sub-sub-TLV if any sub-sub-TLVs are present (see Section 3). If this is equal to Length, there are no sub-sub-TLVs. If this is greater than Length or points to before the end of the Template, the IA APPsub-TLV is corrupt and MUST be discarded. This field is always two bytes in size.

7. Push Mechanism

Under this mode, NVA pushes the inner-outer mapping for all the TSs of the VNs to relevant NVEs. This service is scoped by VN. A Push NVA also advertises whether or not it believes it has pushed complete mapping information for a VN. It might be pushing only a subset of the mapping and/or reachability information for a VN. The Push Model uses the CSNP messages as its distribution mechanism.

With the Push model, if the destination of a data frame arriving at the Ingress NVE can't be found in its inner-outer mapping database that are pushed down from the NVA, the Ingress edge could be configured with one or more of the following policies:

- simply drop the data frame,
- flood the data frames to other NVEs that have the VN enabled, or
- start the "pull" process to get information from Pull NVA.
  When the NVE is waiting for reply from the Pull process, the NVE can either drop or queue the packet.

One drawback of the Push Mode is that it usually will push more mapping entries to an NVE than needed. Under the normal process of edge cache aging and unknown destination address flooding, rarely used entries would have been removed. It would be difficult for NVA to predict the communication patterns among TSs within one VN. Therefore, it is likely that the NVA will push down all the entries for all the VNs that are enabled on the NVE.

Another drawback with Push model: there really can't be any source-based policy. It's all or nothing.

7.1. Requesting Push Service

When a NVE is initialized or re-started, it needs to send request to the relevant NVAs to push down the mapping information for the active VNs on the NVE. NVE could use Virtual Network scoped instances of the IS-IS to announce all the Virtual Networks in which it is participating to NVAs who have the mapping

information for the VNs. A new subTLV (Enabled-VN TLV) under the IS-IS Router Capability TLV [RFC4971] is specified here for NVE to indicate all its interested VNs in the IS-IS LSP message.

For 24-bits VN ID, there could be 16 million VNs. Multiple ways can be used to express the interested VNs:

- Starting VN & End VNs & bit map for the VNs in between.
- Starting VN & End VN (for the VNs that are contiguous)
- Individual VN listing (for a small number of VNs that are not contiguous)

Therefore 3 different types of subTLV are specified:

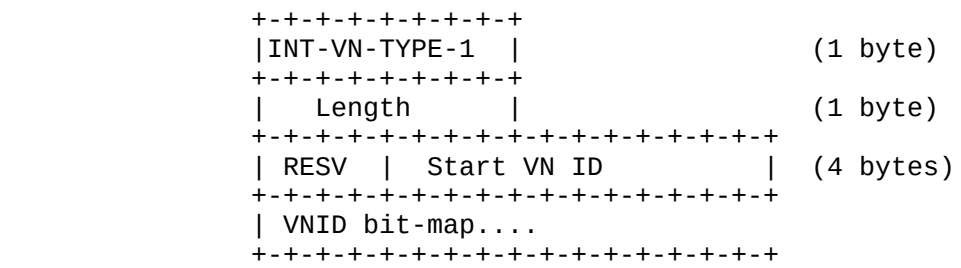


Figure 2. Enabled-VN TLV using bit map

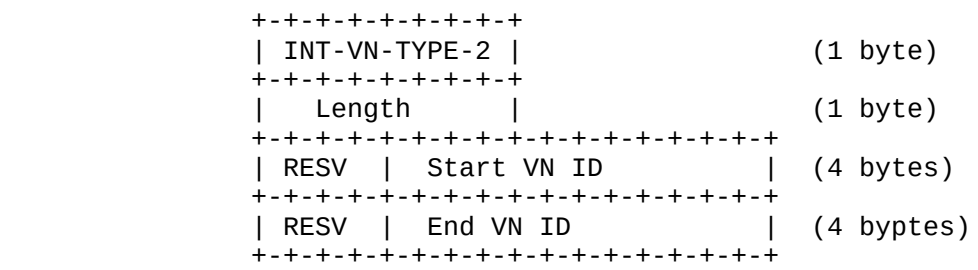


Figure 3. Enabled-VN TLV using Range

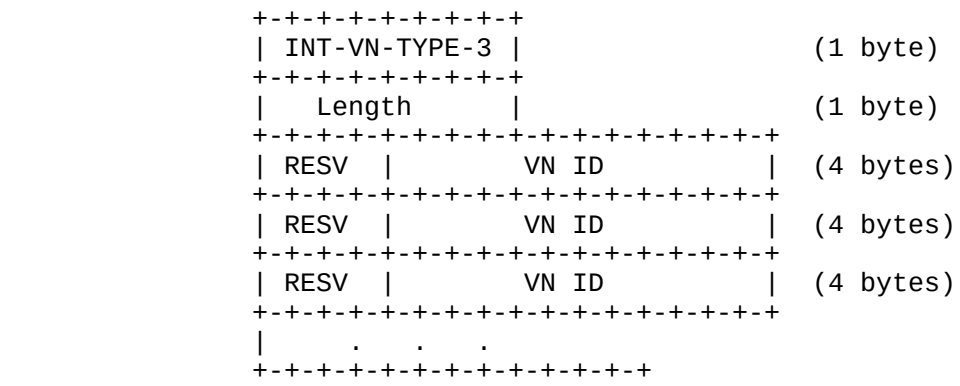


Figure 4. Enabled-VN TLV using list

- Type: indicating different ways to express the VNs that NVE is participating: INT-VN-TYPE-1 is for using bit map to express the interested VNs; INT-VN-TYPE-2 is for using range to express the interested VNs (if the interested VNs are contiguous); IT-VN-TYPE-3 is for using individual VN list to express the interested VNs.

- Length: Variable.

- RESV: 4 reserved bits that MUST be sent as zero and ignored on receipt.

- Start VN ID: The 24-bit VN-ID that is represented by the high-order bit of the first byte of the VN-ID bit-map.

VN-ID bit-map: The highest-order bit indicates the VN equal to the start VN ID, the next highest bit indicates the VN equal to start VN ID + 1, continuing to the end of the VN bit-map field.

If this sub-TLV occurs more than once in a Hello, the set of enabled VNs is the union of the sets of VNs indicated by each of the Enabled-VLAN sub-TLVs in the Hello.

When NVA is distributed, there could be multiple NVAs with each hosting mapping information for a subset of VNs.

Each NVA advertises its availability to push mapping information for a particular virtual network to all NVEs who participate in the VN. NVEs subscribe the relevant NVAs.

The subscription is VN scoped, so that a NVA doesn't need to push down the entire set of mapping entries. Each Push NVA also has a priority. For robustness, the one or two NVAs with the highest priority are considered as Active in pushing information for the VN to all NVEs who have subscribed for that VN.

7.2. Incremental Push Service

Whenever there is any change in TS' association to an NVE, which can be triggered by TS being added, removed, or de-commissioned, an incremental update has be sent to the NVEs that are impacted by the change. Therefore, sequence numbers have to be maintained by NVA and edges NVEs. CSNP message is used to achieve the incremental updates and to maintain the database consistency between NVAs and NVEs. We assume that NVA gets notification from an authoritative entity, such as VM management system when TS-NVE attachment changes occur.

A new TLV is needed for to carry CSNP timeout value and a flag for NVA to indicate it has completed all updates.

If the Push NVA is configured to believe it has complete mapping information for VN X then, after it has actually transmitted all of its LSPs for VN X it sets the Complete Push (CP) bit to one. It then maintains the CP bit as one as long as it is Active.

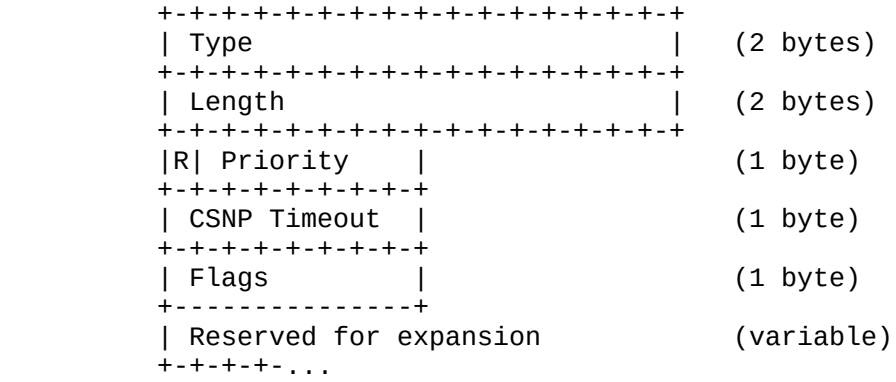
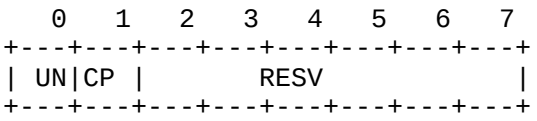


Figure 3. CSNP Complete TLV

Flags: A byte of flags defined as follows:



The UN flag indicates that the NVA originating the NVA-LSP, including this parameter data, will accept and properly process NVA- PDUs sent by unicast

The CP flag is to indicate that NVA has completed its update.

8. Pull Mechanism

Under this mode, an NVE pulls the mapping entries from the NVA when its cache doesn't have the mapping entries.

The main advantage of Pull Mode is that the mapping is stored only where it needs to be stored and only when it is required. In addition, in the Pull Mode, NVEs can age out mapping entries if they haven't been used for a certain period of time. Therefore, each NVE will only keep the entries that are frequently used, so its mapping table size will be smaller than a complete table pushed down from NVA.

The drawback of Pull Mode is that it might take some time for NVEs to pull the needed mapping from NVA. Before NVE gets the response from NVA, the NVE has to buffer the subsequent data frames with destination address to the same target. The buffer could overflow before the NVE gets the response from NVA. However, this scenario should not happen very often in data center environment because most likely the TSs are end systems which have to wait for (TCP) acknowledgement before sending subsequent data frames. Another option is forward, not flood, subsequent frames to a default location, i.e. forward to a re-encapsulating NVE.

The practice of an edge waiting and dropping packets upon receiving an unknown DA is not new. Most deployed routers today drop packets while waiting for target addresses to be resolved. It is too expensive to queue subsequent packets while resolving target address. The routers send ARP/ND requests to the target upon receiving a packet with DA not in its ARP/ND cache and wait for an ARP or ND responses. This practice minimizes flooding when targets don't exist in the subnet. When the target doesn't exist in the subnet, routers generally re-send an ARP/ND request a few more times before dropping the packets. The holding time by

routers to wait for an ARP/ND response when the target doesn't exist in the subnet can be longer than the time taken by the Pull Mode to get mapping from NVA.

8.1. Pull Query Format

Here are some events that can trigger the pulling process:

- An NVE receives a data frame from the attached TSs with a destination whose attached NVE is unknown, or
- The NVE receives an ingress ARP/ND request for a target whose link address (MAC) or attached NVE is unknown.

Each Pull request can have queries for multiple inner-outer mapping entries. The message format is defined below:

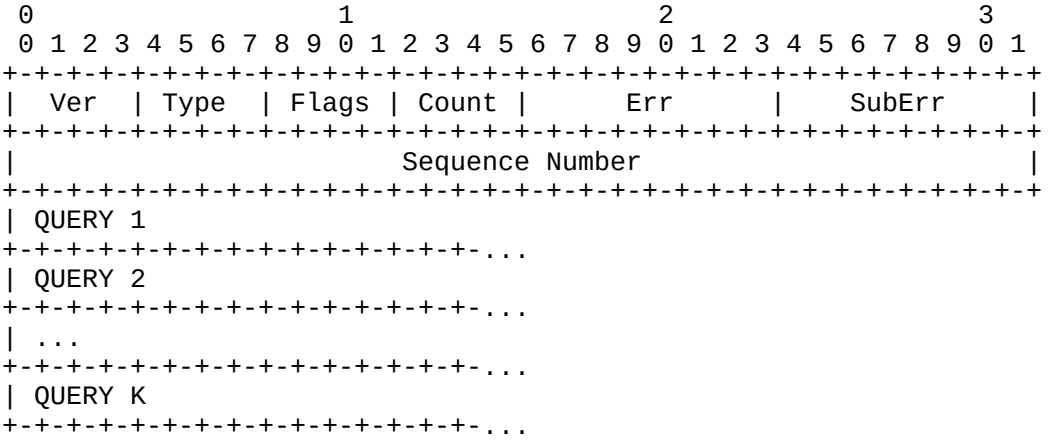


Figure 4. Pull Query TLV

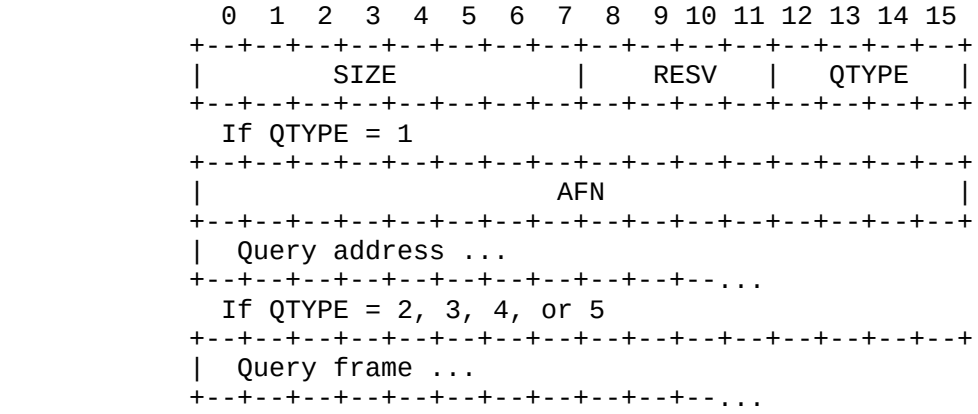
Type: 1 for Query. Queries received by an NVE that is not a Pull NVA result in an error response unless inhibited by rate limiting.

Flags, Err, and SubErr: MUST be sent as zero and ignored on receipt.

Count: Number of QUERY Records present. A Query message Count of zero is explicitly allowed, for the purpose of pinging a Pull NVA server to see if it is responding. On receipt of such

an empty Query message, a Response message that also has a Count of zero is sent unless inhibited by rate limiting.

QUERY: Each QUERY Record within a Pull Directory Query message is formatted as follows:



SIZE: Size of the QUERY record in bytes as an unsigned integer starting after the SIZE field and following byte. Thus the minimum legal value is 2. A value of SIZE less than 2 indicates a malformed QUERY record. The QUERY record with the illegal SIZE value and any subsequent QUERY records MUST be ignored and the entire Query message MAY be ignored.

RESV: A block of reserved bits. MUST be sent as zero and ignored on receipt.

QTYPE: There are several types of QUERY Records currently defined in two classes as follows: (1) a QUERY Record that provides an explicit address and asks for all addresses for the interface specified by the query address and (2) a QUERY Record that includes a frame. The fields of each are specified below. Values of QTYPE are as follows:

QTYPE	Description
0	reserved
1	address query
2	ARP query frame
3	ND query frame
4	RARP query frame
5	Unknown unicast MAC query frame
6-14	assignable by IETF Review

15 reserved

AFN: Address Family Number of the query address.

Address Query: The query is asking for any other addresses, and the address of NVE from which they are reachable, that correspond to the same interface, within the VN of the query. Typically that would be either (1) a MAC address with the querying NVE primarily interested in the NVE by which that MAC address is reachable, or (2) an IP address with the querying NVE interested in the corresponding MAC address and the NVE by which that MAC address is reachable. But it could be some other address type.

Query Frame: Where a QUERY Record is the result of an ARP, ND, RARP, or unknown unicast MAC destination address, the ingress NVE MAY send the frame to a Pull NVA if the frame is small enough that the resulting Query message not exceeding the MTU.

If no response is received to a Pull Directory Query message within a timeout configurable in milliseconds that defaults to 200, the Query message should be re-transmitted with the same Sequence Number up to a configurable number of times that defaults to three. If there are multiple QUERY Records in a Query message, responses can be received to various subsets of these QUERY Records before the timeout. In that case, the remaining unanswered QUERY Records should be re-sent in a new Query message with a new sequence number. If an NVE is not capable of handling partial responses to queries with multiple QUERY Records, it MUST NOT send a Request message with more than one QUERY Record in it.

8.2. Pull Response

There are several possibilities of the Pull Response:

1. Valid inner-outer address mapping, coupled with the valid timer indicating how long the entry can be cached by the NVE.
   The timer for cache should be short in an environment where VMs move frequently. The cache timer can also be configured.

2. The target being queried is not available. The response should include the policy if requester should forward data frame in legacy way, or drop the data frame.

3. The requestor is administratively prohibited from getting an informative response.

Pull NVA Response messages are sent as unicast to the requesting NVE. Responses are sent with the same VN. The specific data format is as follows:

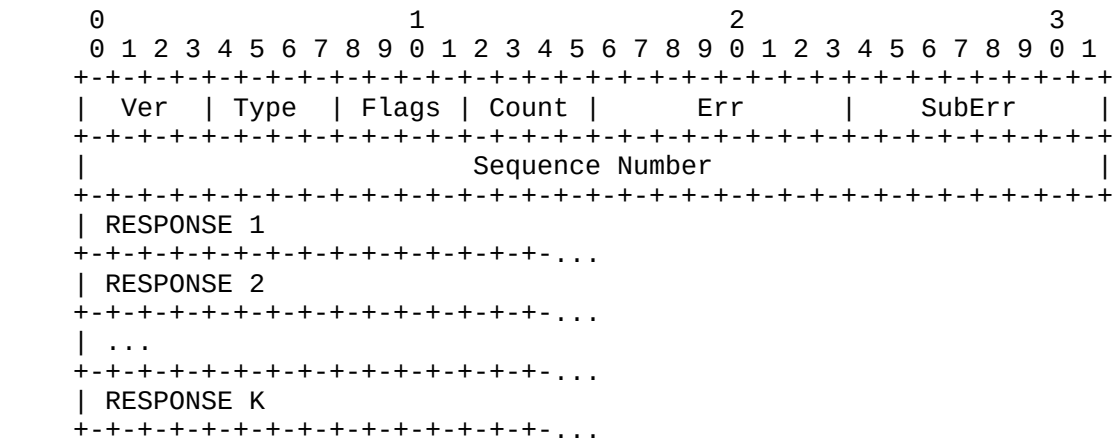


Figure 5. Pull Response TLV

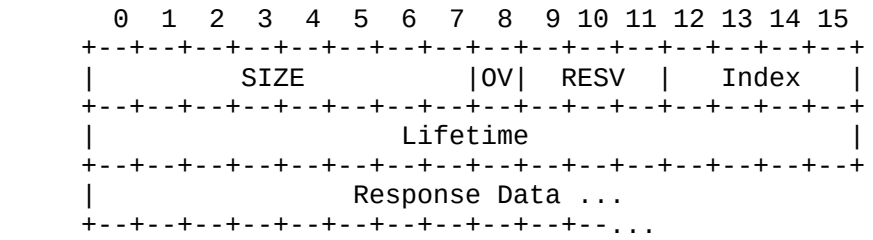
Type: 2 = Response.

Flags: MUST be sent as zero and ignored on receipt.

Count: Count is the number of RESPONSE Records present in the Response message.

Err, SubErr: A two part error code. Zero unless there was an error in the Query message, for which case see Section 3.5.

RESPONSE: Each RESPONSE record within a Pull NVA Response message is formatted as follows:



SIZE: Size of the RESPONSE Record in bytes starting after the SIZE field and following byte. Thus the minimum value of SIZE is 2. If SIZE is less than 2, that RESPONSE Record and all subsequent RESPONSE Records in the Response message MUST be ignored and the entire Response message MAY be ignored.

OV: The overflow flag. Indicates, as described below, that there was too much Response Data to include in one Response message.

RESV: Four reserved bits that MUST be sent as zero and ignored on receipt.

Index: The relative index of the QUERY Record in the Query message to which this RESPONSE Record corresponds. The index will always be one for Query messages containing a single QUERY Record. If the Index is larger than the Count that was in the corresponding Query, that RESPONSE Record MUST be ignored and subsequent RESPONSE Records or the entire Response message MAY be ignored.

Lifetime: The length of time for which the response should be considered valid in units of 200 milliseconds except that the values zero and 2**16-1 are special. If zero, the response can only be used for the particular query from which it resulted and MUST NOT be cached. If 2**16-1, the response MAY be kept indefinitely but not after the Pull NVA goes down or becomes unreachable. The maximum definite time that can be expressed is a little over 3.6 hours.

Response Data: There are various types of RESPONSE Records.

- If the Err field is non-zero, then the Response Data is a copy of the corresponding QUERY Record data, that is, either an AFN followed by an address or a query frame.

- If the Err field is zero and the corresponding QUERY Record was an address query, then the Response Data is the contents

of an Interface Addresses APPsub-TLV [IA]. The maximum size of such contents is 253 bytes in the case when SIZE is 255.

- If the Err field is zero and the corresponding QUERY Record was a frame query, then the Response data consists of the response frame for ARP, ND, or RARP and a copy of the frame for unknown unicast destination MAC.

Multiple RESPONSE Records can appear in a Response message with the same index if the answer to a QUERY Record consists of multiple Interface Address APPsub-TLV contents. This would be necessary if, for example, a MAC address within a Data Label appears to be reachable by multiple NVEs. However, all RESPONSE Records to any particular QUERY Record MUST occur in the same Response message. If a Pull NVA holds more mappings for a queried address than will fit into one Response message, it selects which to include by some method outside the scope of this document and sets the overflow flag (OV) in all of the RESPONSE Records responding to that query address.

If no response is received from a Pull request within a configurable timeout, the request should be re-transmitted with the same Sequence Number up to a configurable number of times that defaults to three.

8.3. Cache Consistency

It is important that the cached information be kept consistent with the actual placement of VMs. Therefore, it is highly desirable to have a mechanism to prevent NVEs from using the staled mapping entries.

When there is any change in a Pull NVA, such as an entry being deleted or new entry added, and there may be unexpired stale information at some NVEs, the Pull NVA MUST send an unsolicited Update message to the relevant NVEs.

To achieve this goal, a Pull NVA server MUST maintain one of the following, in order of increasing specificity.

1. An overall record per VN of when the last returned query data will expire at a requestor and when the last query record specific negative response will expire.

2. For each unit of data (IA APPsub-TLV Address Set) held by the NVA and each address about which a negative response was

sent, when the last expected response with that unit or negative response will expire at a requester.

Note: It is much more important to cache negative reply, because there are many invalid address queries. Study has shown that for each valid ND query, there are 100's of invalid address queries.

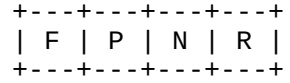
3. For each unit of data held by the NVA and each address about which a negative response was sent, a list of NVEs that were sent that unit as the response or sent a negative response to the address, with the expected time to expiration at each of them.

8.4. Update Message Format

An Update message is formatted as a Response message except that the Type field in the message header is a different value.

Update messages are initiated by a Pull NVA. The Sequence number space used is controlled by the originating Pull NVA and different from Sequence number space used in a Query and the corresponding Response that are controlled by the querying NVE.

The Flags field of the message header for an Update message is as follows:



F: The Flood bit. If zero, the response is to be unicast. If F=1, it is multicast to relevant NVEs.

P, N: Flags used to indicate positive or negative Update messages. P=1 indicates positive. N=1 indicates negative. Both may be 1 for a flooded all addresses Update.

R: Reserved. MUST be sent as zero and ignored on receipt

8.5. Acknowledge Message Format

An Acknowledge message is sent in response to an Update to confirm receipt or indicate an error unless response is inhibited by rate limiting. It is also formatted as a Response message.

If there are no errors in the processing of an Update message, the message is essentially echoed back with the Type changed to Acknowledge.

If there was an overall or header error in an Update message, it is echoed back as an Acknowledge message with the Err and SubErr fields set appropriately.

If there is a RESPONSE Record level error in an Update message, one or more Acknowledge messages may be returned.

8.6. Pull Request Errors

If errors occur at the query level, they MUST be reported in a response message separate from the results of any successful queries. If multiple queries in a request have different errors, they MUST be reported in separate response messages. If multiple queries in a request have the same error, this error response MAY be reported in one response message.

8.7. Redundant Pull NVAs

There could be multiple NVAs holding mapping information for a particular VN for reliability or scalability purposes. Pull NVAs advertise themselves by having the Pull Directory flag on in their Interested VNs sub-TLV [rfc6326bis].

A pull request can be sent to any of them that is reachable but it is RECOMMENDED that pull requests be sent to a NVA that is least cost from the requesting NVE.

9. Hybrid Mode

For some edge nodes that have great number of VNs enabled and combined number of TSs under all those VNs are large, managing the inner-outer address mapping for TSs under all those VNs can be a challenge. This is especially true for Data Center

gateway nodes, which need to communicate with a majority of VNs if not all.

For those NVE nodes, a hybrid mode should be considered. That is the Push Mode being used for some VNs, and the Pull Mode being used for other VNs. It is the network operator's decision by configuration as to which VNs' mapping entries are pushed down from NVA and which VNs' mapping entries are pulled.

In addition, NVA can inform the NVE to use legacy way to forward if it doesn't have the mapping information, or the NVE is administratively prohibited from forwarding data frame to the requested target.

10. Redundancy

For redundancy purpose, there should be multiple NVAs that hold mapping information for each VN. At any given time, only one or a small number of push NVAs is considered as active for a particular VN. All NVAs should announce its capability and priority to all the edges.

11. Inconsistency Processing

If an NVE notices that a Push NVA is no longer reachable, it MUST ignore any mapping entries from that NVA because it is no longer being updated and may be stale.

There may be transient conflicts between mapping information from different Push NVAs or conflicts between locally learned information and information received from a Push NVA. NVA may have a confidence level with address table information so, in case of such conflicts, information with a higher confidence value is preferred over information with a lower confidence. In case of equal confidence, Push NVA information is preferred to locally learned information and if information from Push NVAs conflicts, the information from the higher priority Push NVA is preferred.

12. Security Considerations

Incorrect information in NVA can result in a variety of security threats including the following:

Incorrect directory mappings can result in data being delivered to the wrong hosts/VMs, or set of hosts in the case of multi-destination packets, violation security policy.

Missing or incorrect data in NVA can result in denial of service due to sending data packets to black holes or discarding data on ingress due to incorrect information that their destinations are not reachable.

Push NVA data is distributed through CSNP messages that can be authenticated with the same mechanisms as IS-IS LSPs. See [RFC5304] and [RFC5310].

13. IANA Considerations

This section gives IANA allocation and registry considerations.

14. Acknowledgements

Special thanks to David Black, Dino Farinacci, Mingui Zhang, XiaoHu Xu for valuable suggestions and comments to this draft.

15. References

15.1. Normative References

[RFC4971] J. Vasseur et al, "Intermediate System to Intermediate System (IS-IS) Extensions for Advertising Router Information", July 2007.

[nvo3-nve-nva-cp-req] draft-ietf-nvo3-nve-nva-cp-req-00, "Network Virtualization NVE to NVA Control Protocol Requirements", Kreeger, et al. July 31, 3013.

[IA] - Eastlake, D., L. Yizhou, R. Perlman, "TRILL: Interface Addresses APPsub-TLV", draft-ietf-trill-ia-appsubtlv, work in progress.

15.2. Informative References

[802.1Q] IEEE Std 802.1Q-2011, "IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks", May 2011.

[802.1Qbg] IEEE Std 802.1Qbg-2012, "Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks-Edge Virtual Bridging", July 2012.

[RFC826] Plummer, D., "An Ethernet Address Resolution Protocol", RFC 826, November 1982.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

Authors' Addresses

Linda Dunbar
Huawei Technologies
5430 Legacy Drive, Suite #175
Plano, TX 75024, USA
Phone: (469) 277 5840
Email: linda.dunbar@huawei.com

Donald Eastlake
Huawei Technologies
155 Beaver Street
Milford, MA 01757 USA
Phone: 1-508-333-2270
Email: d3e3e3@gmail.com

Tom Herbert
Google
Email: therbert@google.com

INTERNET-DRAFT
Intended Status: Experimental
Expires: July 2015

T. Herbert
Google

January 20, 2015

Identifier-locator addressing for network virtualization
draft-herbert-nvo3-ila-00

Status of this Memo

Copyright and License Notice

Abstract

This specification describes identifier-locator addressing (ILA) in IPv6 for network virtualization. Identifier-locator addressing differentiates between location and identity of a network node. Part of an address expresses the immutable identity of the node, and another part indicates the location of the node which can be dynamic. In the context of virtualization, a virtual address serves as an identifier and the address of the host where the associated tenant system currently resides is a locator.

Table of Contents

1 Introduction

This document describes the data path, address formats, and expected use cases of identifier-locator addressing in IPv6 ([RFC2460]). The Identifier-Locator Network Protocol (ILNP) ([RFC6740], [RFC6741]) defines a protocol and operations model for identifier-locator addressing in IPv6. Many concepts here are taken from ILNP, however there are some differences in the context of network virtualization--for instance we assume that a centralized control plane will be implemented that provides mappings of identifiers to locators.

In identifier-locator addressing, an IPv6 address is split into a locator and an identifier component. The locator indicates the physical location in the network for a node, and the identifier indicates the node’s identity which is the logical or virtual endpoint in communications. Locators are routable within a network, but identifiers typically are not. An application addresses a destination by identifier. Identifiers are mapped to locators for transit in the network. The on-the-wire address is composed of a locator and an identifier: the locator is sufficient to route the packet to a physical host, and the identifier allows the receiving host to forward the packet to the addressed application.

Identifiers are not statically bound to a host on the network, and in fact their binding (or location) may change. This is the basis for network virtualization and address migration. An identifier is mapped to a locator at any given time, and a set of identifier to locator mappings is propagated throughout a network to allow communications. The mappings are kept synchronized so that if an identifier migrates to a new physical host, its identifier to locator mapping is updated.

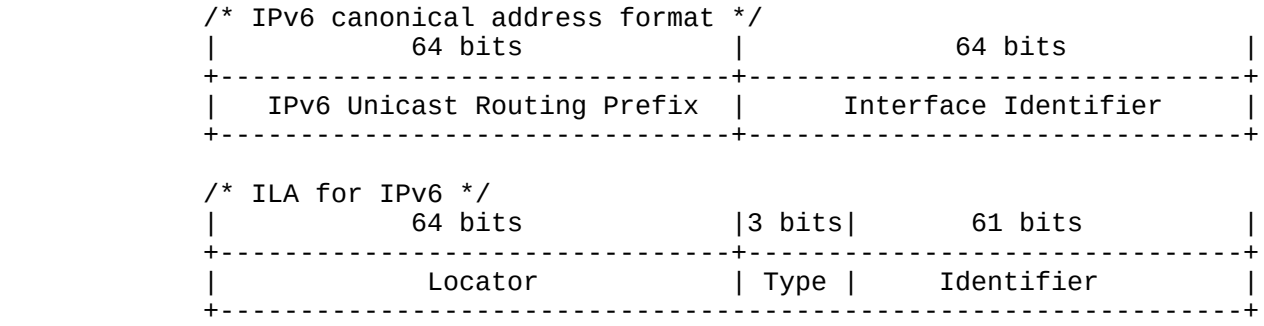
In network virtualization, an identifier may further be split into a virtual network identifier and virtual host address. With identifier-locator addressing network virtualization can be implemented in an IPv6 network without any additional encapsulation headers. Packets sent with identifier-locator addresses look like plain unencapsulated packets (e.g. TCP/IP packets). This "encapsulation" is transparent to the network, so protocol specific mechanisms in network hardware work seamlessly. These mechanisms include hash calculation for ECMP, NIC large segment offload, checksum offload, etc.

2 Address formats

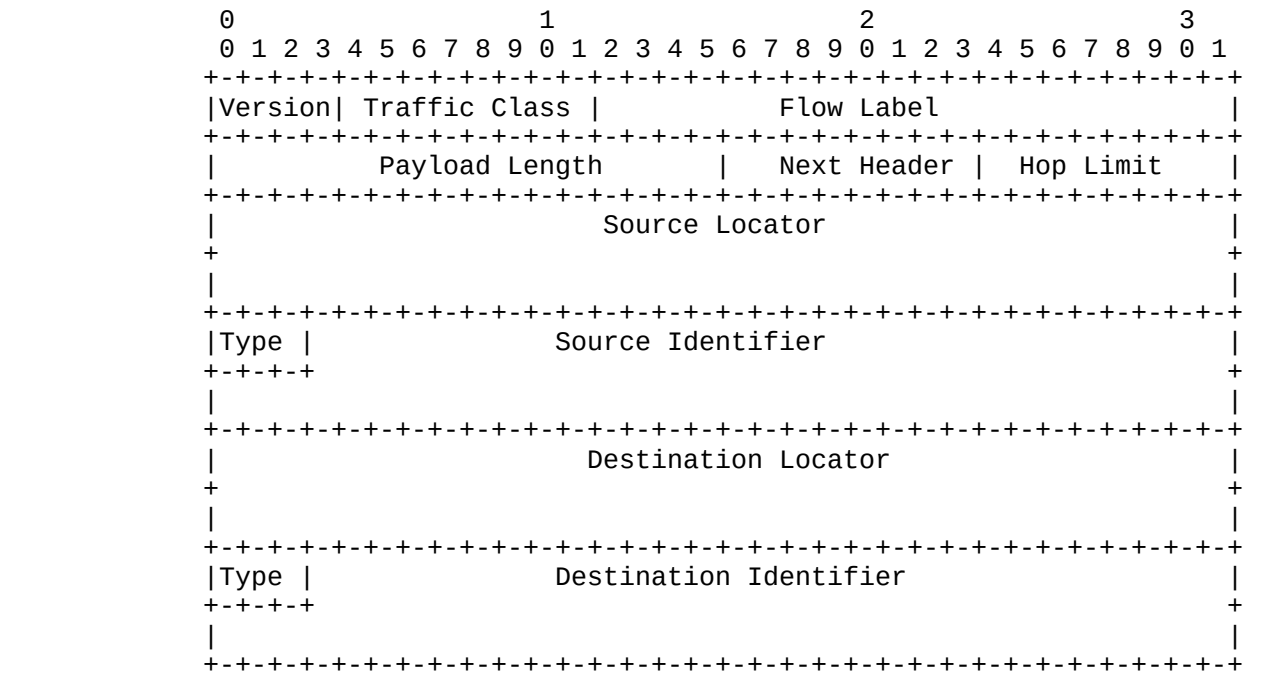
This section describes the address formats associated with identifier-locator addressing in network virtualization.

2.1 ILA format

As described in ILNP ([RFC6741]) an IPv6 address may be encoded to hold a locator and identifier where each occupies 64 bits. In ILA, the upper three bits of the identifier indicate an identifier type.



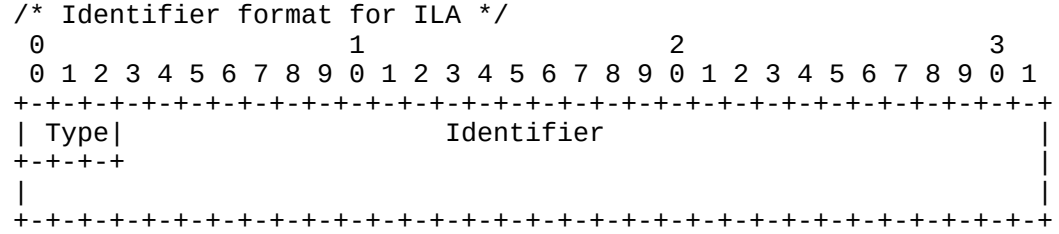
An IPv6 header with ILA addresses would then have the format:



Note that there is no requirement that both the source and destination are identifier-locator addresses.

2.2 Identifier format

An ILA identifier includes a three bit type field and sixy-one bits for an identfier value.



- Type: Type of the identifier (see below).

- Identifier: Identifier value.

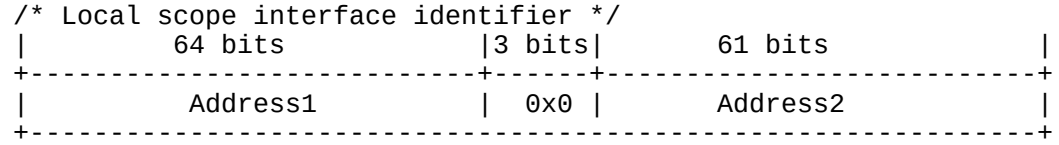
2.3 Identifier types

Defined identifier types are:

0: interface identifier
1: locally unique identifier
2: virtual networking identifier for IPv4 address
3: virtual networking identifier for IPv6 unicast address
4: virtual networking identifier for IPv6 multicast address
5-7: Reserved

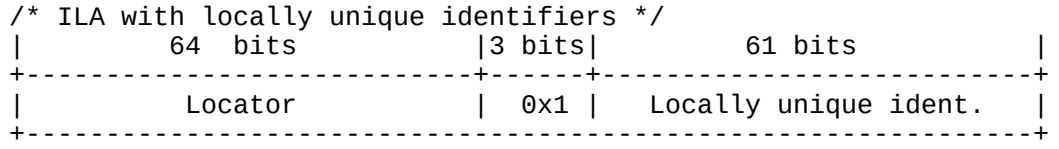
2.4 Interface identifiers

The interface identifier type indicates a plain local scope interface identifier. When this type is used the address is a normal IPv6 address without identifier-locator semantics.



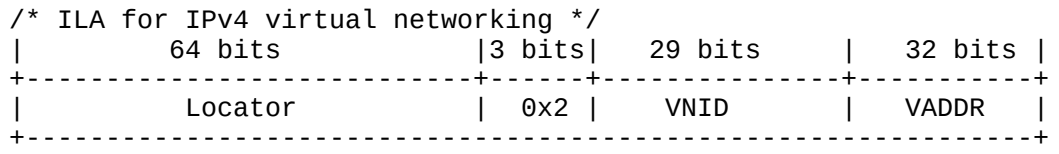
2.5 Locally unique identifiers

Locally unique identifiers (LUI) can be created for various addressable nodes within a network. These identifiers are in a flat 61 bit space and must be unique within a domain (unique within a site for instance). To simplify administration, hierarchical allocation of locally unique identifiers may be done.



2.6 Virtual networking identifiers for IPv4

This type defines a format for encoding an IPv4 virtual address and virtual network identifier within an identifier.



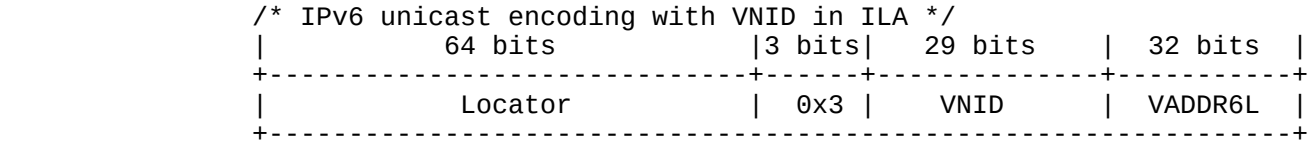
VNID is a virtual network identifier and VADDR is a virtual address within the virtual network indicated by the VNID. The VADDR can be an IPv4 unicast or multicast address, and may often be in a private address space (i.e. [RFC1918]) used in the virtual network.

2.7 Virtual networking identifiers for IPv6

A virtual network identifier and an IPv6 virtual host address (tenant visible address) can be encoded within an identifier. Encoding the virtual host address involves mapping the 128 bit address into a sixy-one bit identifier. Different encodings are used for unicast and multicast addresses.

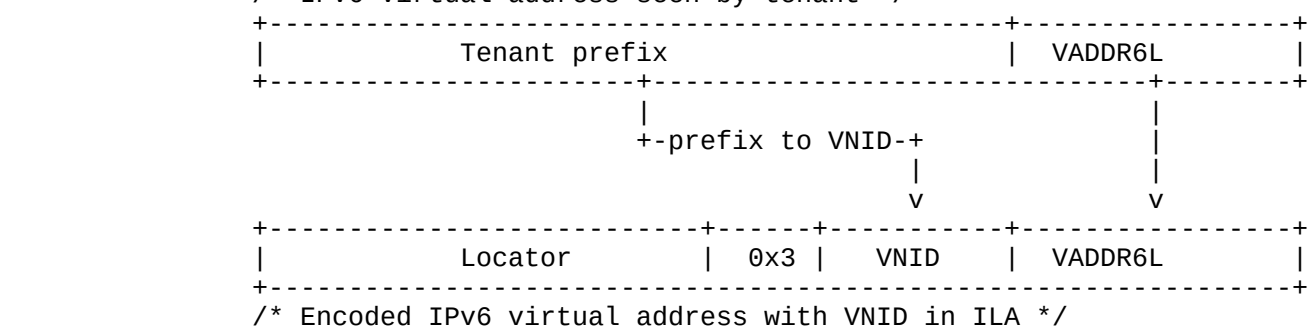
2.7.1 Virtual networking identifiers for IPv6 unicast

In this format, the virtual network identifier and virtual IPv6 unicast address are encoded within an identifier. To facilitate encoding of virtual addresses, there is a unique mapping between a VNID and a 96 bit prefix.

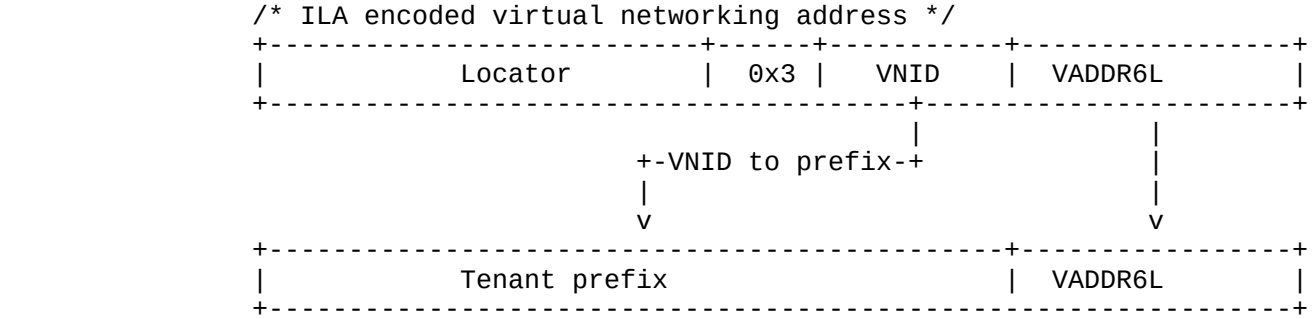


VADDR6L contains the low order 32 bits of the IPv6 virtual address. The upper 96 bits of the virtual address inferred from the VNID to prefix mapping.

The figure below illustrates encoding a tenant IPv6 virtual unicast address into a ILA address.

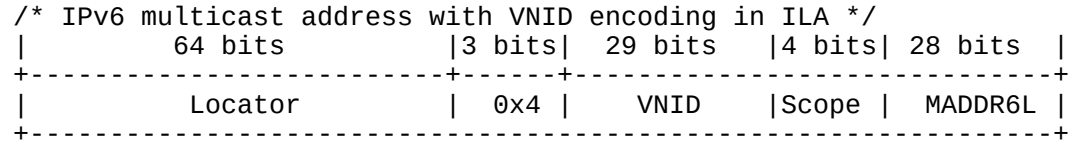


This encoding is reversible, given an ILA address, the virtual address visible to the tenant can be deduced:



2.7.2 Virtual networking identifiers for IPv6 multicast

In this format, a virtual network identifier and virtual IPv6 multicast address are encoded within an identifier.



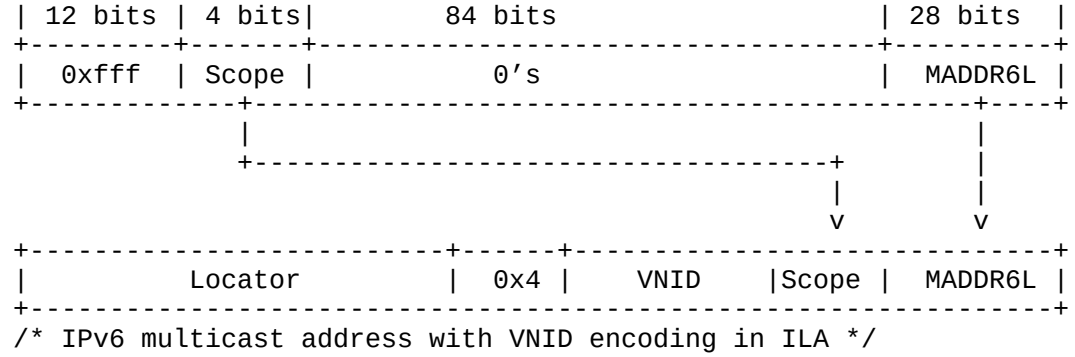
This format encodes a multicast IPv6 address in an identifier. The scope indicates multicast address scope as defined in [RFC7346]. MADDR6L is the low order 28 bits of the multicast address. The full multicast address is thus:

ff0<Scope>::0<MADDRL6 high 12 bits>:<MADDRL6 low 16 bits>

This encoding permits encoding of multicast addresses of the form:

ff0X::0 to ff0X::0fff:ffff

The figure below illustrates encoding a tenant IPv6 virtual multicast address into an ILA address.

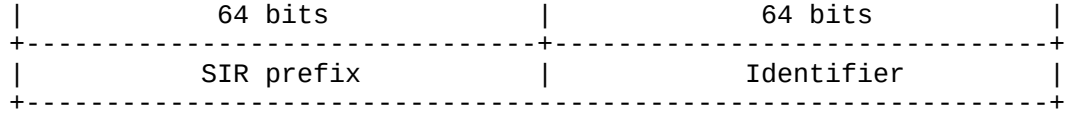


2.8 Standard identifier representation addresses

An identifier serves as the external representation of a network node. For instance, an identifier may refer to a specific host, virtual machine, or tenant system. When a host initiates a connection or sends a packet, it uses the identifier to indicate the peer endpoint of the communication. The endpoints of an established connection context also nominally refer to identifiers. It is only when the packet is actually being sent over a network that the locator for the identifier needs to be resolved.

In order to maintain compatibility with existing networking stacks and applications, identifiers are encoded in IPv6 addresses using a standard identifier representation (SIR) address. A SIR address is a

combination of a prefix which occupies what would be the locator portion of an ILA address, and the identifier in its usual location.

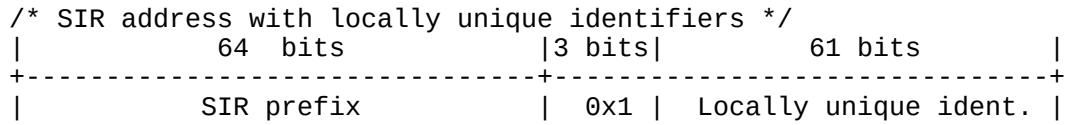


A SIR prefix may be may be site-local, or globally routable. A globally routable SIR prefix allows connectivity between hosts on the Internet and ILA endpoints. A gateway between a site's network and the Internet can translate between SIR prefix and locator for an identifier. A network may have multiple SIR prefixes, and may also allow tenant specific SIR prefixes in network virtualization.

The standard identifier representation can be used as the externally visible address for an node. This can used throughout the network, returned in DNS AAAA records ([RFC3363]), used in logging, etc. An application can use a SIR address without knowledge that it encodes an identifier.

2.8.1 SIR for locally unique identifiers

The SIR address for a locally unique identifier has format:



When using ILA with locally unique identifiers a flow tuple logically has the form:

(source identifier, source port,
destination identifier, destination port)

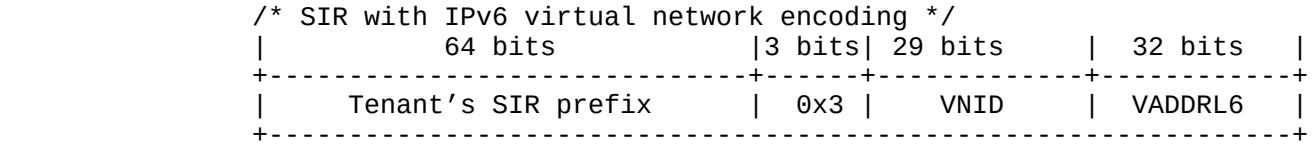
Using standard identifier representation the flow is then represented with IPv6 addresses:

(source SIR address, source port,
destination SIR address, destination port)

2.8.2 SIR for virtual addresses

An ILA virtual address may be encoded using the standard identifier representation. For example, the SIR address for an IPv6 virtual

address may be:



In a tenant system, a flow tuple would have the form:

(local VADDR, local port, remote VADDR, remote port)

After translating packets for the flow into ILA, the flow would be identified on-the-wire as:

((local VNID, local VADDR), local port,
(remote VNID, remote VADDR), remote port

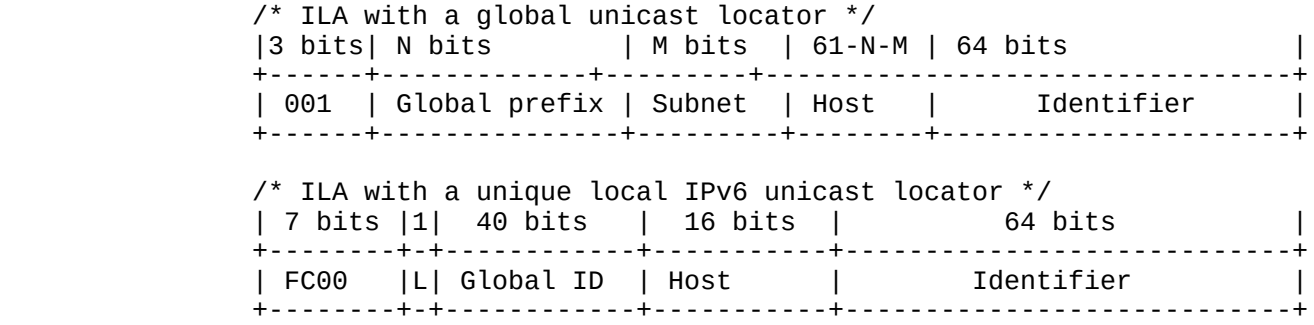
A tenant may communicate with a peer in the network which is not in its virtual network, for instance to reach a network service (see below). In this case the flow tuple at the peer may be:

(local SIR address, local port,
remote SIR address, remote port)

In this example, the remote SIR address is a SIR address for a virtual networking identifier, however from peer's connectivity perspective this is not distinguishable from a SIR address with a locally unique identifier or even a non-ILA address.

2.9 Locators

Locators are routable network address prefixes that address physical hosts within the network. They may be assigned from a global address block [RFC3587], or be based on unique local IPv6 unicast addresses as described in [RFC4193].



3 Operation

This section describes operation methods for using identifier-locator addressing with network virtualization.

3.1 Identifier to locator mapping

An application initiates a communication or flow using a SIR address or virtual address for a destination. In order to send a packet on the network, the destination identifier is mapped to a locator. The mappings are not expected to change frequently, so it is likely that locator mappings can be cached in the flow contexts.

Identifier to locator mapping is nearly identical to the mechanism needed in virtual networking to map a virtual network and virtual host address to a physical host. These mechanisms should leverage a common solution.

The mechanisms of propagating and maintaining identifier to locator mappings are outside the scope of this document.

3.2 Address translations

With ILA, address translation is performed to convert SIR addresses to ILA addresses, and ILA addresses to SIR addresses. Translation may be done on either the source or destination address of a packet. Translation is stateless and is done per IPv6-to-IPv6 Network Prefix Translation (NPTv6) ([RFC6296]).

3.2.1 SIR to ILA address translation

When transmitting a packet, the locator for both the source and destination ILA addresses might need to be set before packet is sent on the wire. In the case that packet was created using a standard identifier representation, the SIR prefix is overridden with a locator. Since this operation is potentially done for every packet the process should be very efficient. Presumably, a host will maintain a cache of identifier locator mappings with a fast lookup function. If there is a connection state associated with the communication, the locator information may be cached with the connection state to obviate the need to perform a lookup per packet.

The typical steps to transmit a packet using ILA are:

1) Stack creates a packet with source address set to SIR address for the local identity, and the destination address is set to the SIR address for the peer. The peer SIR address may have been discovered through DNS or other means.

2) Stack overwrites the SIR prefix in the source address with an appropriate locator for the local host.

3) Stack overwrites the SIR prefix in the destination address with a locator for the peer. This locator is discovered by a lookup in the locator to identifier mappings.

4) If a transport checksum includes a pseudo header that covered the original addresses, the checksum needs to be updated. This should be akin to the checksum update needed in address translation for NAT ([RFC6296]).

5) Packet is sent on the wire. The network routes the packet to the host indicated by the locator.

3.2.2 ILA to SIR address translation

Upon reception, the identifier is used to match a valid address on the host or a connection context. In order to avoid having networking stack operate on a new address type, identifier-locator addresses may be translated to standard identifier representation addresses by overwriting the locator in the address with a SIR prefix.

Receive processing may be:

1) Packet is received, the destination locator matches an interface address prefix on the host.

2) A lookup is performed on the destination identifier to match to a local identifier. If the lookup is address based, the SIR address can be created for the destination (overwrite locator with a SIR prefix).

3) Perform any checks as necessary. Validate locators, identifiers, and check that packet is not illegitimately crossing virtual networks (see below).

4) Forward packet to application processing. If necessary, the addresses in the packet can be converted to SIR addresses in place. Changing the addresses may also entail updating the checksum to reflect that (again similar to a NAT translation).

3.3 Virtual networking operation

When using ILA with virtual networking identifiers, address translation is performed to convert tenant virtual network and virtual addresses to ILA addresses, and ILA addresses back to a virtual network and tenant’s virtual addresses. Address translation

is performed similar to the SIR translation cases described above.

A packet with virtual networking ILA addresses must be verified on reception. By default, the virtual network identifiers in the source and destination addresses must match or the packet is dropped. This would include the case that one address is using ILA with virtual network identifier and the other is not.

3.3.1 Crossing virtual networks

With explicit configuration, virtual network hosts may communicate directly with virtual hosts in another virtual network. This might be done to allow services in one virtual network to be accessed from another (by prior agreement between tenants). In this case, the virtual networking identifiers in the source and destination addresses won't match. This does require that identifiers are unique in a shared space.

3.3.2 IPv4/IPv6 protocol translation

An IPv4 tenant may send a packet that is converted to an IPv6 packet with ILA addresses having IPv4 virtual networking identifiers. Similarly, an IPv6 packet with ILA addresses may be converted to an IPv4 packet to be received by an IPv4-only tenant. These are IPv4/IPv6 stateless protocol translations as described in [RFC6144] and [RFC6145].

3.4 One sided ILA

It is not required that ILA be used for both and destination addresses. For instance a statically addressed server may provide service to virtual hosts or migratable jobs. Note that even though the server's address is static, locators for its ILA clients may change so the server will need identifier to locator mappings.

3.5 Checksum handling

TCP and UDP checksum includes a pseudo checksum that covers the IP addresses in a packet. In the case of identifier-locator addressing the checksum must include the actual addresses set in the packet on the wire. So when creating a checksum for transmit, or verifying a checksum on receive, identifier-locator addressing must be taken into account.

3.5.1 Transmit checksum

If the source and destination locators are available when the transport checksum is being set, these can be used to calculate the

pseudo checksum for the packet. This might be applicable in cases where locator information is cached within the context for a transport connection.

If the locators are set after the transport layer processing, the checksum can be updated following NAT procedures for address translation.

3.5.2 Receive checksum

Similar to the transmit case, if address translation occurs before transport layer processing the checksum must be adjusted per NAT. An implementation may verify a transport checksum before converting addresses to standard identifier representation to potentially obviate modifying the transport checksum to account for translation.

3.6 Address selection

There may be multiple possibilities for creating either a source or destination address. A node may be associated with more than one identifier, and there may be multiple locators for a particular identifier. The selection of an identifier occurs at flow creation and must be constant for the duration of the flow. Locator selection should be done once per flow, however may change (in the case of a migrating connection it will change). ILA address selection should follow guidelines in Default Address Selection for Internet Protocol Version 6 (IPv6) ([RFC6742]).

4. Communication scenarios

This section describes the use of identifier-locator addressing in several scenarios.

4.1 Terminology

A formal notation for identifier-locator addressing with ILNP is described in [RFC6740]. We extend this to include for network virtualization cases.

Basic terms are:

A = IP Address
I = Identifier
L = Locator
LUI = Locally unique identifier
VNI = Virtual network identifier
VA = An IPv4 or IPv6 virtual address
VAX = An IPv6 networking identifier (IPv6 VA mapped to VAX)

SIR = Prefix for standard identifier representation
EXA = An Internet routable prefix, may be use as a SIR
VNET = IPv6 prefix for a tenant

An ILA IPv6 address is denoted by

L:I

A transport endpoint IPv6 address with a locally unique identifier with SIR prefix is denoted by

SIR:LUI

A virtual identifier with a virtual network identifier and a virtual IPv4 address is denoted by

VNI:VA

An ILA IPv6 address with a virtual networking identifier for IPv4 would then be denoted

L:(VNI:VA)

The local and remote address pair in a packet or endpoint is denoted

A,A

An address translation sequence from transport visible addresses to ILA addresses for transmission on the network and back to transport endpoint addresses at the receiver has notation:

A,A -> L:I,L:I -> A,A

4.2 Identifier objects

Identifier-locator addressing is broad enough in scope to address may different types of networking objects within a data center. For descriptive purposes we classify these objects as tasks or tenant systems.

A task is a unit of execution that runs in the data center networks. These do not run in a virtual machine, but typically run in the native host context perhaps within containers. Task are the execution mechanism for native jobs in the data center.

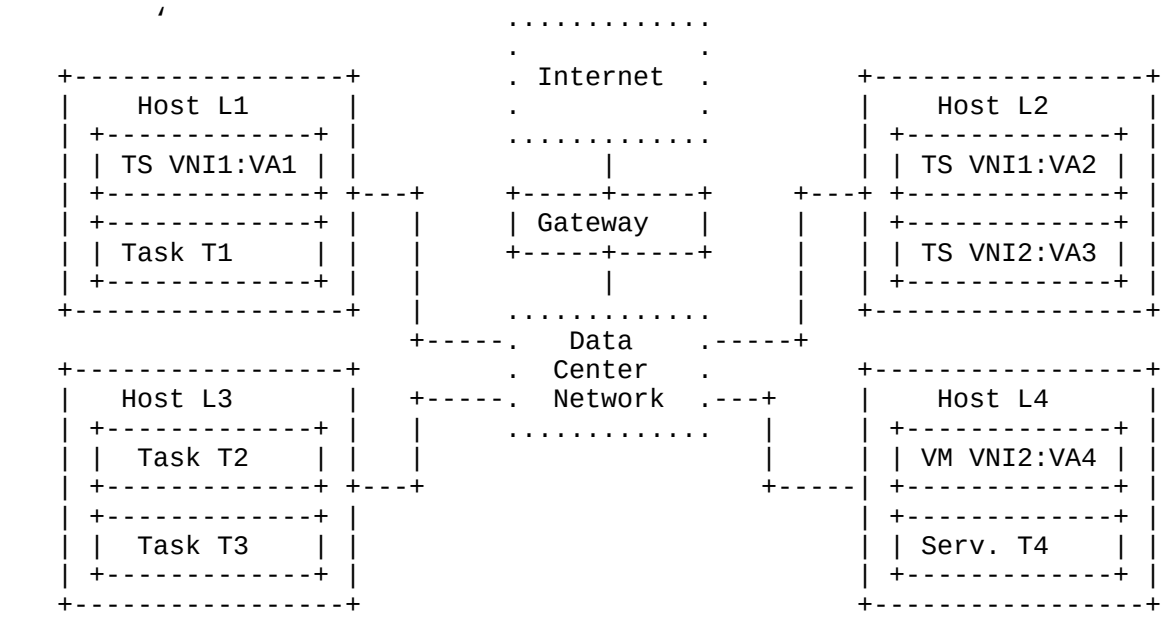
A tenant system, or TS, is a unit of execution which runs on behalf of a tenant in network virtualization. A TS may be implemented as a virtual machine or possibly using containers mechanisms. In either

case, a virtual overlay network is implemented on behalf of a tenant, and isolation between virtual networks is paramount.

A network service is a task that provides some network wide service such as DNS, remote storage, remote logging, etc. A network service may be accessed by tenant systems as well as other tasks.

4.2 Reference network for scenarios

The figure below provides an example network topology with ILA addressing in use. In this example, there are four hosts in the network with locators L1, L2, L3 , and L4. Three tasks with identifiers T1, T2, and T3 exist as well as a networking service task with identifier T4. The identifiers for these tasks may be locally unique identifiers. There are two virtual networks VNI1 and VNI2, and four tenant systems addressed as: VA1 and VA2 in VNI1, VA3 and VA4 in VNI2. The network is connected to the Internet via a gateway.



There are several communications scenario that can be considered:

1) Task to task (service)
2) Task to Internet
3) Internet to task
4) TS to service
5) Task to TS
6) TS to Internet

7) Internet to TS
8) IPv4 TS to service
9) TS to TS in same virtual network using IPv6
10) TS to TS in same virtual network using IPv4
11) TS to TS in different virtual network using IPv6
12) TS to TS in different virtual network using IPv4
13) IPv4 TS to IPv6 TS in different virtual networks

4.3 Scenario 1: Task to task

The transport endpoints for task to task communication are the SIR addresses for the tasks. When a packet is sent on the wire, the locators are set in source and destination addresses of the packet. On reception the source and destination addresses are converted back to SIR representations for processing at the transport layer.

If task T1 is communicating with task T2, the ILA translation sequence would be:

```
SIR:T1,SIR:T2 ->                    // Transport endpoints on T1
L1:T1,L3:T2 ->                      // ILA used on the wire
SIR:T1,SIR:T2                       // Received at T2
```

4.4 Scenario 2: Task to Internet

Communication from a task to the Internet is accomplished through use of a gateway that translates the internal locator for the task source to an externally routable prefix.

If task T1 is sending to an address Iaddr on the Internet, the ILA translation sequence would be:

```
SIR:T1,Iaddr ->                     // Transport endpoints at T1
L1:T1,Iaddr ->                      // On the wire in data center
EXA:T1,Iaddr                        // In the Internet
```

EXA is a globally routable prefix usable on the Internet. On egress from the data center network, a gateway sets EXA in the source address. If the SIR prefix is globally routable then this may be the same as EXA.

4.5 Scenario 3: Internet to task

An Internet host transmits packet to a task using an externally routable prefix and an identifier. The subnet prefix routes the packet to a gateway for the data center. The gateway translates the destination to an ILA address.

If a host on the Internet with address Iaddr is sends a packet to task T3, the ILA translation sequence would be:

```
Iaddr,EXA:T3 ->                    // Transport endpoint at Iaddr
Iaddr,L1:T3 ->                     // On the wire in data center
Iaddr,SIR:T3                       // Received at T3
```

EXA is a globally routable prefix usable on the Internet. On ingress into the data center, a gateway overwrites this with a locator. If the SIR prefix for T3 is globally routable then this may be the same as EXA.

4.6 Scenario 4: TS to service task

A tenant can communicate with a data center service using the SIR address of the service. The source address is translated from the tenant's address and prefix to VNID and VADDR. Locators must be set properly for transmission.

If TS VA1 is communicating with service task T4, the ILA translation sequence would be:

```
VNET:VA1,SIR:T4->                  // Transport endpoints in TS
L1:(VNI1:VAX1),L3:T4->             // On the wire
SIR:(VNI1:VAX1),SIR:T4             // Received at T4
```

VNET is the address prefix for the tenant. Alternatively, the service may map the tenant's address to its SIR representation to use VNET for the endpoint:

```
VNET:VA1,SIR:T4->                  // Transport endpoints in TS
L1:(VNI1:VAX1),L3:T4->             // On the wire
VNET:VA1,SIR:T4                    // Received at T4
```

Note that from the service point of view there is no material difference between a peer that is a tenant system versus a peer that is a task.

4.7 Scenario 5: Task to TS

A task can communicate with a TS through it's externally visible address, or by its virtual networking identifier and virtual address.

If task T2 is communicating with TS VA4, the ILA translation sequence would be:

```
SIR:T2,SIR:(VNI2:VA4) ->           // Transport endpoints at T2
L3:T2,L4:(VNI2:VA4) ->             // On the wire
```

```
SIR:T2,VNET:VA4                  // Received at TS
```

Alternatively, the task can use the VNET prefix to address a TS:

```
SIR:T2,VNET:VA4 ->               // Transport endpoints at T2
L3:T2,L4:(VNI2:VAX4) ->          // On the wire
SIR:T2,VNET:VA4                  // Received at TS
```

4.8 Scenario 6: TS to Internet

Communication from a TS to the Internet is accomplished through use of a gateway that translates the locator in the TS's source address back to the tenant's prefix. This assumes that the tenant's prefix is properly routed to the data center network.

If TS VA4 transmits a packet to address Iaddr on the Internet, the ILA translation sequence would be:

```
VNET:VA4,Iaddr ->                // Transport endpoints at TS
L4:(VNI2:VAX4),Iaddr ->          // On the wire in data center
VNET:VA4,Iaddr                   // On the Internet
```

4.9 Scenario 7: Internet to TS

An Internet host transmits a packet to a tenant system using an externally routable tenant prefix and a tenant system identifier. The prefix routes the packet to a gateway for the data center. The gateway translates the destination to an ILA address.

If a host on the Internet with address Iaddr is sending to TS VA4, the ILA translation sequence would be:

```
Iaddr,VNET:VA4 ->                // Endpoint at Iaddr
Iaddr,L4:(VNI2:VAX4) ->          // On the wire in data center
Iaddr,VNET:VA4                   // Received at TS
```

4.10 Scenario 8: IPv4 TS to service

A TS that is IPv4-only may communicate with a data center network service using NAT protocol translation. The network service would represented as an IPv4 address in the tenant's address space, and stateless NAT64 should be usable as described in [RFC6145].

If TS VA2 communicates with service task T4, the ILA translation sequence would be:

```
VA2,ADDR4 ->                     // IPv4 endpoints at TS
L2:(VNI1:VA2),L4:T4 ->           // On the wire in data center
```

```
  SIR:(VNI1:VA2),SIR:T4              // Received at task
```

VA2 is the IPv4 address in the tenant's virtual network, ADDR4 is an address in the tenant's address space that maps to the network service.

The reverse path, task sending to a TS with an IPv4 address, requires a similar protocol translation.

For service task T4 to communicate with TS VA2, the ILA translation sequence would be:

```
  SIR:T4,SIR:(VNI1:VA2) ->          // Endpoints at T4
  L4:T4,L2:(VNI1:VA2)   ->          // On the wire in data center
  ADDR4,VA2 ->                      // IPv4 endpoint at TS
```

4.11 TS to TS in the same virtual network

ILA may be used to allow tenants within a virtual network to communicate without the need for explicit encapsulation headers.

4.11.1 Scenario 9: TS to TS in same VN using IPV6

If TS VA1 sends a packet to TS VA2, the ILA translation sequence would be:

```
  VNET:VA1,VNET:VA2 ->               // Endpoints at VA1
  L1:(VNI1:VAX1),L2:(VNI1,VAX2) ->   // On the wire
  VNET:VA1,VNET:VA2 ->               // Received at VA2
```

4.11.2 Scenario 10: TS to TS in same VN using IPv4

For two tenant systems to communicate using IPv4 and ILA, IPv4/IPv6 protocol translation is done both on the transmit and receive.

If TS VA1 sends an IPv4 packet to TS VA2, the ILA translation sequence would be:

```
  VA1,VA2 ->                         // Endpoints at VA1
  L1:(VNI1:VA1),L2:(VNI1,VA2) ->     // On the wire
  VA1,VA2                            // Received at VA2
```

4.12 TS to TS in a different virtual network

A tenant system may be allowed to communicate with another tenant system in a different virtual network. This should only be allowed with explicit policy configuration.

4.12.1 Scenario 11: TS to TS in a different VN using IPV6

For TS VA4 to communicate with TS VA1 using IPv6 the translation sequence would be:

```
VNET2:VA4,VNET1:VA1->               // Endpoints at VA4
L4:(VNI2:VA4),L1:(VNI1,VA1)->       // On the wire
SIR:VA4,VNET1:VA1                   // Received at VA1
```

Alternatively, the the VNET prefix can address a TS:

```
VNET2:VA4,VNET1:VA1->               // Endpoint at VA4
L4:(VNI2:VAX4),L1:(VNI1,VAX1)->     // On the wire
VNET2:VA4,VNET1:VA1                 // Received at VA1
```

4.12.2 Scenario 12: TS to TS in a different VN using IPv4

To allow IPv4 tenant systems in different virtual networks to communicate with each other, an address representing the peer would be mapped into the tenant's address space. IPv4/IPv6 protocol translation is done on transmit and receive.

For TS VA4 to communicate with TS VA1 using IPv4 the translation sequence may be:

```
VA4,SADDR1 ->                       // IPv4 endpoint at VA4
L4:(VNI2:VA4),L1:(VNI1,VA1)->       // On the wire
SADDR4,VA1                          // Received at VA1
```

SADDR1 is the mapped address for VA1 in VA4's address space, and SADDR4 is the mapped address for VA4 in VA1's address space.

4.12.3 Scenario 13: IPv4 TS to IPv6 TS in different VNs

Communication may also be mixed so that an IPv4 tenants system can communicate with an IPv6 tenant system in another virtual network. IPv4/IPv6 protocol translation is done on transmit.

For VM VA4 using IPv4 to communicate with VM VA1 using IPv6 the translation sequence may be:

```
VA4,SADDR1 ->                       // IPv4 endpoint at VA4
L4:(VNI2:VA4),L1:(VNI1,VAX1)->      // On the wire
SIR:VA4,VNET1:VA1                   // Received at VA1
```

Alternatively the task can use the VNET prefix to address a TS:

```
VA4,SADDR1 ->                       // IPv4 endpoint at VA4
```

```
L4:(VNI2:VA4),L1:(VNI1,VA1)->        // On the wire
VNET2:VA4,VNET1:VA1                  // Received at VA1
```

SADDR1 is the mapped IPv4 address for VA1 in VA4's address space.

5. Use cases

This section highlights some use cases for identifier-locator addressing.

5.1 Data center virtualization

A primary motivation for identifier-locator addressing is data center virtualization. Virtualization within a data center permits malleability and flexibility in using data center resources. In particular, identifier-locator addressing virtualizes networking to allow flexible job scheduling and possibility of live task migration.

5.1.1 Job scheduling

In the usual data center model, jobs are scheduled to run as tasks on some number of machines. A distributed job scheduler provides the scheduling which may entail considerable complexity since jobs will often have a variety of resource constraints. The scheduler takes these constraints into account while trying to maximize utility of the data center in terms utilization, cost, latency, etc. Data center jobs do not typically run in virtual machines (VMs), but may run within containers. Containers are mechanisms that provide resource isolation between tasks running on the same host OS. These resources can include CPU, disk, memory, and networking.

A fundamental problem arises in that once a task for a job is scheduled on a machine, it often needs to run to completion. If the scheduler needs to schedule a higher priority job or change resource allocations, there may be little recourse but to kill tasks and restart them on a different machine. In killing a task, progress is lost which results in increased latency and wasted CPU cycles. Some tasks may checkpoint progress to minimize the amount of progress lost, but this is not a very transparent or general solution.

An alternative approach is to allow transparent job migration. The scheduler may migrate running jobs from one machine to another.

Under the orchestration of the job scheduler, the steps to migrate a job may be:

1) Stop running tasks for the job.
2) Package the run time state of the job. The run time state is

derived from the containers for the jobs.
3) Send the run time state of the job to the new machine where the job is to run.
4) Instantiate the job's state on the new machine.
5) Start the tasks for the job continuing from the point at which it was stopped.

This model similar to virtual machine (VM) migration except that the run time state is typically much less data-- just task state as opposed to a full OS image. Task state may be compressed to reduce latency in migration.

The networking state of interest to migrate are the addresses used by the task and open transport connections.

5.1.1 Address migration

To allow for task migration, each migratable task is assigned a unique address which be moved to a new location at task migration.

With identifier-locator addressing, tasks are assigned locally unique identifiers (see below for assignment techniques). A LUI is combined with a SIR prefix to give each task its own IPv6 address. To communicate with a running task, the LUI is mapped to a locator which is placed in the on-the-wire packet as discussed above. When a task migrates to a new machine, the identifier to locator mapping for the task is updated to reflect the change.

5.1.2 Connection migration

When a task and its addresses are migrated between machines, the disposition of existing TCP connections needs to be considered.

The simplest course of action is to drop TCP connections across a migration. Since migrations should be relatively rare events, it is conceivable that TCP connections could be automatically closed in the network stack during a migration event. If the applications running are known to handle this gracefully (i.e. reopen dropped connections) then this may be viable.

For seamless migration, open connections may be migrated between hosts. Migration of these entails pausing the connection, packaging connection state and sending to target, instantiating connection state in the peer stack, and restarting the connection. From the time the connection is paused to the time it is running again in the new stack, packets received for the connection should be silently dropped. For some period of time, the old stack will need to keep a record of the migrated connection. If it receives a packet, it should

either silently drop the packet or forward it to the new location.

5.1.3 Task identifier generation

Potentially every task in a data center could be migratable as long as each task is assigned a unique identifier. Since the identifier is fifty-nine bits it is conceivable that identifiers could be allocated using a shared counter or based on a timestamp.

5.1.3.1 Gobally unique identifiers method

For small to moderate sized deployments the technique for creating locally assigned global identifiers described in [RFC4193] could be used. In this technique a SHA-1 digest of the time of day in NTP format and an EUI-64 identifier of the local host is performed. N bits of the result are used as the globally unique identifier.

The probability that two or more of these IDs will collide can be approximated using the formula:

```
P = 1 - exp(-N**2 / 2**(L+1))
```

where P is the probability of collision, N is the number of identifiers, and L is the length of an identifier.

The following table shows the probability of a collision for a range of identifiers using a 61-bit length.

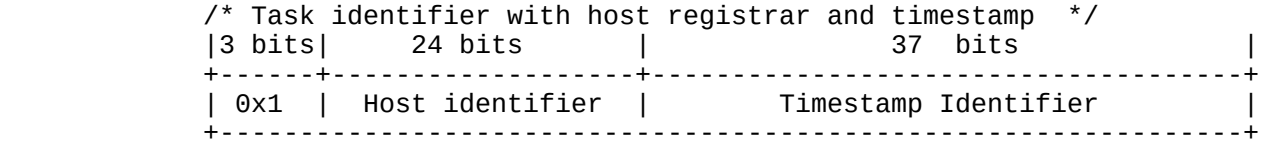
Identifiers	Probability of Collision
1000	2.1684*10^-13
10000	2.1684*10^-11
100000	2.1684*10^-09
1000000	2.1684*10^-07

Note that locally unique identifiers may be ephemeral, for instance a task may only exist for a few seconds. This should be considered when determining the probability of identifier collision.

5.1.3.2 Universally Unique Identifiers method

For larger deployments, hierarchical allocation may be desired. The techniques in Universally Unique IDentifier (UUID) URN ([RFC4122]) can be adapted for allocating unique task identifiers in sixty-one bits. An identifier is split into two components: a registrar prefix and sub-identifier. The registrar prefix defines an identifier block which is managed by the same host, the sub-identifier is a unique value within the registrar block.

For instance, a task identifier could be created on the initial running host that runs a task. The identifier could be composed of a 24 bit host identifier followed by a 37 bit timestamp. Assuming that a host can start up to 100 tasks per second, this allows 43.5 years before wrap around.



Hierarchical allocation may also be used to support hierarchical locator lookup.

5.1.3.3 Duplicate identifier detection

As part of implementing the locator to identifier mapping, duplicate identifier detection may be implemented in a centralized control plane. A registry of identifiers would be maintained. When a node creates an identifier it registers the identifier, and when the identifier is no longer in use (e.g. task completes) the identifier is unregistered. The control plane should able to detect a registration attempt for an existing identifier and deny the request.

5.2 Multi-tenant virtualization

Identifier-locator addressing may be used as an alternative to nvo3 encapsulation protocols (such as GUE [GUE]). In multi-tenant virtualization, overlay networks are established for various tenants to create virtual networks and a tenant’s nodes are assigned virtual addresses. Virtual networking identifiers are used to encode a virtual network identifier and a virtual address in an ILA address.

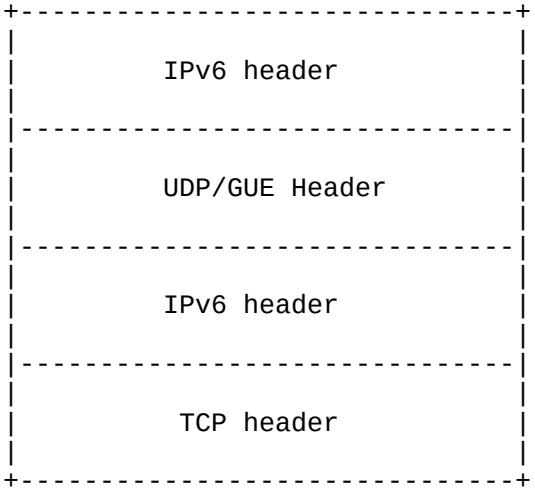
An advantage of identifier-locator addressing is that the overhead of encapsulation is reduced and use of virtualization can be transparent to the underlying network. A downside is that some features that use additional data in an encapsulation aren’t available (security option in GUE for instance [GUESEC]).

Identifier-locator addressing may be appropriate in network virtualization where the users are trusted, for instance if virtual networks were assigned to different departments within an enterprise. Network virtualization in this context provides a means of isolation of traffic belonging to different departments of a single tenant. If this isolation is broken and traffic illegitimately crosses between virtual networks, this is not considered a significant security risk.

The communication scenarios section above describes communication within a virtual network, communications with network services, and communication with hosts on the Internet.

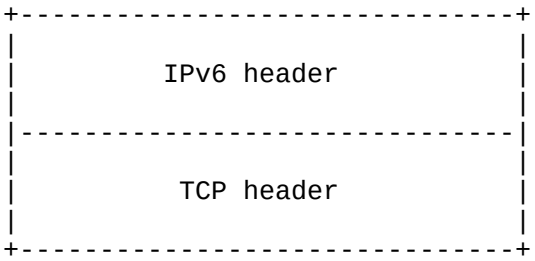
5.2.1 IPv6 over IPv6 network virtualization

In a canonical implementation of overlay networks for network virtualization, encapsulation headers are used between outer and IP inner headers which contains a virtual network identifier and possibly other data. Typical encapsulation of an IPv6 packet using GUE is illustrated below:



The addresses in the outer IPv6 header indicate the physical nodes (source and destination NVEs) in the network. The inner IPv6 addresses are IPv6 addresses within the virtual network specified by the VNID in the GUE header.

Using ILA eliminates the encapsulation headers and inner IP headers:



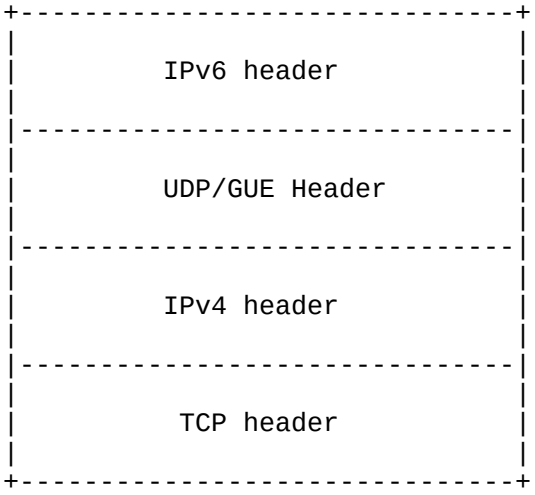
The IPv6 addresses are ILA addresses with virtual networking IPv6

identifiers. The encoded VNID indicates the virtual network the address belongs to, and the encoded VADDR provides the low order 32 bits of the virtual address for both source and destination. The tenant visible upper 96 bits of the IPv6 address is inferred from the VNID.

If the destination is multicast, the appropriate multicast identifier can be used in the destination address.

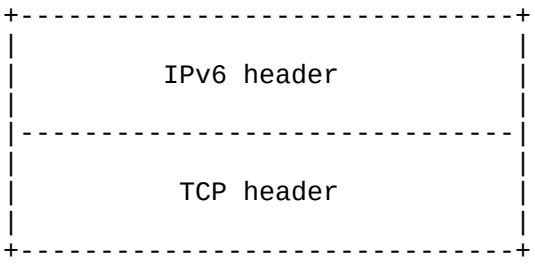
5.2.2 IPv4 over IPv6 network virtualization

The figure below illustrates the protocol headers when encapsulating a tenant's IPv4 packet using GUE.



The addresses in the outer IPv6 header indicate the physical nodes (source and destination NVEs) in the network. The inner IPv4 addresses are in the virtual network specified by the VNID in the GUE header.

Using ILA eliminates the encapsulation headers and inner IP headers:



The IPv6 addresses are ILA addresses with virtual networking IPv4 identifiers. The encoded VNID indicates the virtual network the addresses belongs to, and the encoded VADDRs provide the IPv4 virtual addresses for both source and destination. The IPv4 virtual address are visible to the tenant systems.

6 Security Considerations

Security must be considered when using identifier-locator addressing. In particular, the risk of address spoofing or address corruption must be addressed. To classify this risk the set possible destinations for a packet are classified as trusted or untrusted. The set of possible destinations includes those that a packet may inadvertently be sent due to address or header corruption.

If the set of possible destinations are trusted then packet misdelivery is considered relatively innocuous. This might be the case in a data center if all nodes were tightly controlled under single management. Identifier-locator addressing can be used this case without further additional security.

If the set of possible destinations are untrusted, then packet misdelivery is considered detrimental. This may be the case that virtual machines with third party applications and OS are running in the network. A malicious user may be snooping for misdelivered packets, or may attempt to spoof addresses. Identifier locator addressing should be used with stronger security and isolation mechanisms such as IPsec or GUESEC.

7 IANA Considerations

There are no IANA considerations in this specification.

8 References

8.1 Normative References

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, June 2011.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, September 2012.

8.2 Informative References

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC6740] RJ Atkinson and SN Bhatti, "Identifier-Locator Network Protocol (ILNP) Architectural Description", RFC 6740, November 2012.

[RFC6741] RJ Atkinson and SN Bhatti, "Identifier-Locator Network Protocol (ILNP) Engineering Considerations", RFC 6741, November 2012.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC3363] Bush, R., Durand, A., Fink, B., Gudmundsson, O., and T. Hain, "Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS)", RFC 3363, August 2002.

[RFC3587] Hinden, R., Deering, S., and E. Nordmark, "IPv6 Global Unicast Address Format", RFC 3587, August 2003.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.

[RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, April 2011.

[GUE] Herbert, T., and Yong, L., "Generic UDP Encapsulation", draft-herbert-gue-02, work in progress.

[GUESEC] Yong L., and Herbert, T. "Generic UDP Encapsulation (GUE) for Secure Transport", draft-hy-gue-4-secure-transport-00, work in progress

9 Acknowledgments

The authors would like to thank Mark Smith, Lucy Yong, and Erik Kline for their insightful comments for this draft; Roy Bryant, Lorenzo Colitti, Mahesh Bandewar, and Erik Kline for their work on defining

and applying ILA.

Authors' Addresses

Tom Herbert
Google
1600 Amphitheatre Parkway
Mountain View, CA
EMail: therbert@google.com

Internet Engineering Task Force
Internet Draft
Intended status: Informational
Expires: Oct 2014

Nabil Bitar
Verizon

Marc Lasserre
Florin Balus
Alcatel-Lucent

Thomas Morin
France Telecom Orange

Lizhong Jin

Bhumip Khasnabish
ZTE

April 15, 2014

NVO3 Data Plane Requirements
draft-ietf-nvo3-dataplane-requirements-03.txt

Status of this Memo

Copyright Notice

Abstract

Several IETF drafts relate to the use of overlay networks to support large scale virtual data centers. This draft provides a list of data plane requirements for Network Virtualization over L3 (NVO3) that have to be addressed in solutions documents.

Table of Contents

1. Introduction

1.1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

1.2. General terminology

The terminology defined in [NVO3-framework] is used throughout this document. Terminology specific to this memo is defined here and is introduced as needed in later sections.

BUM: Broadcast, Unknown Unicast, Multicast traffic

TS: Tenant System

2. Data Path Overview

The NVO3 framework [NVO3-framework] defines the generic NVE model depicted in Figure 1:

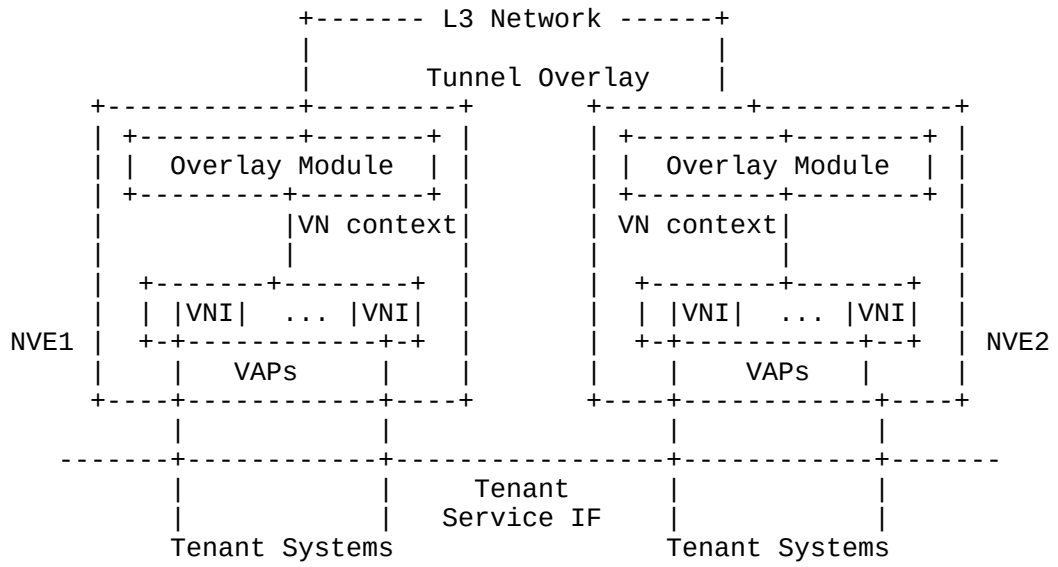


Figure 1 : Generic reference model for NV Edge

When a frame is received by an ingress NVE from a Tenant System over a local VAP, it needs to be parsed in order to identify which virtual network instance it belongs to. The parsing function can examine various fields in the data frame (e.g., VLANID) and/or associated interface/port the frame came from.

Once a corresponding VNI is identified, a lookup is performed to determine where the frame needs to be sent. This lookup can be based on any combinations of various fields in the data frame (e.g., destination MAC addresses and/or destination IP addresses). Note that additional criteria such as Ethernet 802.1p priorities and/or DSCP markings might be used to select an appropriate tunnel or local VAP destination.

Lookup tables can be populated using different techniques: data plane learning, management plane configuration, or a distributed control plane. Management and control planes are not in the scope of this document. The data plane based solution is described in this document as it has implications on the data plane processing function.

The result of this lookup yields the corresponding information needed to build the overlay header, as described in section 3.3. This information includes the destination L3 address of the egress NVE. Note that this lookup might yield a list of tunnels such as when ingress replication is used for BUM traffic.

The overlay header MUST include a context identifier which the egress NVE will use to identify which VNI this frame belongs to.

The egress NVE checks the context identifier and removes the encapsulation header and then forwards the original frame towards the appropriate recipient, usually a local VAP.

3. Data Plane Requirements

3.1. Virtual Access Points (VAPs)

The NVE forwarding plane MUST support VAP identification through the following mechanisms:

- Using the local interface on which the frames are received, where the local interface may be an internal, virtual port in a virtual switch or a physical port on a ToR switch
- Using the local interface and some fields in the frame header, e.g. one or multiple VLANs or the source MAC

3.2. Virtual Network Instance (VNI)

VAPs are associated with a specific VNI at service instantiation time.

A VNI identifies a per-tenant private context, i.e. per-tenant policies and a FIB table to allow overlapping address space between tenants.

There are different VNI types differentiated by the virtual network service they provide to Tenant Systems. Network virtualization can be provided by L2 and/or L3 VNIs.

3.2.1. L2 VNI

An L2 VNI MUST provide an emulated Ethernet multipoint service as if Tenant Systems are interconnected by a bridge (but instead by using a set of NVO3 tunnels). The emulated bridge could be 802.1Q enabled (allowing use of VLAN tags as a VAP). An L2 VNI provides per tenant virtual switching instance with MAC addressing isolation and L3 tunneling. Loop avoidance capability MUST be provided.

Forwarding table entries provide mapping information between tenant system MAC addresses and VAPs on directly connected VNIs and L3 tunnel destination addresses over the overlay. Such entries could be populated by a control or management plane, or via data plane.

Unless a control plane is used to disseminate address mappings, data plane learning MUST be used to populate forwarding tables. As frames arrive from VAPs or from overlay tunnels, standard MAC learning procedures are used: The tenant system source MAC address is learned against the VAP or the NVO3 tunneling encapsulation source address on which the frame arrived. Data plane learning implies that unknown unicast traffic will be flooded (i.e. broadcast).

When flooding is required, either to deliver unknown unicast, or broadcast or multicast traffic, the NVE MUST either support ingress replication or multicast.

When using underlay multicast, the NVE MUST have one or more underlay multicast trees that can be used by local VNIs for flooding to NVEs belonging to the same VN. For each VNI, there is at least one underlay flooding tree used for Broadcast, Unknown Unicast and Multicast forwarding. This tree MAY be shared across VNIs. The flooding tree is equivalent with a multicast (*,G) construct where all the NVEs for which the corresponding VNI is instantiated are members.

When tenant multicast is supported, it SHOULD also be possible to select whether the NVE provides optimized underlay multicast trees inside the VNI for individual tenant multicast groups or whether the default VNI flooding tree is used. If the former option is selected the VNI SHOULD be able to snoop IGMP/MLD messages in order to efficiently join/prune Tenant System from multicast trees.

3.2.2. L3 VNI

L3 VNIs MUST provide virtualized IP routing and forwarding. L3 VNIs MUST support per-tenant forwarding instance with IP addressing isolation and L3 tunneling for interconnecting instances of the same VNI on NVEs.

In the case of L3 VNI, the inner TTL field MUST be decremented by (at least) 1 as if the NVO3 egress NVE was one (or more) hop(s) away. The TTL field in the outer IP header MUST be set to a value appropriate for delivery of the encapsulated frame to the tunnel exit point. Thus, the default behavior MUST be the TTL pipe model where the overlay network looks like one hop to the sending NVE. Configuration of a "uniform" TTL model where the outer tunnel TTL is

set equal to the inner TTL on ingress NVE and the inner TTL is set to the outer TTL value on egress MAY be supported. [RFC2983] provides additional details on the uniform and pipe models.

L2 and L3 VNIs can be deployed in isolation or in combination to optimize traffic flows per tenant across the overlay network. For example, an L2 VNI may be configured across a number of NVEs to offer L2 multi-point service connectivity while a L3 VNI can be co-located to offer local routing capabilities and gateway functionality. In addition, integrated routing and bridging per tenant MAY be supported on an NVE. An instantiation of such service may be realized by interconnecting an L2 VNI as access to an L3 VNI on the NVE.

When underlay multicast is supported, it MAY be possible to select whether the NVE provides optimized underlay multicast trees inside the VNI for individual tenant multicast groups or whether a default underlay VNI multicasting tree, where all the NVEs of the corresponding VNI are members, is used.

3.3. Overlay Module

The overlay module performs a number of functions related to NVO3 header and tunnel processing.

The following figure shows a generic NVO3 encapsulated frame:

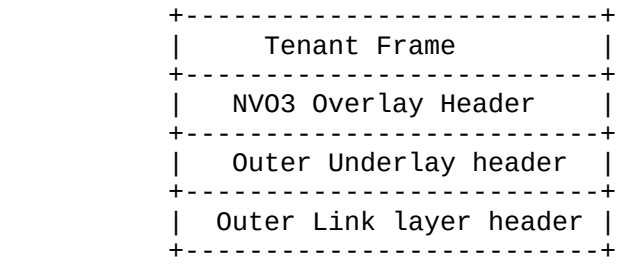


Figure 2 : NVO3 encapsulated frame

where

. Tenant frame: Ethernet or IP based upon the VNI type

. NVO3 overlay header: Header containing VNI context information and other optional fields that can be used for processing this packet.

. Outer underlay header: Can be either IP or MPLS

. Outer link layer header: Header specific to the physical transmission link used

3.3.1. NVO3 overlay header

An NVO3 overlay header MUST be included after the underlay tunnel header when forwarding tenant traffic.

Note that this information can be carried within existing protocol headers (when overloading of specific fields is possible) or within a separate header.

3.3.1.1. Virtual Network Context Identification

The overlay encapsulation header MUST contain a field which allows the encapsulated frame to be delivered to the appropriate virtual network endpoint by the egress NVE.

The egress NVE uses this field to determine the appropriate virtual network context in which to process the packet. This field MAY be an explicit, unique (to the administrative domain) virtual network identifier (VNID) or MAY express the necessary context information in other ways (e.g. a locally significant identifier).

In the case of a global identifier, this field MUST be large enough to scale to 100’s of thousands of virtual networks. Note that there is typically no such constraint when using a local identifier.

3.3.1.2. Quality of Service (QoS) identifier

Traffic flows originating from different applications could rely on differentiated forwarding treatment to meet end-to-end availability and performance objectives. Such applications may span across one or more overlay networks. To enable such treatment, support for multiple Classes of Service (Cos) across or between overlay networks MAY be required.

To effectively enforce CoS across or between overlay networks without Deep Packet Inspection (DPI) repeat, NVEs MAY be able to map

CoS markings between networking layers, e.g., Tenant Systems, Overlays, and/or Underlay, enabling each networking layer to independently enforce its own CoS policies. For example:

- TS (e.g. VM) CoS

    - Tenant CoS policies MAY be defined by Tenant administrators

    - QoS fields (e.g. IP DSCP and/or Ethernet 802.1p) in the tenant frame are used to indicate application level CoS requirements

- NVE CoS: Support for NVE Service CoS MAY be provided through a QoS field, inside the NVO3 overlay header

    - NVE MAY classify packets based on Tenant CoS markings or other mechanisms (eg. DPI) to identify the proper service CoS to be applied across the overlay network

    - NVE service CoS levels are normalized to a common set (for example 8 levels) across multiple tenants; NVE uses per tenant policies to map Tenant CoS to the normalized service CoS fields in the NVO3 header

- Underlay CoS

    - The underlay/core network MAY use a different CoS set (for example 4 levels) than the NVE CoS as the core devices MAY have different QoS capabilities compared with NVEs.

    - The Underlay CoS MAY also change as the NVO3 tunnels pass between different domains.

3.3.2. Tunneling function

This section describes the underlay tunneling requirements. From an encapsulation perspective, IPv4 or IPv6 MUST be supported, both IPv4 and IPv6 SHOULD be supported, MPLS MAY be supported.

3.3.2.1. LAG and ECMP

For performance reasons, multipath over LAG and ECMP paths MAY be supported.

LAG (Link Aggregation Group) [IEEE 802.1AX-2008] and ECMP (Equal Cost Multi Path) are commonly used techniques to perform load-balancing of microflows over a set of a parallel links either at

Layer-2 (LAG) or Layer-3 (ECMP). Existing deployed hardware implementations of LAG and ECMP uses a hash of various fields in the encapsulation (outermost) header(s) (e.g. source and destination MAC addresses for non-IP traffic, source and destination IP addresses, L4 protocol, L4 source and destination port numbers, etc). Furthermore, hardware deployed for the underlay network(s) will be most often unaware of the carried, innermost L2 frames or L3 packets transmitted by the TS.

Thus, in order to perform fine-grained load-balancing over LAG and ECMP paths in the underlying network, the encapsulation needs to present sufficient entropy to exercise all paths through several LAG/ECMP hops.

The entropy information can be inferred from the NVO3 overlay header or underlay header. If the overlay protocol does not support the necessary entropy information or the switches/routers in the underlay do not support parsing of the additional entropy information in the overlay header, underlay switches and routers should be programmable, i.e. select the appropriate fields in the underlay header for hash calculation based on the type of overlay header.

All packets that belong to a specific flow MUST follow the same path in order to prevent packet re-ordering. This is typically achieved by ensuring that the fields used for hashing are identical for a given flow.

The goal is for all paths available to the overlay network to be used efficiently. Different flows should be distributed as evenly as possible across multiple underlay network paths. For instance, this can be achieved by ensuring that some fields used for hashing are randomly generated.

3.3.2.2. DiffServ and ECN marking

When traffic is encapsulated in a tunnel header, there are numerous options as to how the Diffserv Code-Point (DSCP) and Explicit Congestion Notification (ECN) markings are set in the outer header and propagated to the inner header on decapsulation.

[RFC2983] defines two modes for mapping the DSCP markings from inner to outer headers and vice versa. The Uniform model copies the inner DSCP marking to the outer header on tunnel ingress, and copies that outer header value back to the inner header at tunnel egress. The Pipe model sets the DSCP value to some value based on local policy

at ingress and does not modify the inner header on egress. Both models SHOULD be supported.

[RFC6040] defines ECN marking and processing for IP tunnels.

3.3.2.3. Handling of BUM traffic

NVO3 data plane support for either ingress replication or point-to-multipoint tunnels is required to send traffic destined to multiple locations on a per-VNI basis (e.g. L2/L3 multicast traffic, L2 broadcast and unknown unicast traffic). It is possible that both methods be used simultaneously.

There is a bandwidth vs state trade-off between the two approaches. User-configurable settings MUST be provided to select which method(s) gets used based upon the amount of replication required (i.e. the number of hosts per group), the amount of multicast state to maintain, the duration of multicast flows and the scalability of multicast protocols.

When ingress replication is used, NVEs MUST maintain for each VNI the related tunnel endpoints to which it needs to replicate the frame.

For point-to-multipoint tunnels, the bandwidth efficiency is increased at the cost of more state in the Core nodes. The ability to auto-discover or pre-provision the mapping between VNI multicast trees to related tunnel endpoints at the NVE and/or throughout the core SHOULD be supported.

3.4. External NVO3 connectivity

It is important that NVO3 services interoperate with current VPN and Internet services. This may happen inside one DC during a migration phase or as NVO3 services are delivered to the outside world via Internet or VPN gateways (GW).

Moreover the compute and storage services delivered by a NVO3 domain may span multiple DCs requiring Inter-DC connectivity. From a DC perspective a set of GW devices are required in all of these cases albeit with different functionalities influenced by the overlay type across the WAN, the service type and the DC network technologies used at each DC site.

A GW handling the connectivity between NVO3 and external domains represents a single point of failure that may affect multiple tenant

services. Redundancy between NVO3 and external domains MUST be supported.

3.4.1. Gateway (GW) Types

3.4.1.1. VPN and Internet GWs

Tenant sites may be already interconnected using one of the existing VPN services and technologies (VPLS or IP VPN). If a new NVO3 encapsulation is used, a VPN GW is required to forward traffic between NVO3 and VPN domains. Internet connected Tenants require translation from NVO3 encapsulation to IP in the NVO3 gateway. The translation function SHOULD minimize provisioning touches.

3.4.1.2. Inter-DC GW

Inter-DC connectivity MAY be required to provide support for features like disaster prevention or compute load re-distribution. This MAY be provided via a set of gateways interconnected through a WAN. This type of connectivity MAY be provided either through extension of the NVO3 tunneling domain or via VPN GWs.

3.4.1.3. Intra-DC gateways

Even within one DC there may be End Devices that do not support NVO3 encapsulation, for example bare metal servers, hardware appliances and storage. A gateway device, e.g. a ToR switch, is required to translate the NVO3 to Ethernet VLAN encapsulation.

3.4.2. Path optimality between NVEs and Gateways

Within an NVO3 overlay, a default assumption is that NVO3 traffic will be equally load-balanced across the underlying network consisting of LAG and/or ECMP paths. This assumption is valid only as long as: a) all traffic is load-balanced equally among each of the component-links and paths; and, b) each of the component-links/paths is of identical capacity. During the course of normal operation of the underlying network, it is possible that one, or more, of the component-links/paths of a LAG may be taken out-of-service in order to be repaired, e.g.: due to hardware failure of cabling, optics, etc. In such cases, the administrator may configure the underlying network such that an entire LAG bundle in the underlying network will be reported as operationally down if there is a failure of any single component-link member of the LAG bundle, (e.g.: N = M configuration of the LAG bundle), and, thus, they know that traffic will be carried sufficiently by alternate, available (potentially ECMP) paths in the underlying network. This is a likely

an adequate assumption for Intra-DC traffic where presumably the costs for additional, protection capacity along alternate paths is not cost-prohibitive. In this case, there are no additional requirements on NVO3 solutions to accommodate this type of underlying network configuration and administration.

There is a similar case with ECMP, used Intra-DC, where failure of a single component-path of an ECMP group would result in traffic shifting onto the surviving members of the ECMP group. Unfortunately, there are no automatic recovery methods in IP routing protocols to detect a simultaneous failure of more than one component-path in a ECMP group, operationally disable the entire ECMP group and allow traffic to shift onto alternative paths. This problem is attributable to the underlying network and, thus, out-of-scope of any NVO3 solutions.

On the other hand, for Inter-DC and DC to External Network cases that use a WAN, the costs of the underlying network and/or service (e.g.: IPVPN service) are more expensive; therefore, there is a requirement on administrators to both: a) ensure high availability (active-backup failover or active-active load-balancing); and, b) maintaining substantial utilization of the WAN transport capacity at nearly all times, particularly in the case of active-active load-balancing. With respect to the dataplane requirements of NVO3 solutions, in the case of active-backup fail-over, all of the ingress NVE's need to dynamically adapt to the failure of an active NVE GW when the backup NVE GW announces itself into the NVO3 overlay immediately following a failure of the previously active NVE GW and update their forwarding tables accordingly, (e.g.: perhaps through dataplane learning and/or translation of a gratuitous ARP, IPv6 Router Advertisement). Note that active-backup fail-over could be used to accomplish a crude form of load-balancing by, for example, manually configuring each tenant to use a different NVE GW, in a round-robin fashion.

3.4.2.1. Load-balancing

When using active-active load-balancing across physically separate NVE GW's (e.g.: two, separate chassis) an NVO3 solution SHOULD support forwarding tables that can simultaneously map a single egress NVE to more than one NVO3 tunnels. The granularity of such mappings, in both active-backup and active-active, MUST be specific to each tenant.

3.4.2.2. Triangular Routing Issues

L2/ELAN over NVO3 service may span multiple racks distributed across different DC regions. Multiple ELANs belonging to one tenant may be interconnected or connected to the outside world through multiple Router/VRF gateways distributed throughout the DC regions. In this scenario, without aid from an NVO3 or other type of solution, traffic from an ingress NVE destined to External gateways will take a non-optimal path that will result in higher latency and costs, (since it is using more expensive resources of a WAN). In the case of traffic from an IP/MPLS network destined toward the entrance to an NVO3 overlay, well-known IP routing techniques MAY be used to optimize traffic into the NVO3 overlay, (at the expense of additional routes in the IP/MPLS network). In summary, these issues are well known as triangular routing (a.k.a. traffic tromboning).

Procedures for gateway selection to avoid triangular routing issues SHOULD be provided.

The details of such procedures are, most likely, part of the NVO3 Management and/or Control Plane requirements and, thus, out of scope of this document. However, a key requirement on the dataplane of any NVO3 solution to avoid triangular routing is stated above, in Section 3.4.2, with respect to active-active load-balancing. More specifically, an NVO3 solution SHOULD support forwarding tables that can simultaneously map a single egress NVE to more than one NVO3 tunnel.

The expectation is that, through the Control and/or Management Planes, this mapping information may be dynamically manipulated to, for example, provide the closest geographic and/or topological exit point (egress NVE) for each ingress NVE.

3.5. Path MTU

The tunnel overlay header can cause the MTU of the path to the egress tunnel endpoint to be exceeded.

IP fragmentation SHOULD be avoided for performance reasons.

The interface MTU as seen by a Tenant System SHOULD be adjusted such that no fragmentation is needed. This can be achieved by configuration or be discovered dynamically.

Either of the following options MUST be supported:

- o Classical ICMP-based MTU Path Discovery [RFC1191] [RFC1981] or Extended MTU Path Discovery techniques such as defined in [RFC4821]

- o Segmentation and reassembly support from the overlay layer operations without relying on the Tenant Systems to know about the end-to-end MTU

- o The underlay network MAY be designed in such a way that the MTU can accommodate the extra tunnel overhead.

3.6. Hierarchical NVE dataplane requirements

It might be desirable to support the concept of hierarchical NVEs, such as spoke NVEs and hub NVEs, in order to address possible NVE performance limitations and service connectivity optimizations.

For instance, spoke NVE functionality may be used when processing capabilities are limited. In this case, a hub NVE MUST provide additional data processing capabilities such as packet replication.

3.7. Other considerations

3.7.1. Data Plane Optimizations

Data plane forwarding and encapsulation choices SHOULD consider the limitation of possible NVE implementations, specifically in software based implementations (e.g. servers running virtual switches)

NVE SHOULD provide efficient processing of traffic. For instance, packet alignment, the use of offsets to minimize header parsing, padding techniques SHOULD be considered when designing NVO3 encapsulation types.

The NV03 encapsulation/decapsulation processing in software-based NVEs SHOULD make use of hardware assist provided by NICs in order to speed up packet processing.

3.7.2. NVE location trade-offs

In the case of DC traffic, traffic originated from a VM is native Ethernet traffic. This traffic can be switched by a local VM switch or ToR switch and then by a DC gateway. The NVE function can be embedded within any of these elements.

The NVE function can be supported in various DC network elements such as a VM, VM switch, ToR switch or DC GW.

The following criteria SHOULD be considered when deciding where the NVE processing boundary happens:

- Processing and memory requirements
  - Datapath (e.g. lookups, filtering, encapsulation/decapsulation)
  - Control plane processing (e.g. routing, signaling, OAM)
- FIB/RIB size
- Multicast support
  - Routing protocols
  - Packet replication capability
- Fragmentation support
- QoS transparency
- Resiliency

4. Security Considerations

This requirements document does not raise in itself any specific security issues.

5. IANA Considerations

IANA does not need to take any action for this draft.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

[NVOPS] Narten, T. et al, "Problem Statement: Overlays for Network Virtualization", draft-narten-nvo3-overlay-problem-statement (work in progress)

[NVO3-framework] Lasserre, M. et al, "Framework for DC Network Virtualization", draft-lasserre-nvo3-framework (work in progress)

[OVCPREQ] Kreeger, L. et al, "Network Virtualization Overlay Control Protocol Requirements", draft-kreeger-nvo3-overlay-cp (work in progress)

[FLOYD] Sally Floyd, Allyn Romanow, "Dynamics of TCP Traffic over ATM Networks", IEEE JSAC, V. 13 N. 4, May 1995

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, February 2006.

[RFC1191] Mogul, J. "Path MTU Discovery", RFC1191, November 1990

[RFC1981] McCann, J. et al, "Path MTU Discovery for IPv6", RFC1981, August 1996

[RFC4821] Mathis, M. et al, "Packetization Layer Path MTU Discovery", RFC4821, March 2007

[RFC2983] Black, D. "Diffserv and tunnels", RFC2983, Cotober 2000

[RFC6040] Briscoe, B. "Tunnelling of Explicit Congestion Notification", RFC6040, November 2010

[RFC6438] Carpenter, B. et al, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC6438, November 2011

[RFC6391] Bryant, S. et al, "Flow-Aware Transport of Pseudowires over an MPLS Packet Switched Network", RFC6391, November 2011

7. Acknowledgments

In addition to the authors the following people have contributed to this document:

Shane Amante, David Black, Dimitrios Stiliadis, Rotem Salomonovitch, Larry Kreeger, Eric Gray and Erik Nordmark.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Nabil Bitar
Verizon
40 Sylvan Road
Waltham, MA 02145
Email: nabil.bitar@verizon.com

Marc Lasserre
Alcatel-Lucent
Email: marc.lasserre@alcatel-lucent.com

Florin Balus
Alcatel-Lucent
777 E. Middlefield Road
Mountain View, CA, USA 94043
Email: florin.balus@alcatel-lucent.com

Thomas Morin
France Telecom Orange
Email: thomas.morin@orange.com

Lizhong Jin
Email : lizho.jin@gmail.com

Bhumip Khasnabish
ZTE
Email : Bhumip.khasnabish@zteusa.com

NVO3 Working Group
INTERNET-DRAFT
Intended Status: Informational

Yizhou Li
Lucy Yong
Huawei Technologies
Lawrence Kreeger
Cisco
Thomas Narten
IBM
David Black
EMC

Expires: August 13, 2015
February 9, 2015

Hypervisor to NVE Control Plane Requirements
draft-ietf-nvo3-hpvr2nve-cp-req-02

Abstract

In a Split-NVE architructure, the functions of the NVE are split across the hypervisor/container on a server and an external network equipment which is called an external NVE. A control plane protocol(s) between a hypervisor and its associated external NVE(s) is used for the hypervisor to distribute its virtual machine networking state to the external NVE(s) for further handling. This document illustrates the functionality required by this type of control plane signaling protocol and outlines the high level requirements. Virtual machine states as well as state transitioning are summarized to help clarifying the needed protocol requirements.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Table of Contents

1. Introduction

In the Split-NVE architecture shown in Figure 1, the functionality of the NVE is split across an end device supporting virtualization and an external network device which is called an external NVE. The portion of the NVE functionality located on the hypervisor/container is called the tNVE and the portion located on the external NVE is called the nNVE in this document. Overlay encapsulation/decapsulation functions are normally off-loaded to the nNVE on the external NVE. The tNVE is normally implemented as a part of hypervisor or container in an virtualized end device.

The problem statement [RFC7364], discusses the needs for a control plane protocol (or protocols) to populate each NVE with the state needed to perform the required functions. In one scenario, an NVE provides overlay encapsulation/decapsulation packet forwarding services to Tenant Systems (TSs) that are co-resident within the NVE on the same End Device (e.g. when the NVE is embedded within a hypervisor or a Network Service Appliance). In such cases, there is no need for a standardized protocol between the hypervisor and NVE, as the interaction is implemented via software on a single device. While in the Split-NVE architecture scenarios, as shown in figure 2 to figure 4, a control plane protocol(s) between a hypervisor and its associated external NVE(s) is required for the hypervisor to distribute the virtual machines networking states to the NVE(s) for further handling. The protocol indeed is an NVE-internal protocol and runs between tNVE and nNVE logical entities. This protocol is mentioned in NVO3 problem statement [RFC7364] and appears as the third work item.

Virtual machine states and state transitioning are summarized in this document to show events where the NVE needs to take specific actions. Such events might correspond to actions the control plane signaling protocols between the hypervisor and external NVE will need to take. Then the high level requirements to be fulfilled are outlined.

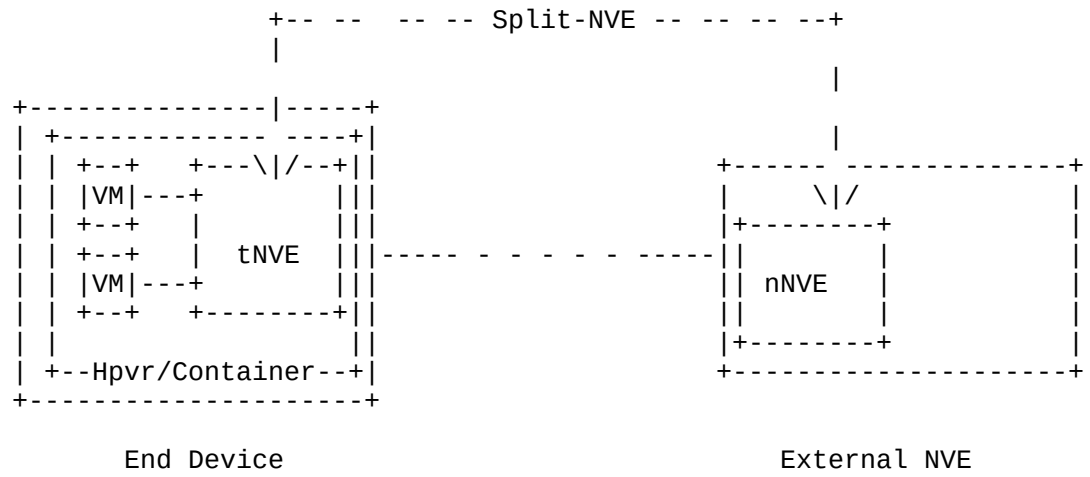


Figure 1 Split-NVE structure

This document uses the term "hypervisor" throughout when describing the Split-NVE scenario where part of the NVE functionality is off-loaded to a separate device from the "hypervisor" that contains a VM connected to a VN. In this context, the term "hypervisor" is meant to cover any device type where part of the NVE functionality is off-loaded in this fashion, e.g.,a Network Service Appliance, Linux Container.

This document often uses the term "VM" and "Tenant System" (TS) interchangeably, even though a VM is just one type of Tenant System that may connect to a VN. For example, a service instance within a Network Service Appliance may be another type of TS, or a system running on an OS-level virtualization technologies like LinuX Containers. When this document uses the term VM, it will in most cases apply to other types of TSs.

Section 2 describes VM states and state transitioning in its lifecycle. Section 3 introduces Hypervisor-to-NVE control plane protocol functionality derived from VM operations and network events. Section 4 outlines the requirements of the control plane protocol to achieve the required functionality.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this

document are to be interpreted as described in RFC 2119 [RFC2119].

This document uses the same terminology as found in [RFC7365] and [I-D.ietf-nvo3-nve-nva-cp-req]. This section defines additional terminology used by this document.

Split-NVE: a type of NVE that the functionalities of it are split across an end device supporting virtualization and an external network device.

tNVE: the portion of Split-NVE functionalities located on the end device supporting virtualization.

nNVE: the portion of Split-NVE functionalities located on the network device which is directly or indirectly connects to the end device holding the corresponding tNVE.

External NVE: the physical network device holding nNVE

Hypervisor/Container: the logical collection of software, firmware and/or hardware that allows the creation and running of server or service appliance virtualization. tNVE is located on Hypervisor/Container. It is loosely used in this document to refer to the end device supporting the virtualization. For simplicity, we also use Hypervisor in this document to represent both hypervisor and container.

VN Profile: Meta data associated with a VN that is applied to any attachment point to the VN. That is, VAP properties that are appliaed to all VAPs associated with a given VN and used by an NVE when ingressing/egressing packets to/from a specific VN. Meta data could include such information as ACLs, QoS settings, etc. The VN Profile contains parameters that apply to the VN as a whole. Control protocols between the NVE and NVA could use the VN ID or VN Name to obtain the VN Profile.

VSI: Virtual Station Interface. [IEEE 802.1Qbg]

VDP: VSI Discovery and Configuration Protocol [IEEE 802.1Qbg]

1.2 Target Scenarios

In the Split-NVE architecture, an external NVE can provide an offload of the encapsulation / decapsulation function, network policy enforcement, as well as the VN Overlay protocol overhead. This offloading may provide performance improvements and/or resource savings to the End Device (e.g. hypervisor) making use of the

external NVE.

The following figures give example scenarios of a Split-NVE architecture.

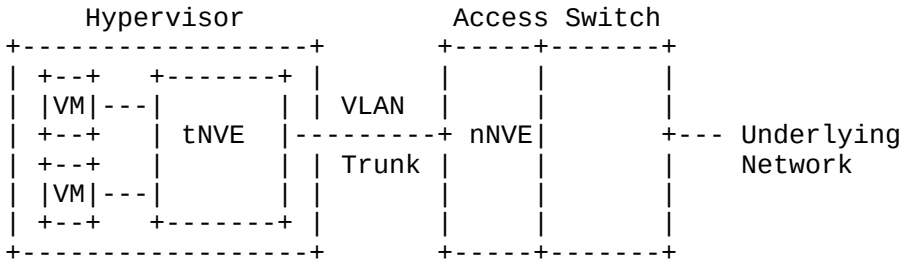


Figure 2 Hypervisor with an External NVE

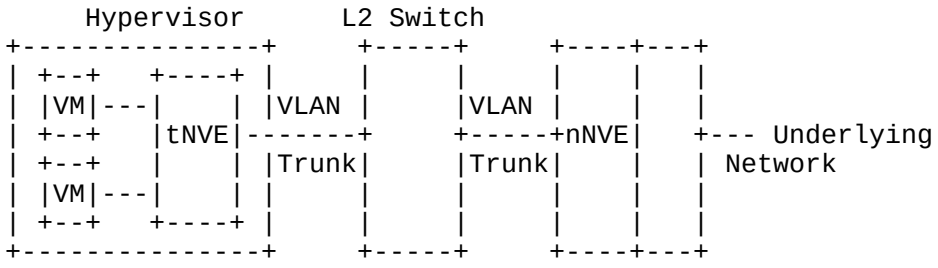


Figure 3 Hypervisor with an External NVE across an Ethernet Access Switch

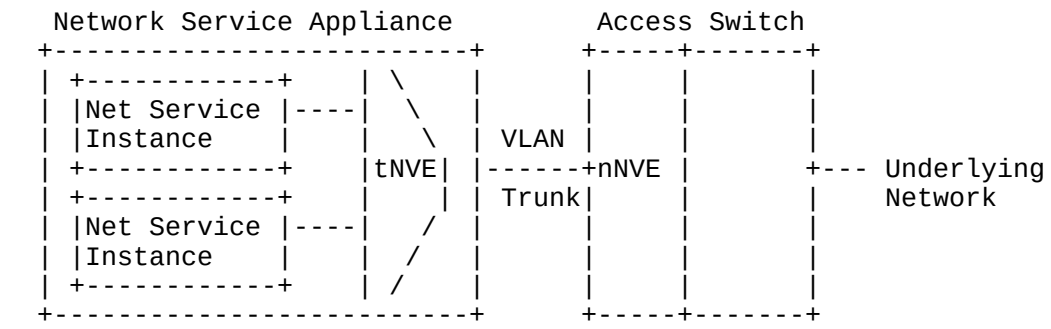


Figure 4 Physical Network Service Appliance with an External NVE

Tenant Systems connect to external NVEs via a Tenant System Interface

(TSI). The TSI logically connects to the external NVE via a Virtual Access Point (VAP) [I-D.ietf-nvo3-arch]. The external NVE may provide Layer 2 or Layer 3 forwarding. In the Split-NVE architecture, the external NVE may be able to reach multiple MAC and IP addresses via a TSI. For example, Tenant Systems that are providing network services (such as transparent firewall, load balancer, VPN gateway) are likely to have complex address hierarchy. This implies that if a given TSI disassociates from one VN, all the MAC and/or IP addresses are also disassociated. There is no need to signal the deletion of every MAC or IP when the TSI is brought down or deleted. In the majority of cases, a VM will be acting as a simple host that will have a single TSI and single MAC and IP visible to the external NVE.

2. VM Lifecycle

Figure 2 of [I-D.ietf-opsawg-vmm-mib] shows the state transition of a VM. Some of the VM states are of interest to the external NVE. This section illustrates the relevant phases and events in the VM lifecycle. It should be noted that the following subsections do not give an exhaustive traversal of VM lifecycle state. They are intended as the illustrative examples which are relevant to Split-NVE architecture, not as prescriptive text; the goal is to capture sufficient detail to set a context for the signaling protocol functionality and requirements described in the following sections.

2.1 VM Creation Event

VM creation event makes the VM state transiting from Preparing to Shutdown and then to Running [I-D.ietf-opsawg-vmm-mib]. The end device allocates and initializes local virtual resources like storage in the VM Preparing state. In Shutdown state, the VM has everything ready except that CPU execution is not scheduled by the hypervisor and VM’s memory is not resident in the hypervisor. From the Shutdown state to Running state, normally it requires the human execution or system triggered event. Running state indicates the VM is in the normal execution state. As part of transitioning the VM to the Running state, the hypervisor must also provision network connectivity for the VM’s TSI(s) so that Ethernet frames can be sent and received correctly. No ongoing migration, suspension or shutdown is in process.

In the VM creation phase, the VM’s TSI has to be associated with the external NVE. Association here indicates that hypervisor and the external NVE have signaled each other and reached some agreement. Relevant networking parameters or information have been provisioned properly. The External NVE should be informed of the VM’s TSI MAC address and/or IP address. In addition to external network

connectivity, the hypervisor may provide local network connectivity between the VM's TSI and other VM's TSI that are co-resident on the same hypervisor. When the intra or inter-hypervisor connectivity is extended to the external NVE, a locally significant tag, e.g. VLAN ID, should be used between the hypervisor and the external NVE to differentiate each VN's traffic. Both the hypervisor and external NVE sides must agree on that tag value for traffic identification, isolation and forwarding.

The external NVE may need to do some preparation work before it signals successful association with TSI. Such preparation work may include locally saving the states and binding information of the tenant system interface and its VN, communicating with the NVA for network provisioning, etc.

Tenant System interface association should be performed before the VM enters running state, preferably in Shutdown state. If association with external NVE fails, the VM should not go into running state.

2.2 VM Live Migration Event

Live migration is sometimes referred to as "hot" migration, in that from an external viewpoint, the VM appears to continue to run while being migrated to another server (e.g., TCP connections generally survive this class of migration). In contrast, "cold" migration consists of shutdown VM execution on one server and restart it on another. For simplicity, the following abstract summary about live migration assumes shared storage, so that the VM's storage is accessible to the source and destination servers. Assume VM live migrates from hypervisor 1 to hypervisor 2. Such migration event involves the state transition on both hypervisors, source hypervisor 1 and destination hypervisor 2. VM state on source hypervisor 1 transits from Running to Migrating and then to Shutdown [I-D.ietf-opsawg-vmm-mib]. VM state on destination hypervisor 2 transits from Shutdown to Migrating and then Running.

The external NVE connected to destination hypervisor 2 has to associate the migrating VM's TSI with it by discovering the TSI's MAC and/or IP addresses, its VN, locally significant VID if any, and provisioning other network related parameters of the TSI. The external NVE may be informed about the VM's peer VMs, storage devices and other network appliances with which the VM needs to communicate or is communicating. The migrated VM on destination hypervisor 2 SHOULD not go to Running state before all the network provisioning and binding has been done.

The migrating VM SHOULD not be in Running state at the same time on

the source hypervisor and destination hypervisor during migration. The VM on the source hypervisor does not transition into Shutdown state until the VM successfully enters the Running state on the destination hypervisor. It is possible that VM on the source hypervisor stays in Migrating state for a while after VM on the destination hypervisor is in Running state.

2.3 VM Termination Event

VM termination event is also referred to as "powering off" a VM. VM termination event leads to its state going to Shutdown. There are two possible causes to terminate a VM [I-D.ietf-opsawg-vmm-mib], one is the normal "power off" of a running VM; the other is that VM has been migrated to another hypervisor and the VM image on the source hypervisor has to stop executing and to be shutdown.

In VM termination, the external NVE connecting to that VM needs to deprovision the VM, i.e. delete the network parameters associated with that VM. In other words, the external NVE has to de-associate the VM’s TSI.

2.4 VM Pause, Suspension and Resumption Events

The VM pause event leads to the VM transiting from Running state to Paused state. The Paused state indicates that the VM is resident in memory but no longer scheduled to execute by the hypervisor [I-D.ietf-opsawg-vmm-mib]. The VM can be easily re-activated from Paused state to Running state.

The VM suspension event leads to the VM transiting from Running state to Suspended state. The VM resumption event leads to the VM transiting state from Suspended state to Running state. Suspended state means the memory and CPU execution state of the virtual machine are saved to persistent store. During this state, the virtual machine is not scheduled to execute by the hypervisor [I-D.ietf-opsawg-vmm-mib].

In the Split-NVE architecture, the external NVE should keep any paused or suspended VM in association as the VM can return to Running state at any time.

3. Hypervisor-to-NVE Control Plane Protocol Functionality

The following subsections show the illustrative examples of the state transitions on external NVE which are relevant to Hypervisor-to-NVE Signaling protocol functionality. It should be noted they are not prescriptive text for full state machines.

3.1 VN connect and Disconnect

In Split-NVE scenario, a protocol is needed between the End Device(e.g. Hypervisor) making use of the external NVE and the external NVE in order to make the external NVE aware of the changing VN membership requirements of the Tenant Systems within the End Device.

A key driver for using a protocol rather than using static configuration of the external NVE is because the VN connectivity requirements can change frequently as VMs are brought up, moved and brought down on various hypervisors throughout the data center or external cloud.

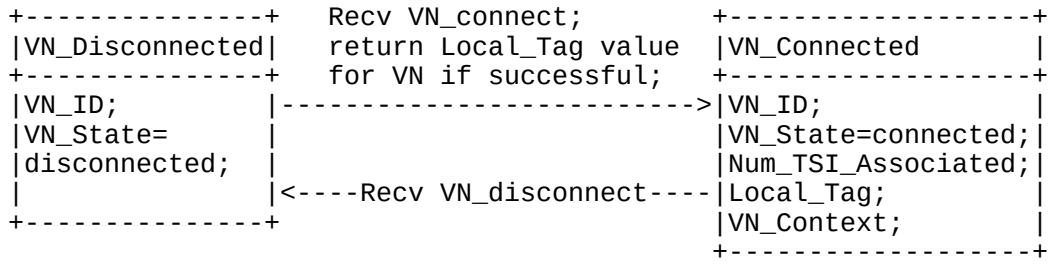


Figure 5 State Transition Example of a VAP Instance on an External NVE

Figure 5 shows the state transition for a VAP on the external NVE. An NVE that supports the hypervisor to NVE control plane protocol should support one instance of the state machine for each active VN. The state transition on the external NVE is normally triggered by the hypervisor-facing side events and behaviors. Some of the interleaved interaction between NVE and NVA will be illustrated for better understanding of the whole procedure; while others of them may not be shown. More detailed information regarding that is available in [I-D.ietf-nvo3-nve-nva-cp-req].

The external NVE must be notified when an End Device requires connection to a particular VN and when it no longer requires connection. In addition, the external NVE must provide a local tag value for each connected VN to the End Device to use for exchange of packets between the End Device and the external NVE (e.g. a locally significant 802.1Q tag value). How "local" the significance is depends on whether the Hypervisor has a direct physical connection to

the external NVE (in which case the significance is local to the physical link), or whether there is an Ethernet switch (e.g. a blade switch) connecting the Hypervisor to the NVE (in which case the significance is local to the intervening switch and all the links connected to it).

These VLAN tags are used to differentiate between different VNs as packets cross the shared access network to the external NVE. When the external NVE receives packets, it uses the VLAN tag to identify the VN of packets coming from a given TSI, strips the tag, and adds the appropriate overlay encapsulation for that VN and sends it towards the corresponding remote NVE across the underlying IP network.

The Identification of the VN in this protocol could either be through a VN Name or a VN ID. A globally unique VN Name facilitates portability of a Tenant's Virtual Data Center. Once an external NVE receives a VN connect indication, the NVE needs a way to get a VN Context allocated (or receive the already allocated VN Context) for a given VN Name or ID (as well as any other information needed to transmit encapsulated packets). How this is done is the subject of the NVE-to-NVA protocol which are part of work items 1 and 2 in [RFC7364].

VN_connect message can be explicit or implicit. Explicit means the hypervisor sending a message explicitly to request for the connection to a VN. Implicit means the external NVE receives other messages, e.g. very first TSI associate message (see the next subsection) for a given VN, to implicitly indicate its interest to connect to a VN.

A VN_disconnect message will indicate that the NVE can release all the resources for that disconnected VN and transit to VN_disconnected state. The local tag assigned for that VN can possibly be reclaimed by other VN.

3.2 TSI Associate and Activate

Typically, a TSI is assigned a single MAC address and all frames transmitted and received on that TSI use that single MAC address. As mentioned earlier, it is also possible for a Tenant System to exchange frames using multiple MAC addresses or packets with multiple IP addresses.

Particularly in the case of a TS that is forwarding frames or packets from other TSs, the external NVE will need to communicate the mapping between the NVE's IP address (on the underlying network) and ALL the addresses the TS is forwarding on behalf of for the corresponding VN to the NVA.

The NVE has two ways in which it can discover the tenant addresses for which frames must be forwarded to a given End Device (and ultimately to the TS within that End Device).

1. It can glean the addresses by inspecting the source addresses in packets it receives from the End Device.

2. The hypervisor can explicitly signal the address associations of a TSI to the external NVE. The address association includes all the MAC and/or IP addresses possibly used as source addresses in a packet sent from the hypervisor to external NVE. The external NVE may further use this information to filter the future traffic from the hypervisor.

To perform the second approach above, the "hypervisor-to-NVE" protocol requires a means to allow End Devices to communicate new tenant addresses associations for a given TSI within a given VN.

Figure 6 shows the example of a state transition for a TSI connecting to a VAP on the external NVE. An NVE that supports the hypervisor to NVE control plane protocol may support one instance of the state machine for each TSI connecting to a given VN.

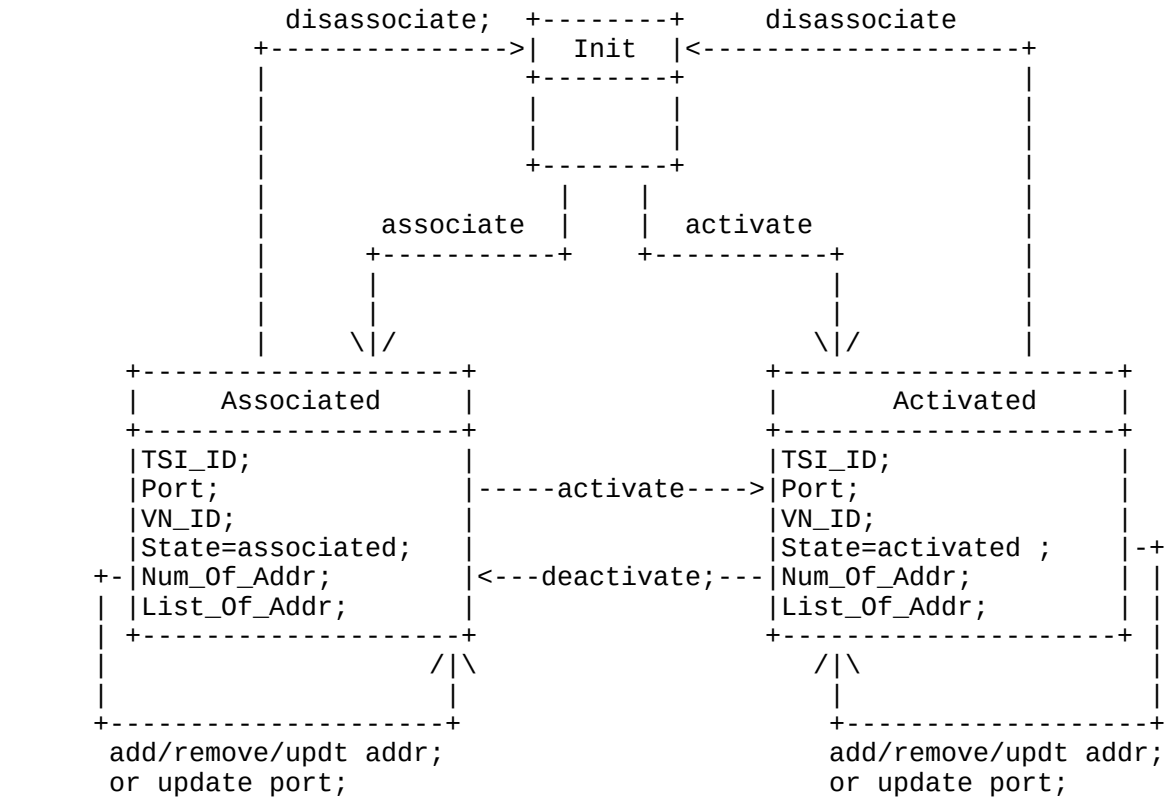


Figure 6 State Transition Example of a TSI Instance on an External NVE

Associated state of a TSI instance on an external NVE indicates all the addresses for that TSI have already associated with the VAP of the external NVE on port p for a given VN but no real traffic to and from the TSI is expected and allowed to pass through. An NVE has reserved all the necessary resources for that TSI. An external NVE may report the mappings of its' underlay IP address and the associated TSI addresses to NVA and relevant network nodes may save such information to its mapping table but not forwarding table. A NVE may create ACL or filter rules based on the associated TSI addresses on the attached port p but not enable them yet. Local tag for the VN corresponding to the TSI instance should be provisioned on port p to receive packets.

VM migration event(discussed section 2) may cause the hypervisor to send an associate message to the NVE connected to the destination hypervisor the VM migrates to. VM creation event may also lead to the

same practice.

The Activated state of a TSI instance on an external NVE indicates that all the addresses for that TSI functioning correctly on port p and traffic can be received from and sent to that TSI via the NVE. The mappings of the NVE's underlay IP address and the associated TSI addresses should be put into the forwarding table rather than the mapping table on relevant network nodes. ACL or filter rules based on the associated TSI addresses on the attached port p in NVE are enabled. Local tag for the VN corresponding to the TSI instance MUST be provisioned on port p to receive packets.

The Activate message makes the state transit from Init or Associated to Activated. VM creation, VM migration and VM resumption events discussed in section 4 may trigger the Activate message to be sent from the hypervisor to the external NVE.

TSI information may get updated either in Associated or Activated state. The following are considered updates to the TSI information: add or remove the associated addresses, update current associated addresses (for example updating IP for a given MAC), update NVE port information based on where the NVE receives messages. Such updates do not change the state of TSI. When any address associated to a given TSI changes, the NVE should inform the NVA to update the mapping information on NVE's underlying address and the associated TSI addresses. The NVE should also change its local ACL or filter settings accordingly for the relevant addresses. Port information update will cause the local tag for the VN corresponding to the TSI instance to be provisioned on new port p and removed from the old port.

3.3 TSI Disassociate and Deactivate

Disassociate and deactivate conceptually are the reverse behaviors of associate and activate. From Activated state to Associated state, the external NVE needs to make sure the resources are still reserved but the addresses associated to the TSI are not functioning and no traffic to and from the TSI is expected and allowed to pass through. For example, the NVE needs to inform the NVA to remove the relevant addresses mapping information from forwarding or routing table. ACL or filtering rules regarding the relevant addresses should be disabled. From Associated or Activated state to the Init state, the NVE will release all the resources relevant to TSI instances. The NVE should also inform the NVA to remove the relevant entries from mapping table. ACL or filtering rules regarding the relevant addresses should be removed. Local tag provisioning on the connecting port on NVE should be cleared.

A VM suspension event(discussed in section 2) may cause the relevant TSI instance(s) on the NVE to transit from Activated to Associated state. A VM pause event normally does not affect the state of the relevant TSI instance(s) on the NVE as the VM is expected to run again soon. The VM shutdown event will normally cause the relevant TSI instance(s) on NVE transit to Init state from Activated state. All resources should be released.

A VM migration will lead the TSI instance on the source NVE to leave Activated state. When a VM migrates to another hypervisor connecting to the same NVE, i.e. source and destination NVE are the same, NVE should use TSI_ID and incoming port to differentiate two TSI instance.

Although the triggering messages for state transition shown in Figure 6 does not indicate the difference between VM creation/shutdown event and VM migration arrival/departure event, the external NVE can make optimizations if it is notified of such information. For example, if the NVE knows the incoming activate message is caused by migration rather than VM creation, some mechanisms may be employed or triggered to make sure the dynamic configurations or provisionings on the destination NVE are the same as those on the source NVE for the migrated VM. For example IGMP query [RFC2236] can be triggered by the destination external NVE to the migrated VM on destination hypervisor so that the VM is forced to answer an IGMP report to the multicast router. Then multicast router can correctly send the multicast traffic to the new external NVE for those multicast groups the VM had joined before the migration.

4. Hypervisor-to-NVE Control Plane Protocol Requirements

Req-1: The protocol MUST support a bridged network connecting End Devices to External NVE.

Req-2: The protocol MUST support multiple End Devices sharing the same External NVE via the same physical port across a bridged network.

Req-3: The protocol MAY support an End Device using multiple external NVEs simultaneously, but only one external NVE for each VN.

Req-4: The protocol MAY support an End Device using multiple external NVEs simultaneously for the same VN.

Req-5: The protocol MUST allow the End Device initiating a request to its associated External NVE to be connected/disconnected to a given VN.

Req-6: The protocol MUST allow an External NVE initiating a request to its connected End Devices to be disconnected to a given VN.

Req-7: When a TS attaches to a VN, the protocol MUST allow for an End Device and its external NVE to negotiate a locally-significant tag for carrying traffic associated with a specific VN (e.g., 802.1Q tags).

Req-8: The protocol MUST allow an End Device initiating a request to associate/disassociate and/or activate/deactive address(es) of a TSI instance to a VN on an NVE port.

Req-9: The protocol MUST allow the External NVE initiating a request to disassociate and/or deactivate address(es) of a TSI instance to a VN on an NVE port.

Req-10: The protocol MUST allow an End Device initiating a request to add, remove or update address(es) associated with a TSI instance on the external NVE. Addresses can be expressed in different formats, for example, MAC, IP or pair of IP and MAC.

Req-11: The protocol MUST allow the External NVE to authenticate the End Device connected.

Req-12: The protocol MUST be able to run over L2 links between the End Device and its External NVE.

Req-13: The protocol SHOULD support the End Device indicating if an associate or activate request from it results from a VM hot migration event.

5. VDP Applicability and Enhancement Needs

Virtual Station Interface (VSI) Discovery and Configuration Protocol (VDP) [IEEE 802.1Qbg] can be the control plane protocol running between the hypervisor and the external NVE. Appendix A illustrates VDP for reader’s information.

VDP facilitates the automatic discovery and configuration for Edge Virtual Bridging (EVB) station and Edge Virtual Bridging (EVB) bridge. EVB station is normally an end station running multiple VMs. It is conceptually equivalent to hypervisor in this document. And EVB bridge is conceptually equivalent to the external NVE.

VDP is able to pre-associate/associate/de-associate a VSI on EVB station to a port on the EVB bridge. VSI is approximately the concept of a virtual port a VM connects to the hypervisor in this document

context. The EVB station and the EVB bridge can reach the agreement on VLAN ID(s) assigned to a VSI via VDP message exchange. Other configuration parameters can be exchanged via VDP as well. VDP is carried over Edge Control Protocol(ECP) [IEEE8021Qbg] which provides a reliable transportation over a layer 2 network.

VDP protocol needs some extensions to fulfill the requirements listed in this document. Table 1 shows the needed extensions and/or clarifications in NVO3 context.

<table>
<tr><th>Req</th><th>VDP supported?</th><th>remarks</th></tr>
<tr><td>Req-1</td><td rowspan="2">Partially</td><td>Needs extension. Dest MAC can be a specific unicast MAC besides Nearest Customer Bridge group MAC</td></tr>
<tr><td>Req-2</td></tr>
<tr><td>Req-3</td><td rowspan="2">Partially</td><td>Needs clarification and extension for link aggregation support.
For req-4, (pre-)associate status needs to be synchronized on all NVE ports.</td></tr>
<tr><td>Req-4</td></tr>
<tr><td>Req-5</td><td>Yes</td><td>VN is indicated by GroupID</td></tr>
<tr><td>Req-6</td><td>Yes</td><td>Bridge sends De-Associate</td></tr>
<tr><td>Req-7</td><td>Yes</td><td>VID==NULL in request and bridge returns the assigned value in response</td></tr>
<tr><td rowspan="3">Req-8</td><td rowspan="3">Partially</td><td>requirements | VDP equivalence
associate/disassociate | pre-asso/de-associate
activate/deactivate | associate/de-associate</td></tr>
<tr><td>Needs extension to allow associate->pre-assoc</td></tr>
<tr><td></td></tr>
<tr><td>Req-9</td><td>Yes</td><td>VDP bridge initiates de-associate</td></tr>
<tr><td>Req-10</td><td>Partially</td><td>Needs extension for IPv4/IPv6 address</td></tr>
<tr><td>Req-11</td><td>No</td><td>Needs extension for authentication</td></tr>
<tr><td>Req-12</td><td>Yes</td><td>L2 protocol naturally</td></tr>
</table>

		M bit for migrated VM on destination hypervisor and S bit for that on source hypervisor.
Req-13	Partially	It is indistinguishable when M/S is 0 between no guidance and events not caused by migration where NVE may act differently. Needs extension to clearly define them.

Table 1 Compare VDP with the requirements

Simply adding the ability to carry layer 3 addresses, VDP can serve the Hypervisor-to-NVE control plane functions pretty well. Other extensions are the improvement of the protocol capabilities for better fit in NVO3 network.

6. Security Considerations

NVEs must ensure that only properly authorized Tenant Systems are allowed to join and become a part of any specific Virtual Network. In addition, NVEs will need appropriate mechanisms to ensure that any hypervisor wishing to use the services of an NVE are properly authorized to do so. One design point is whether the hypervisor should supply the NVE with necessary information (e.g., VM addresses, VN information, or other parameters) that the NVE uses directly, or whether the hypervisor should only supply a VN ID and an identifier for the associated VM (e.g., its MAC address), with the NVE using that information to obtain the information needed to validate the hypervisor-provided parameters or obtain related parameters in a secure manner.

7. IANA Considerations

No IANA action is required. RFC Editor: please delete this section before publication.

8. Acknowledgements

This document was initiated and merged from the drafts draft-kreeger-nvo3-hypervisor-nve-cp, draft-gu-nvo3-tes-nve-mechanism and draft-kompella-nvo3-server2nve. Thanks to all the co-authors and contributing members of those drafts.

The authors would like to specially thank Jon Hudson for his generous help in improving the readability of this document.

8. References

8.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2 Informative References

[RFC7364] Narten, T., Gray, E., Black, D., Fang, L., Kreeger, L., and M. Napierala, "Problem Statement: Overlays for Network Virtualization", October 2014.

[RFC7365] Lasserre, M., Balus, F., Morin, T., Bitar, N., and Y. Rekhter, "Framework for DC Network Virtualization", October 2014.

[I-D.ietf-nvo3-nve-nva-cp-req] Kreeger, L., Dutt, D., Narten, T., and D. Black, "Network Virtualization NVE to NVA Control Protocol Requirements", draft-ietf-nvo3-nve-nva-cp-req-01 (work in progress), October 2013.

[I-D.ietf-nvo3-arch] Black, D., Narten, T., et al, "An Architecture for Overlay Networks (NVO3)", draft-narten-nvo3-arch, work in progress.

[I-D.ietf-opsawg-vmm-mib] Asai H., MacFaden M., Schoenwaelder J., Shima K., Tsou T., "Management Information Base for Virtual Machines Controlled by a Hypervisor", draft-ietf-opsawg-vmm-mib-00 (work in progress), February 2014.

[IEEE 802.1Qbg] IEEE, "Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks - Amendment 21: Edge Virtual Bridging", IEEE Std 802.1Qbg, 2012

[8021Q] IEEE, "Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks", IEEE Std 802.1Q-2011, August, 2011

Appendix A. IEEE 802.1Qbg VDP Illustration (For information only)

VDP has the format shown in Figure A.1. Virtual Station Interface (VSI) is an interface to a virtual station that is attached to a downlink port of an internal bridging function in server. VSI’s VDP packet will be handled by an external bridge. VDP is the controlling protocol running between the hypervisor and the external bridge.

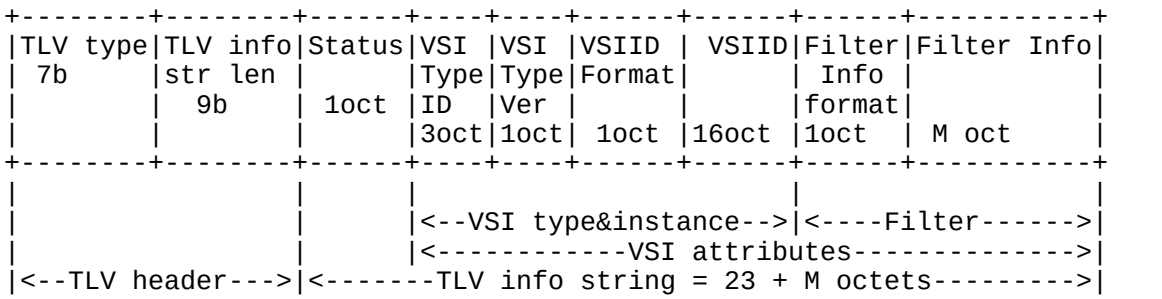


Figure A.1: VDP TLV definitions

There are basically four TLV types.

1. Pre-Associate: Pre-Associate is used to pre-associate a VSI instance with a bridge port. The bridge validates the request and returns a failure Status in case of errors. Successful pre-association does not imply that the indicated VSI Type or provisioning will be applied to any traffic flowing through the VSI. The pre-associate enables faster response to an associate, by allowing the bridge to obtain the VSI Type prior to an association.

2. Pre-Associate with resource reservation: Pre-Associate with Resource Reservation involves the same steps as Pre-Associate, but on successful pre-association also reserves resources in the Bridge to prepare for a subsequent Associate request.

3. Associate: The Associate creates and activates an association between a VSI instance and a bridge port. The Bridge allocates any required bridge resources for the referenced VSI. The Bridge activates the configuration for the VSI Type ID. This association is then applied to the traffic flow to/from the VSI instance.

4. Deassociate: The de-associate is used to remove an association between a VSI instance and a bridge port. Pre-Associated and Associated VSIs can be de-associated. De-associate releases any resources that were reserved as a result of prior Associate or Pre-Associate operations for that VSI instance.

Deassociate can be initiated by either side and the rest types of messages can only be initiated by the server side.

Some important flag values in VDP Status field:

1. M-bit (Bit 5): Indicates that the user of the VSI (e.g., the VM) is migrating (M-bit = 1) or provides no guidance on the migration of the user of the VSI (M-bit = 0). The M-bit is used as an indicator relative

to the VSI that the user is migrating to.

2. S-bit (Bit 6): Indicates that the VSI user (e.g., the VM) is suspended (S-bit = 1) or provides no guidance as to whether the user of the VSI is suspended (S-bit = 0). A keep-alive Associate request with S-bit = 1 can be sent when the VSI user is suspended. The S-bit is used as an indicator relative to the VSI that the user is migrating from.

The filter information format currently supports 4 types as the following.

1. VID Filter Info format

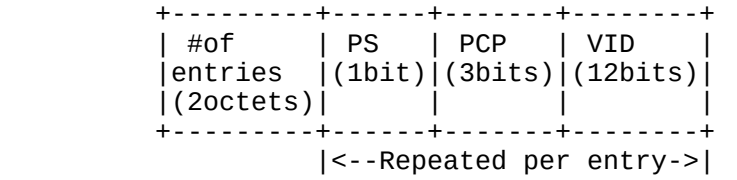


Figure A.2 VID Filter Info format

2. MAC/VID filter format

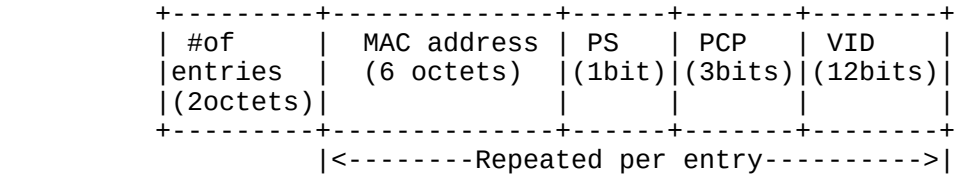


Figure A.3 MAC/VID filter format

3. GroupID/VID filter format

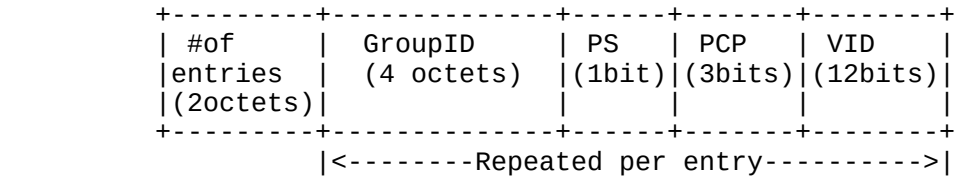


Figure A.4 GroupID/VID filter format

4. GroupID/MAC/VID filter format

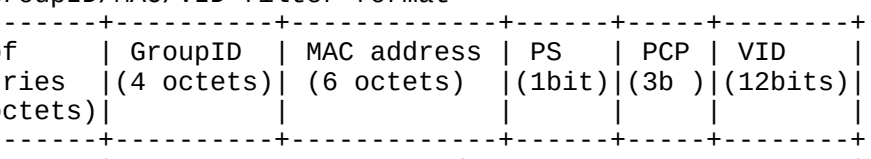


Figure A.5 GroupID/MAC/VID filter format

The null VID can be used in the VDP Request sent from the hypervisor to the external bridge. Use of the null VID indicates that the set of VID values associated with the VSI is expected to be supplied by the Bridge. The Bridge can obtain VID values from the VSI Type whose identity is specified by the VSI Type information in the VDP Request. The set of VID values is returned to the station via the VDP Response. The returned VID value can be a locally significant value. When GroupID is used, it is equivalent to the VN ID in NVO3. GroupID will be provided by the hypervisor to the bridge. The bridge will map GroupID to a locally significant VLAN ID.

The VSIID in VDP request that identify a VM can be one of the following format: IPV4 address, IPV6 address, MAC address, UUID or locally defined.

Authors' Addresses

Yizhou Li
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

Phone: +86-25-56625409
EMail: liyizhou@huawei.com

Lucy Yong
Huawei Technologies, USA

Email: lucy.yong@huawei.com

Lawrence Kreeger
Cisco

Email: kreeger@cisco.com

Thomas Narten
IBM

Email: narten@us.ibm.com
David Black
EMC

Email: david.black@emc.com

NVO3
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2015

P. Jain
K. Singh
D. Garcia del Rio
Nuage Networks
W. Henderickx
Alcatel-Lucent
V. Bannai
PayPal
R. Shekhar
A. Lohiya
Juniper Networks
March 06, 2015

Generic Overlay OAM and Datapath Failure Detection
draft-jain-nvo3-overlay-oam-03

Abstract

This proposal describes a mechanism that can be used to detect Data Path Failures of various overlay technologies as VXLAN, NVGRE, MPLSoGRE and MPLSoUDP and verifying/sanity of their Control and Data Plane for given Overlay Segment. This document defines the following for each of the above Overlay Technologies:

- Encapsulation of OAM Packet, such that it has same Outer and Overlay Header as any End-System’s data going over the same Overlay Segment.

- The mechanism to trace the Underlay that is exercised by any Overlay Segment.

- Procedure to verify presence of any given Tenant VM or End-System within a given Overlay Segment at Overlay End-Point.

Even though the present proposal addresses Overlay OAM for VXLAN, NVGRE, MPLSoGRE and MPLSoUDP, but the procedures described are generic enough to accommodate OAM for any other Overlay Technology.

Status of This Memo

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

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

Copyright Notice

Table of Contents

1. Introduction

VXLAN [RFC7348], NVGRE [I-D.draft-sridharan-virtualization-nvgre], MPLSoGRE [RFC4023] and MPLSoUDP [I-D.draft-ietf-mpls-in-udp] are well known technologies and are used as tunneling mechanism to Overlay either Layer 2 networks or Layer 3 networks on top of Layer 3 Underlay networks. For all above Overlay Models there are two Tunnel End Points for a given Overlay Segment. One End Point is where the Overlay Originates, and other where Overlay Terminates. In most cases the Tunnel End Point is intended to be at the edge of the network, typically connecting an access switch to an IP transport network. The access switch could be a physical or a virtual switch

located within the hypervisor on the server which is connected to End System which is a VM.

This document describes a mechanism that can be used to detect Data Plane failures and sanity of Overlay Control and Data Plane for a given Overlay Segment, and the method to trace the Underlay path that is exercised by any given Overlay Segment.

The document also defines procedures for validating the presence of any given Tenant VM/End-System/End-System or Flow representing the End-System System within a given Overlay Segment.

The proposal describes:

- The mechanism to verify Overlay Control Plane and Data Plane consistency at the Overlay End Point(s), by encapsulating the OAM Packet in exact the same way as that of any End System Traffic that is transported over the Overlay Segment.

- The mechanism to trace the Underlay that is exercised by any Overlay Segment.

- The mechanism to verify presence of any "End-System" in a given Overlay Segment.

The proposal defines the information to check correct operation of the Data Plane, as well as a mechanism to verify the Data Plane against the Control Plane for a given Overlay Segment.

It is important consideration in this proposal to carry Echo Request along same Data Path that any End System’s data using the given Overlay Segment takes.

The tenants VM(s) or End System(s) are not aware of the Overlays and as such the need for the verification of the Data Path MUST solely rest with the Cloud Provider. The use cases where the Tenant VM(s) need to be aware of the Data Plane failures is beyond the scope of this document.

2. Terminology

Terminology used in this document:

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

When used in lower case, these words convey their typical use in common language, and are not to be interpreted as described in RFC2119 [RFC2119].

OAM: Operations, Administration, and Management

VXLAN: Virtual eXtensible Local Area Network.

NVGRE: Network Virtualization using GRE.

MPLSoGRE: Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)

MPLSoUDP: Encapsulating MPLS in UDP.

Originating End Point: Overlay Segment’s Head End or Starting Point of Overlay Tunnel.

Terminating End Point: Overlay Segment’s Tail End or Terminating Point of Overlay Tunnel.

VM: Virtual Machine.

VNI: VXLAN Network Identifier (or VXLAN Segment ID)

VSID: Virtual Subnet ID. (for NVGRE)

NVE: Network Virtualized Edge

End System: Could be Tenant VM, Host, Bridge etc. - System whose data is expected to go over Overlay Segment.

Echo Request: Throughout this document, Echo Request packet is expected to be transmitted by Originator Overlay End Point and destined to Overlay Terminating End Point.

Echo Reply: Throughout this document, Echo Reply packet is expected to be transmitted by Terminating Overlay End Point and destined to Overlay Originating End Point.

Other terminologies are as defined in [RFC7348], [I-D.draft-sridharan-virtualization-nvgre], [RFC4023] and [I-D.draft-ietf-mpls-in-udp]

3. Motivation for Overlay OAM

When any Overlay Segment fails to deliver user traffic, there is a need to provide a tool that would enable users, as Cloud Providers to detect such failures, and a mechanism to isolate faults. It may also be desirable to test the data path before mapping End System traffic to the Overlay Segment.

The basic idea is to facilitate following verifications:-

- End-System's data that are expected to go over a particular Overlay Segment actually ends up using the Data-Path represented by given Overlay Segment between the two End-Points.
- To verify the correct value of Overlay Segment Identifier is programmed at Originating and Terminating End Point(s) for a given Overlay Segment. Segment Identifier will be VNI for VXLAN, VSID for NVGRE, MPLS Label for MPLSoGRE and MPLSoUDP.
- The facilitate mechanism to trace the Underlay that is exercised by any Overlay Segment.
- The mechanism to verify presence of any "End-System" in a given Overlay Segment.

To facilitate verification of Overlay Segment or any End-System using the Overlay, this document proposes sending of a Packet (called an "Echo Request") along the same data path as other Packets belonging to this Segment. Echo Request also carries information about the Overlay Segment whose Data Path is to be verified. This Echo Request is forwarded just like any other End System Data Packet belonging to that Overlay Segment, as it contains the same Overlay Encapsulation as regular End System's data.

On receiving Echo Request at the end of the Overlay Segment, it is sent to the Control Plane of the Terminating Overlay End Point, which in-turn would respond with Echo Reply.

To facilitate tracing of the Underlay used by any given Overlay Segment, the document proposes Echo Request/Reply encapsulation in "trace mode", which would allow the user or Cloud Provider to gather information of the Underlay network.

4. Approach

The proposal aims at validating Data Plane and its view of Control Plane for a particular Overlay Segment. To achieve this aim, the draft proposes creating an Overlay OAM Packet which MUST be

encapsulated with the Overlay Header as that of any End-Point data going over the same Overlay Segment. This would guarantee the data-path for OAM Packet follows the same path as that for any End User data going over the same Overlay Segment.

The draft outlines procedures to encode Overlay Header and Inner Ethernet or IP Header based on the type of payload that Overlay is expected to carry.

5. Packet Format

Generic Overlay Echo Request/Reply is a UDP Packet identified by well known UDP Port XXXX. The payload carried by Overlay typically could be either be Layer 2 / Ethernet Frame, or it could be Layer 3 / IP Packet.

5.1. Overlay OAM Encapsulation in Layer 2 Context

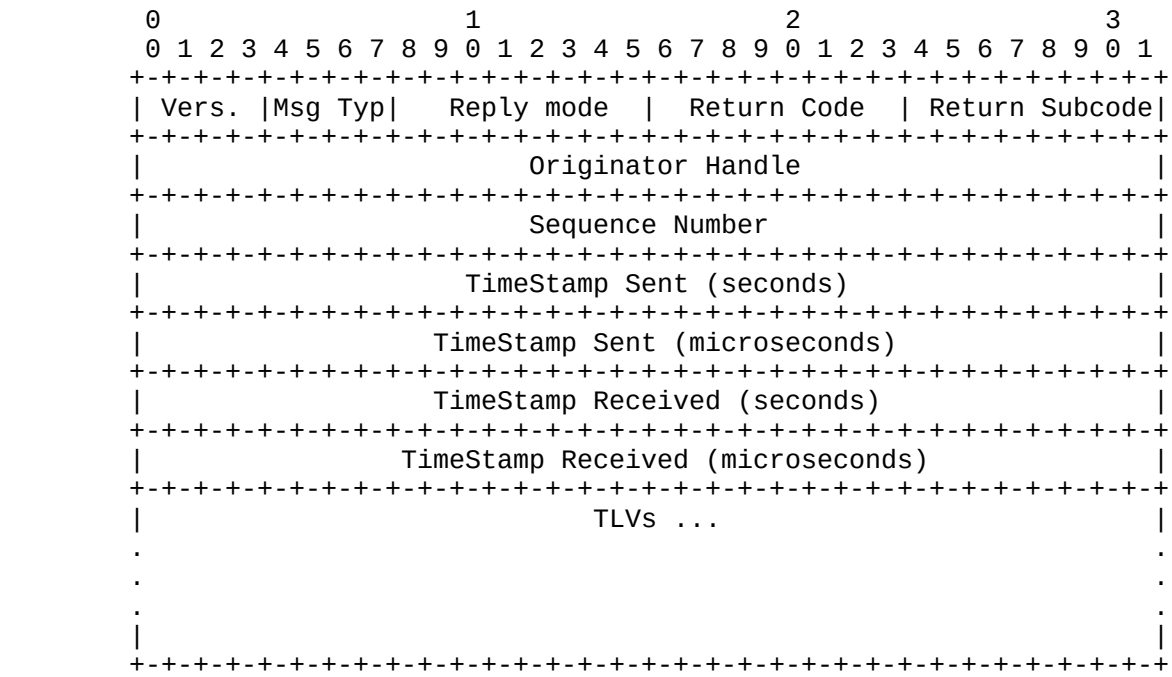
If the encapsulated payload carried by Overlay is of type Ethernet, then the OAM Echo Request packet would have inner Ethernet Header, followed by IP and UDP Header. The payload of inner UDP would be as described in below section "Generic Overlay OAM Packet Format".

5.2. Overlay OAM Encapsulation in Layer 3 Context

If the encapsulated payload carried by Overlay is of type IP, then the OAM Echo Request packet would have inner IP Header, followed by UDP Header. The payload of inner UDP would be as described in below section "Generic Overlay OAM Packet Format".

5.3. Generic Overlay OAM Packet Format

Following is the format of UDP payload of Generic Overlay OAM Packet:



Generic Overlay OAM Packet

The Vers. field represents the PDU encoding version

Value	What it means
0	Initial Version
15	Reserved value

The Message Type is one of the following:-

Value	What it means
1	Echo Request
2	Echo Reply

Reply Mode Values:-

Value	What it means
1	Do not reply
2	Reply via an IPv4/IPv6 UDP Packet
3	Reply via Overlay Segment

Echo Request with 1 (Do not reply) in the Reply Mode field may be used for one-way connectivity tests. The receiving node may log gaps in the Sequence Numbers and/or maintain delay/jitter statistics. For normal operation Echo Request would have 2 (Reply via an IPv4 UDP Packet) in the Reply Mode field.

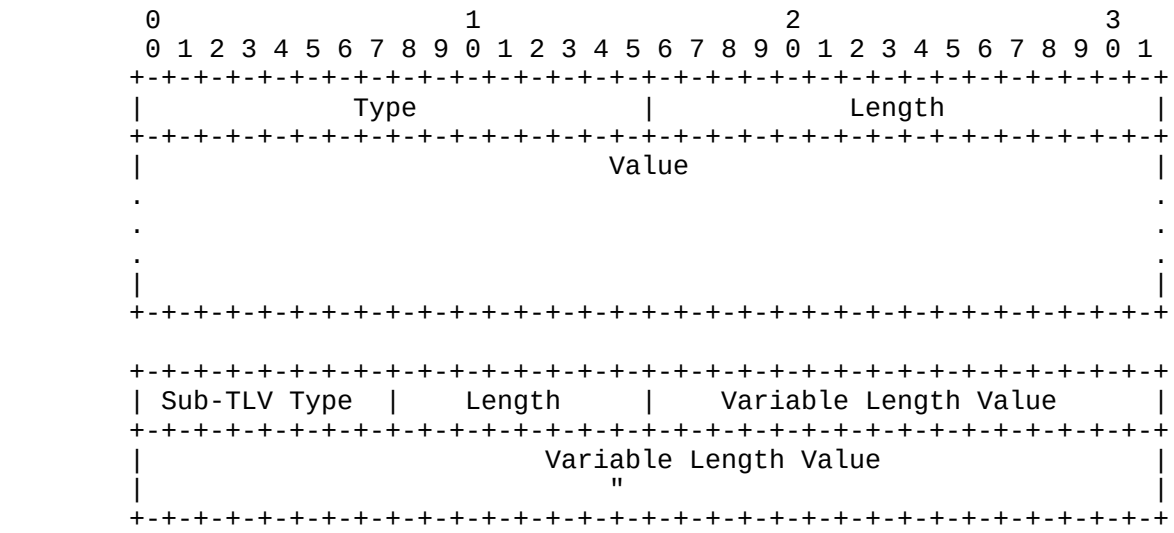
If it is desired that the reply also comes back via Overlay Segment i.e. encapsulated with the Overlay Header, then the Reply Mode filed needs to be set to 3 (Reply via Overlay Segment).

The Originator’s Handle is filled in by the Originator, and returned unchanged by the receiver in the Echo Reply (if any). The value used for this field can be implementation dependent, this MAY be used by the Originator for matching up requests with replies.

The Sequence Number is assigned by the Originator of Echo Request and can be (for example) used to detect missed replies.

The TimeStamp Sent is the time-of-day (in seconds and microseconds, according to the sender’s clock) in NTP format [NTP] when the VXLAN Echo Request is sent. The TimeStamp Received in an Echo Reply is the time-of-day (according to the receiver’s clock) in NTP format that the corresponding Echo Request was received.

TLVs (Type-Length-Value tuples) have the following format:



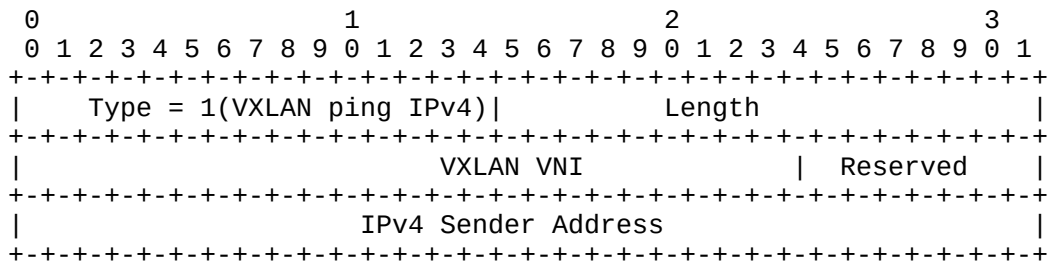
Types are defined below; Length is the length of the Value field in octets. The Value field depends on the Type; it is zero padded to align to a 4-octet boundary. There could be one or many optional Sub-TLV that could be encoded under the TLV.

5.3.1. TLV Types for various Overlay Ping Models

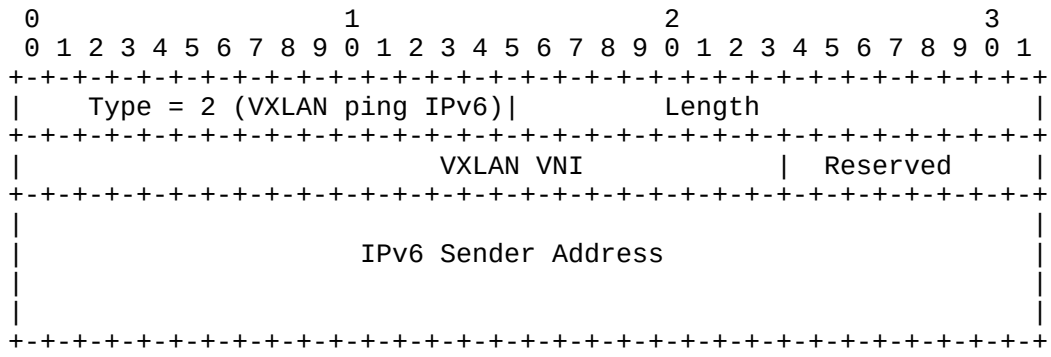
TLV Types:-

Value	What it means
1	VXLAN Segment Ping for IPv4
2	VXLAN Segment Ping for IPv6
3	NVGRE Segment Ping for IPv4
4	NVGRE Segment Ping for IPv6
5	MPLSoGRE Segment Ping for IPv4
6	MPLSoGRE Segment Ping for IPv6
7	MPLSoUDP Segment Ping for IPv4
8	MPLSoUDP Segment Ping for IPv6

5.3.1.1. TLV for VXLAN Ping

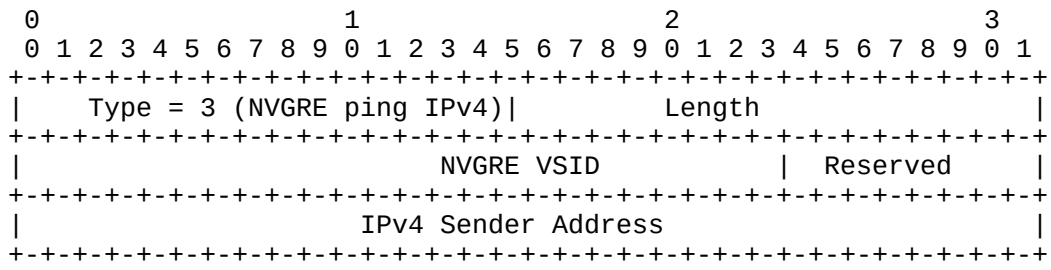


TLV if Sender Address is IPv4

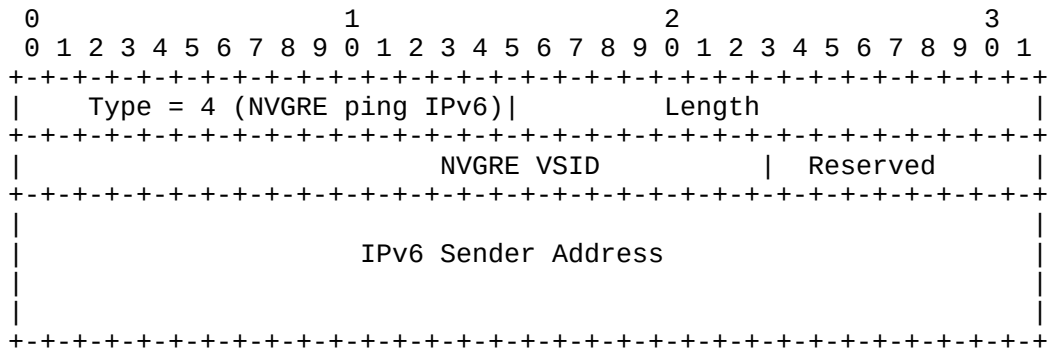


TLV if Sender Address is IPv6

5.3.1.2. TLV for NVGRE Ping

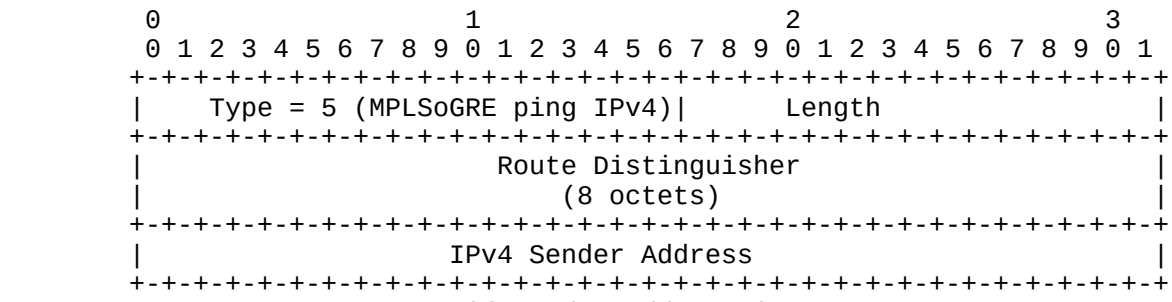


TLV if Sender Address is IPv4

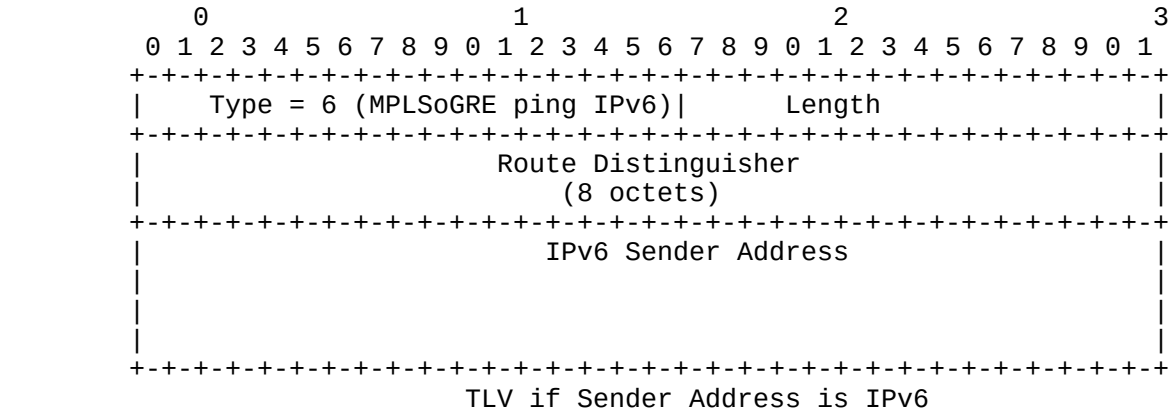


TLV if Sender Address is IPv6

5.3.1.3. TLV for MPLSoGRE Ping



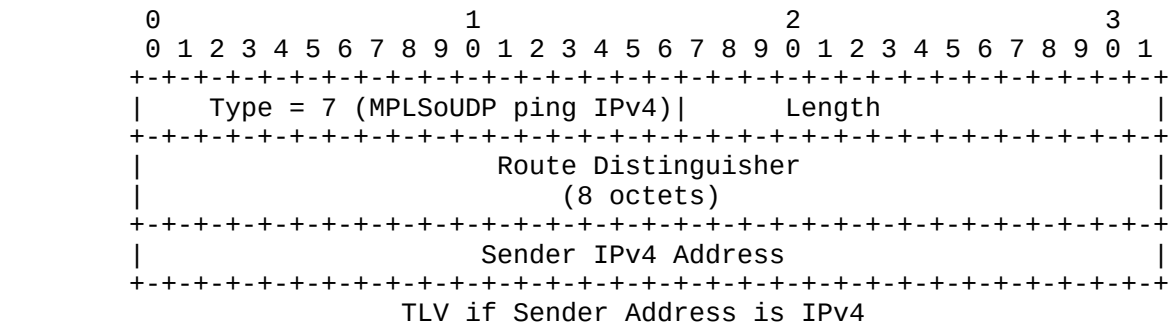
TLV if Sender Address is IPv4



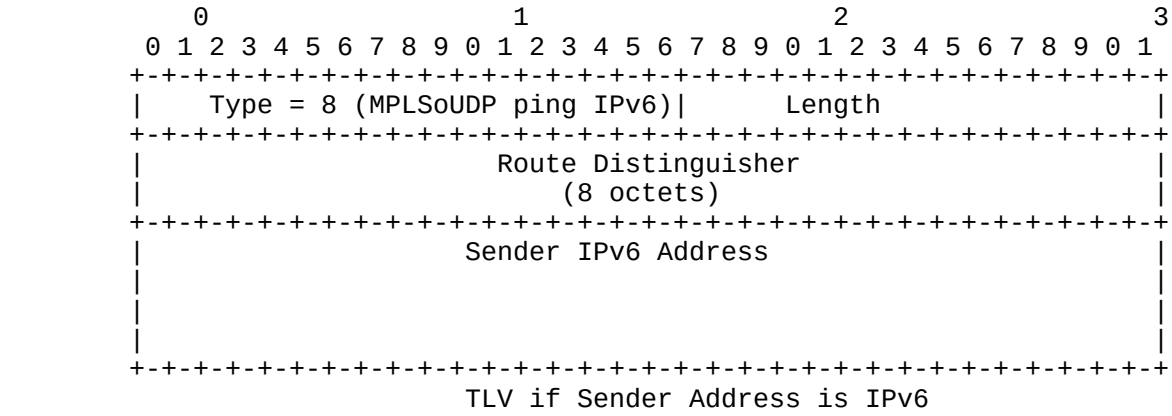
TLV if Sender Address is IPv6

Route Distinguisher is defined as part of [RFC4365]

5.3.1.4. TLV for MPLSoUDP Ping



TLV if Sender Address is IPv4



TLV if Sender Address is IPv6

Route Distinguisher is defined as part of [RFC4365]

6. Return Codes

Sender MUST always set the Return Code set to zero. The receiver can set it to one of the values listed below when replying back to Echo-Request.

Following are the Return Codes (Suggested):-

Value	What it means
0	No return code
1	Malformed Echo Request Received
2	Overlay Segment Not Present
3	Overlay Segment Not Operational
4	Return-Code-OK

7. Procedure for Overlay Segment Ping

Echo Request is used to test Data Plane and its view of Control Plane for particular Overlay Segment. The Overlay Segment to be verified is identified differently for various Overlay Technologies. For VXLAN, VNI is used to identify given Overlay Segment. For NVGRE, VSID is used. For MPLSoGRE and MPLSoUDP the MPLS Stack is used to identify a given Overlay Segment.

For the Data Plane verification, the Overlay Echo Request Packet MUST be encapsulated within the Overlay Header, which is same as that of any End-Point data going over the same Overlay Segment. This would guarantee the data-path for OAM Packet follows the same path as that for any End User data going over the same Overlay Segment.

The payload carried by Overlay typically could be either be Layer 2 or Ethernet Frame, or it could be Layer 3 or IP Packet. Based on the type of payload following is the way inner Header(s) of Echo Request would be encoded.

7.1. Encoding of Inner Header for Echo Request in Layer 2 Context

If the encapsulated payload carried by Overlay is of type Ethernet, then the OAM Echo Request packet would have inner Ethernet Header, followed by IP and UDP Header. The payload of inner UDP would be as described in below section "Generic Overlay OAM Packet Format".

Inner Ethernet Header for the Echo Request Packet MUST have the Destination Mac set to 00-00-5E-90-XX-XX (to be assigned IANA). The Source Mac should be set to Mac Address of the Originating VTEP. However, it is desired that the Inner Source Mac SHOULD not be learnt in the MAC-Table as this represent Control Packet in context of Overlay OAM.

Inner IP header is set with the Source IP Address which is a routable Address of the sender; the Destination IP Address is a (randomly chosen) IPv4 Address from the range 127/8, IPv6 addresses are chosen from the range 0:0:0:0:0:FFFF:127/104. The IP TTL is set to 255.

The inner Destination UDP port is set to xxxx (assigned by IANA for Overlay OAM).

The "Generic Overlay OAM Packet" will now be encoded, with following information.

The sender chooses a Originator’s Handle and a Sequence Number. When sending subsequent Overlay Echo Requests, the sender SHOULD increment the Sequence Number by 1.

The TimeStamp Sent is set to the time-of-day (in seconds and microseconds) that the Echo Request is sent. The TimeStamp Received is set to zero. Also, the Reply Mode must be set to the desired reply mode. The Return Code and Subcode are set to zero.

Next, the TLV is Encoded for desired Overlay Type, as per Section "Types of TLVs defined for various Overlay Ping Models"

7.2. Encoding of Inner Header for Echo Request in Layer 3 Context

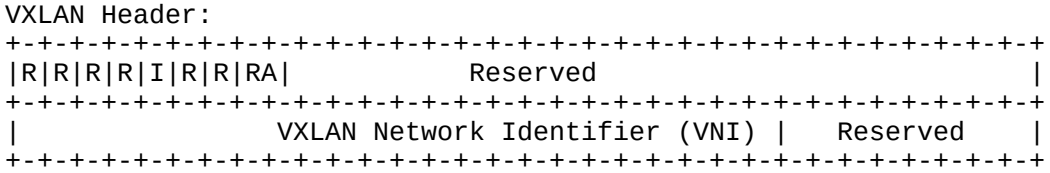
If the encapsulated payload carried by Overlay is of type IP, then the Encoding of the Echo Request would be same as above Section "Encoding of Inner Header for Echo Request in Layer 2 Context", but without the presence of Inner Ethernet Header.

7.3. VXLAN Procedures

7.3.1. Sending VXLAN Echo Request

The Outer VxLAN header for the Echo Request packet follows the encapsulation as defined in [RFC7348]. The VNI is same as that of the VXLAN Segment that is being verified. This would make sure that OAM Packet takes the same datapath as any other End System data going over this VXLAN Segment.

The VXLAN Router Alert option [I-D.draft-singh-nvo3-vxlan-router-alert] MUST be set in the VXLAN header as shown below.



Originating VTEP MAY set the I Bit to 0 in VXLAN Header when sending OAM Frame. This would cause dropping of such VXLAN frames on any Terminating VTEP that does not understand Overlay OAM framework, and prevent sending those frames to End-Systems or VMs.

It is desired to choose the Source UDP port (in the outer header), so as to exercise the same Data-Path as that of the traffic carried over the VXLAN Segment and is left to the implementation.

The Encoding of Inner Header(s) and UDP payload of Generic Overlay OAM Packet is as described in above Sub-Section i.e. "Encoding of Inner Header for Echo Request in Layer 2/Layer 3 Context".

7.3.2. Receiving VXLAN Echo Request

At the Terminating Overlay End Point or VTEP, since the Overlay OAM Packet is exactly same as that of End-System Packet(s). It is important to send OAM packet to Control Plane and prevent it from sending to the End System. The trapping and sending VXLAN Echo Request to the Control Plane is triggered by one of the following Packet processing exceptions: VXLAN Router Alert option, [I-D.draft-singh-nvo3-vxlan-router-alert] the Inner Destination MAC Address of 00-00-5E-90-XX-XX as defined in above section, and the Destination IP Address in the 127/8 Address range for IPv4 Address, or 0:0:0:0:0:FFFF:127/104 for IPv6 Address.

The Control Plane further identifies the Overlay OAM Application by UDP well know destination port xxxx.

Since the VxLAN Router Alert bit is set in VxLAN Header, which signifies the presence of Control Packet. The terminating VTEP SHOULD not learn the Mac address set in the Inner Mac Header of VxLAN Echo Request Packet.

Once the VXLAN Echo Request Packet is identified at Control Plane, it is processed as follows:-

- o General Packet sanity is verified. If the Packet is not well-formed, VTEP SHOULD send VXLAN Echo Reply with the Return Code set to "Malformed Echo Request received" and the Subcode to zero. The header fields Originator's Handle, Sequence Number, and Timestamp Sent are not examined, but are included in the VXLAN Echo Reply message

- o VNI Validation: If there is no entry for VNI, it indicates that there could be a transient or permanent disconnect between Control Plane and data Plane and VTEP needs to report an error with Return Code of "Overlay Segment Not Present" and a Return Subcode of Zero. If the mapping for VNI Exists, but the state is not Operational, VTEP needs to report an error with Return Code of "Overlay Segment Not Operational" If the mapping exists then send VXLAN Echo Reply with a Return Code of "Return-Code-OK", and a Return Subcode of Zero. The procedures for sending the Echo Reply are found in subsection below section.

7.3.3. Sending VXLAN Echo Reply

If the Reply Mode is set to "Reply via an IPv4/IPv6 UDP Packet", the Echo Reply is a UDP Packet. It MUST ONLY be sent in response to Echo Request. The Source IP Address in the Header should be Routable Address of the replier; The Destination IP Address should be IP Address of the Echo Request's Originating End Point or the requester. The destination UDP Port is set to XXXX (assigned by IANA for identifying VXLAN OAM application). The IP TTL is set to 255.

The format of the Echo Reply is the same as the Echo Request. The Originator Handle, the Sequence Number, and TimeStamp Sent are copied from the Echo Request; the TimeStamp Received is set to the time-of-day that the Echo Request is received (note that this information is most useful if the time-of-day clocks on the requester and the replier are synchronized). The replier MUST fill in the Return Code and Subcode, as determined in the previous subsection.

If the Reply Mode is set to "Reply via Overlay Segment", then the Replying Overlay End Point is expected to place Echo Reply packet in-band in the Overlay Segment destined to the Originating Overlay End Point. The detailed encapsulation for this would be covered in next revision of the draft.

7.3.4. Receiving VXLAN Echo Reply

An Originating Overlay End Point should only receive Echo Reply in response to an Echo Request that it sent. When the Reply Mode is "Reply via an IPv4/IPv6 UDP Packet", the Echo Reply would be and IP Packet/UDP Packet, and is identified by the destination UDP Port

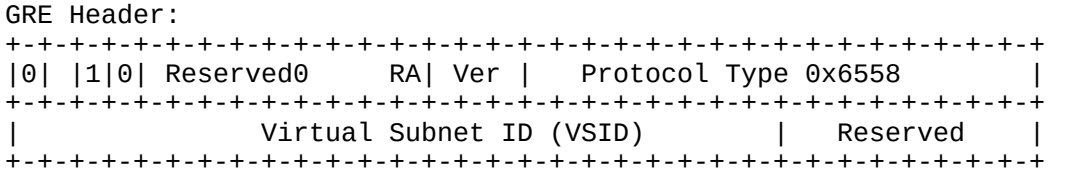
XXXX. The Originating Overlay End Point should parse the Packet to ensure that it is well-formed, then attempt to match up the Echo Reply with an Echo Request that it had previously sent, and the Originator Handle. If no match is found, then it should drop the Echo Reply Packet; otherwise, it checks the Sequence Number to see if it matches.

7.4. NVGRE Procedures

7.4.1. Sending NVGRE Echo Request

The Outer NVGRE header for the Echo Request packet follows the encapsulation as defined in [I-D.draft-sridharan-virtualization-nvgre]. The VSID is same as that of the NVGRE Segment that is being verified. This would make sure that OAM Packet takes the same datapath as any other End System data going over this NVGRE Segment.

The NVGRE Router Alert option [I-D.draft-singh-nvo3-nvgre-router-alert] MUST be set in the NVGRE header as shown below.



The Encoding of Inner Header(s) and UDP payload of Generic Overlay OAM Packet is as described in above Sub-Section i.e. "Encoding of Inner Header for Echo Request in Layer 2/Layer 3 Context".

7.4.2. Receiving NVGRE Echo Request

At the Terminating Overlay End Point, since the Overlay OAM Packet is exactly same as that of End-System Packet(s). It is important to send OAM packet to Control Plane and prevent it from sending to the End System. The trapping and sending NVGRE Echo Request to the Control Plane is triggered by one of the following Packet processing exceptions: NVGRE Router Alert option, [I-D.draft-singh-nvo3-nvgre-router-alert] the Inner Destination MAC Address of 00-00-5E-90-XX-XX as defined in above section, and the Destination IP Address in the 127/8 Address range for IPv4 Address, or 0:0:0:0:0:FFFF:127/104 for IPv6 Address.

The Control Plane further identifies the Overlay OAM Application by UDP well know destination port xxxx.

Since the NVGRE Router Alert bit is set in NVGRE Header, which signifies the presence of Control Packet. The Terminating Overlay End Point SHOULD not learn the Mac address set in the Inner Mac Header of NVGRE Echo Request Packet.

Once the NVGRE Echo Request Packet is identified at Control Plane, it is processed as follows:-

- o General Packet sanity is verified. If the Packet is not well-formed, NVGRE End Point SHOULD send NVGRE Echo Reply with the Return Code set to "Malformed Echo Request received" and the Subcode to zero. The header fields Originator's Handle, Sequence Number, and Timestamp Sent are not examined, but are included in the NVGRE Echo Reply message

- o VSID Validation: If there is no entry for VSID, it indicates that there could be a transient or permanent disconnect between Control Plane and data Plane and NVGRE End Point needs to report an error with Return Code of "Overlay Segment Not Present" and a Return Subcode of Zero. If the mapping for VSID Exists, but the state is not Operational, NVGRE End Point needs to report an error with Return Code of "Overlay Segment Not Operational" If the mapping exists then send NVGRE Echo Reply with a Return Code of "Return-Code-OK", and a Return Subcode of Zero. The procedures for sending the Echo Reply are found in subsection below section.

7.4.3. Sending NVGRE Echo Reply

The procedure for sending NVGRE Echo Reply are exactly same as defined in above section "Sending VXLAN Echo Reply".

7.4.4. Receiving NVGRE Echo Reply

The procedure for Receiving NVGRE Echo Reply are exactly same as defined in above section "Receiving VXLAN Echo Reply".

7.5. MPLSoGRE Procedures

7.5.1. Sending MPLSoGRE Echo Request

The Outer header of MPLSoGRE for the Echo Request packet follows the encapsulation as defined in [RFC4023]. The MPLS Stack is same as that of the MPLSoGRE Segment that is being verified. This would make sure that OAM Packet takes the same datapath as any other End System data going over this MPLSoGRE Segment.

However, the bottommost Label in MPLS Stack MUST be MPLS Router Alert Label [RFC3032]. This would indicate the Overlay Terminating End Point that the payload is a Control Packet and needs to be delivered to Control Plane.

The Encoding of Inner Header(s) and UDP payload of Generic Overlay OAM Packet is as described in above Sub-Section i.e. "Encoding of Inner Header for Echo Request in Layer 2/Layer 3 Context".

7.5.2. Receiving MPLSoGRE Echo Request

At the Terminating Overlay End Point, since the Overlay OAM Packet is exactly same as that of End-System Packet(s). It is important to send OAM packet to Control Plane and prevent it from sending to the End System. The trapping and sending MPLSoGRE Echo Request to the Control Plane is triggered by one of the following Packet processing exceptions: MPLS Router Alert Label, and the Destination IP Address in the 127/8 Address range for IPv4 Address, or 0:0:0:0:0:FFFF:127/104 for IPv6 Address.

The Control Plane further identifies the Overlay OAM Application by UDP well know destination port xxxx.

Once the MPLSoGRE Echo Request Packet is identified at Control Plane, it is processed as follows:-

- General Packet sanity is verified. If the Packet is not well-formed, MPLSoGRE End Point SHOULD send MPLSoGRE Echo Reply with the Return Code set to "Malformed Echo Request received" and the Subcode to zero. The header fields Originator’s Handle, Sequence Number, and Timestamp Sent are not examined, but are included in the MPLSoGRE Echo Reply message

- Segment Validation: If there is no entry for service represented by given Route Distinguisher for the MPLSoGRE Segment, it indicates that there could be a transient or permanent disconnect between Control Plane and Data Plane and MPLSoGRE End Point needs to report an error with Return Code of "Overlay Segment Not Present" and a Return Subcode of Zero. If the entry for service represented by given Route Distinguisher for the MPLSoGRE Segment is present, but is Operationally Down. The End Point needs to report an error with Return Code of "Overlay Segment Not Operational" If the mapping of service represented by given Route Distinguisher for the MPLSoGRE Segment is present and Active, then send MPLSoGRE Echo Reply with a Return Code of "Return-Code-OK".

7.5.3. Sending MPLSoGRE Echo Reply

The procedure for sending MPLSoGRE Echo Reply are exactly same as defined in above section "Sending VXLAN Echo Reply".

7.5.4. Receiving MPLSoGRE Echo Reply

The procedure for Receiving MPLSoGRE Echo Reply are exactly same as defined in above section "Receiving VXLAN Echo Reply".

7.6. MPLSoUDP Procedures

7.6.1. Sending MPLSoUDP Echo Request

The Outer header of MPLSoUDP for the Echo Request packet follows the encapsulation as defined in [I-D.draft-ietf-mpls-in-udp]. The MPLS Stack is same as that of the MPLSoUDP Segment that is being verified. This would make sure that OAM Packet takes the same datapath as any other End System data going over this MPLSoUDP Segment.

However, the bottommost Label in MPLS Stack MUST be MPLS Router Alert Label [RFC3032]. This would indicate the Overlay Terminating End Point that the payload is a Control Packet and needs to be delivered to Control Plane.

It is desired to choose the Source UDP port (in the outer header), so as to exercise the same Data-Path as that of the traffic carried over the MPLSoUDP Segment and is left to the implementation.

The Encoding of Inner Header(s) and UDP payload of Generic Overlay OAM Packet is as described in above Sub-Section i.e. "Encoding of Inner Header for Echo Request in Layer 2/Layer 3 Context".

7.6.2. Receiving MPLSoUDP Echo Request

At the Terminating Overlay End Point, since the Overlay OAM Packet is exactly same as that of End-System Packet(s). It is important to send OAM packet to Control Plane and prevent it from sending to the End System. The trapping and sending MPLSoGRE Echo Request to the Control Plane is triggered by one of the following Packet processing exceptions: MPLS Router Alert Label, and the Destination IP Address in the 127/8 Address range for IPv4 Address, or 0:0:0:0:0:FFFF:127/104 for IPv6 Address.

The Control Plane further identifies the Overlay OAM Application by UDP well know destination port xxxx.

Once the MPLSoUDP Echo Request Packet is identified at Control Plane, it is processed as follows:-

- o General Packet sanity is verified. If the Packet is not well-formed, MPLSoUDP End Point SHOULD send MPLSoUDP Echo Reply with the Return Code set to "Malformed Echo Request received" and the Subcode to zero. The header fields Originator's Handle, Sequence Number, and Timestamp Sent are not examined, but are included in the MPLSoUDP Echo Reply message

- o Segment Validation: If there is no entry for service represented by given Route Distinguisher for the MPLSoUDP Segment, it indicates that there could be a transient or permanent disconnect between Control Plane and data Plane and MPLSoUDP End Point needs to report an error with Return Code of "Overlay Segment Not Present" and a Return Subcode of Zero. If the entry for service represented by given Route Distinguisher for the MPLSoUDP Segment is present, but is Operationally Down. The End Point needs to report an error with Return Code of "Overlay Segment Not Operational" If the mapping of service represented by given Route Distinguisher for the MPLSoUDP Segment is present and Active, then send MPLSoUDP Echo Reply with a Return Code of "Return-Code-OK".

7.6.3. Sending MPLSoUDP Echo Reply

The procedure for sending MPLSoGRE Echo Reply are exactly same as defined in above section "Sending VXLAN Echo Reply".

7.6.4. Receiving MPLSoUDP Echo Reply

The procedure for Receiving MPLSoGRE Echo Reply are exactly same as defined in above section "Receiving VXLAN Echo Reply".

8. Procedure for Trace

In order to be able to trace the Path that a particular flow in the Overlay takes through the Underlay Network, following mechanism can be used - An overlay Echo Request packet is built and sent using the mechanisms described in the Section "Procedure for Overlay Segment Ping" so that the overlay traceroute follows the same path as the data packet for the overlay segment being traced.

The Echo Request packet in the traceroute mode is sent with the initial TTL set to 1 in the Outer IP header and thereafter incremented by 1 in each successive request. At each transit hop where the TTL expires, an exception is created. Because of this exception, the packet gets delivered to the Control Plane. Control plane can further deliver the packet to the OAM application based on

the TTL exception and the specific UDP port XXXX in the incoming overlay echo request packet. If the transit node has the IP reachability to the destination IP address in the outer IP header, it sends back an overlay echo reply response otherwise the Overlay Echo Request is discarded by the Overlay OAM module on the transit nodes. If the transit node does not support overlay OAM functionality, it will simply generate a regular ICMP TTL exceeded response. This could result into "false negatives". The originating Overlay node that generated the OAM echo request SHOULD try sending the echo request with TTL=n+1, n+2, ... to probe the nodes further down the path to the terminating overlay End-point.

At the originating node, when the Echo Reply from the transit node corresponding to the traceroute query is received, it can correlate the incoming Echo Reply with the traceroute query by matching on the sequence numbers in the Overlay Echo Request/Reply packets. Even if the intermedite node is not capable of generatin an OAM-aware reply, the ICMP TTL exceeded response SHOULD [RFC1812] include enough information of the original packet that allows the sender to identify the request that originated the received response.

Current revision of this draft limits overlay traceroute capability to fault isolation only. A subsequent version of the draft will include mechanisms to trace all possible paths in the underlay that can be used to carry overlay tunnel traffic. Implementations can use a mechanism of randomising/incrementing the source UDP port of the outer IP header as well as incrementing the TTL in order to attempt to cover multiple underlay paths followed by the encapsulated traffic. A system could increment the source UDP port 8 or 16 times, for example, before incrementing the TTL field by one, then repeating the UDP port sweet and continuing.

9. Procedure for End-System Ping

In typical Overlay deployment scenarios there is a desired to check the presence of any given Tenant VM/End-System or Flow representing the End-System System within a given Overlay Segment. This draft proposes the way to achieve it via End-System Ping.

The End-System can be identified at Overlay End Point by either its IP Address, Ethernet MAC Address or combination of IP/MAC Address, as well as an arbitrary packet.

In that case, it would be important to verify the End-System connectivity by procedure which goes over the Overlay Segment from Originating Overlay End-Point and verifies the presence of the End-System at the Terminating Overlay End-Point.

The scope of End-System Ping is solely with the Cloud Provider which owns control of the Overlay End Point(s). It is expected that the Overlay End Point traps this request and checks the Presence of the End-System via its MAC Address, Route or Flow information and replies back. There SHOULD not be a case where the End-System Ping is delivered to the actual End-Point.

9.1. Sub-TLV for End-System Ping

This section defines new set of Sub-TLVs, that needs to be added to be carried in Echo Request/Reply packets to verify presence of one of more End-System(s) which are present in Overlay Segment.

Sub-TLV Types:-

Value	What it means
1	End-System MAC Sub-TLV
2	End-System IPv4 Sub-TLV
3	End-System IPv6 Sub-TLV
4	End-System MAC/IPv4 Sub-TLV
5	End-System MAC/IPv6 Sub-TLV
6	End-System Arbitrary Packet Sub-TLV

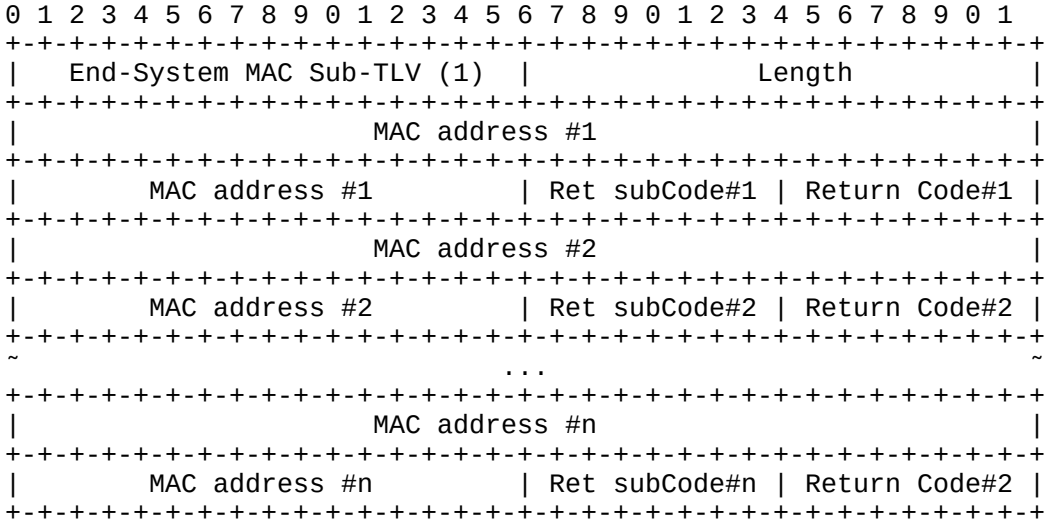
End-System Return Code:-

Value	What it means
1	End-System Present
2	End-System Not Present

End-System Return sub-Code:-

Value	What it means
0	Cannot determine action
1	End system action forward
2	End system action flood
3	End-System action dropped by rules
4	End-System action dropped by other

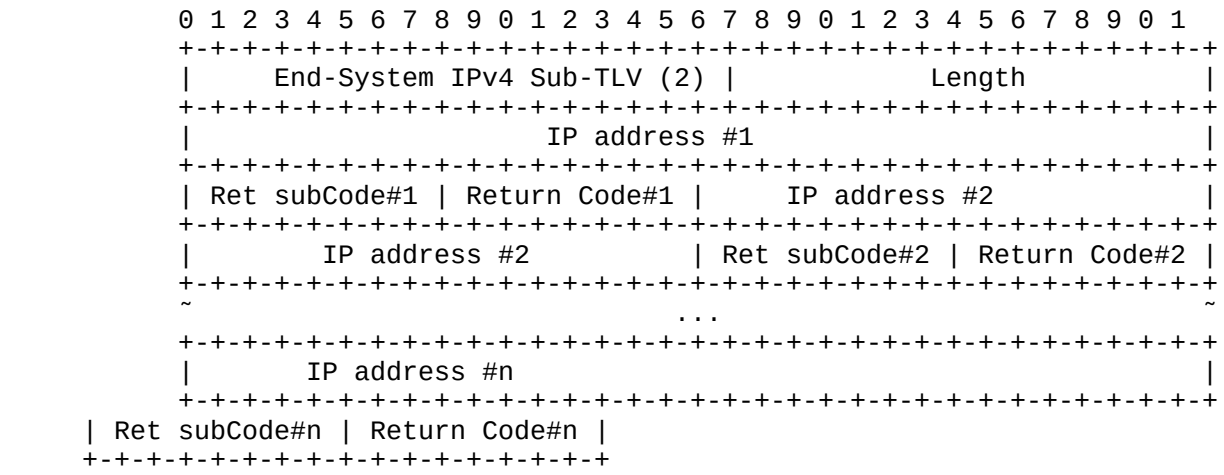
9.1.1. Sub-TLV for Validating End-System MAC Address

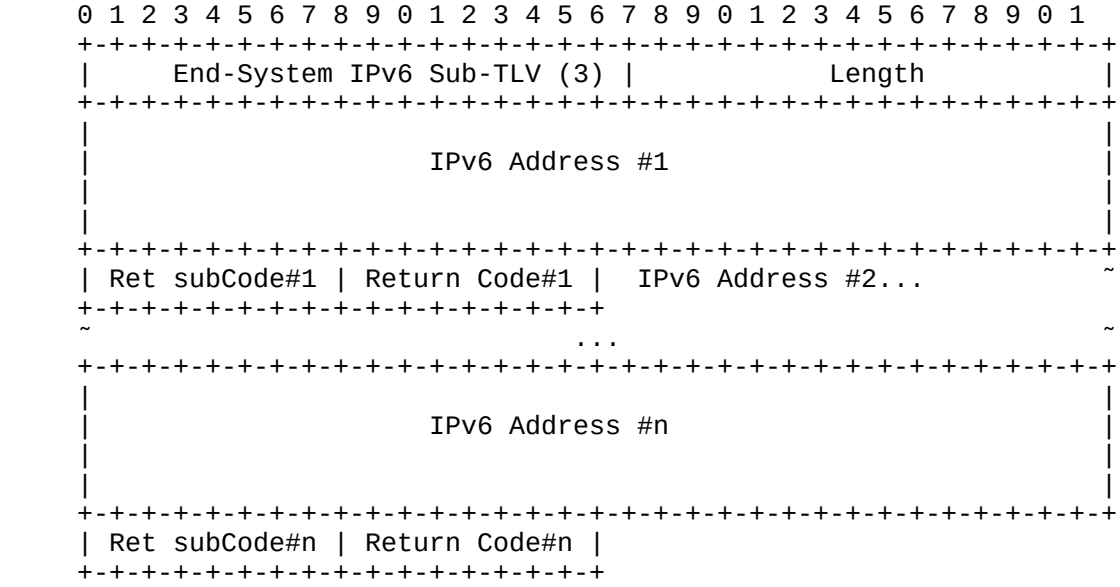


MAC Address: MAC Address of the End-System, that user is interested to validate.

Return Code: Return Code specifying status of End-System at Overlay End Point

9.1.2. Sub-TLV for Validating End-System IP Address

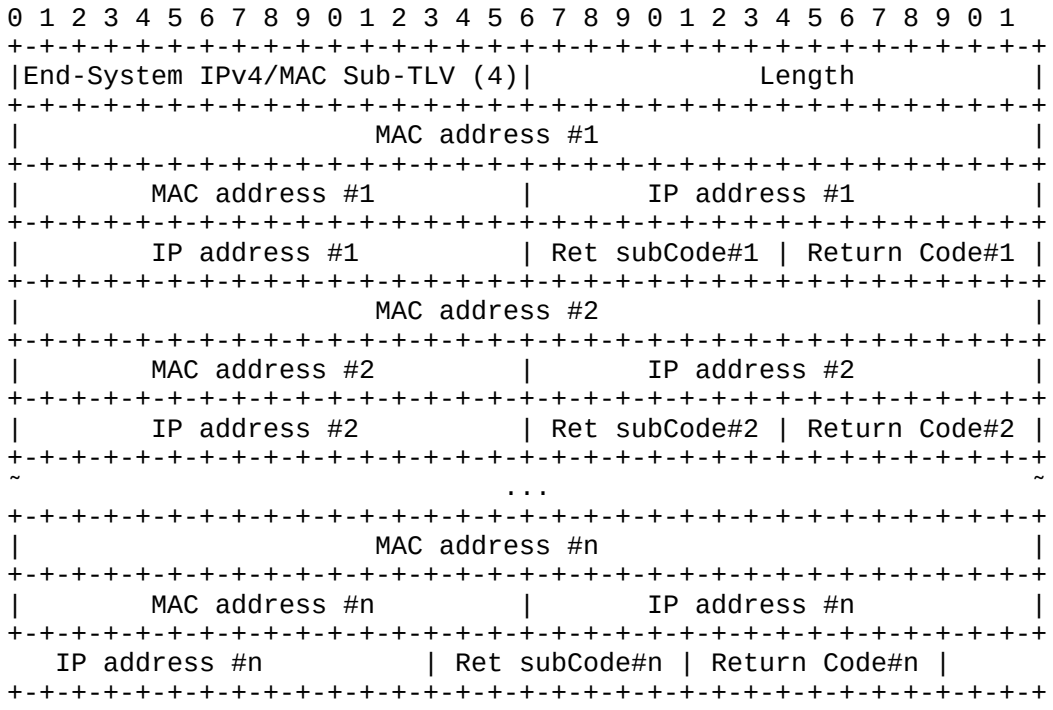


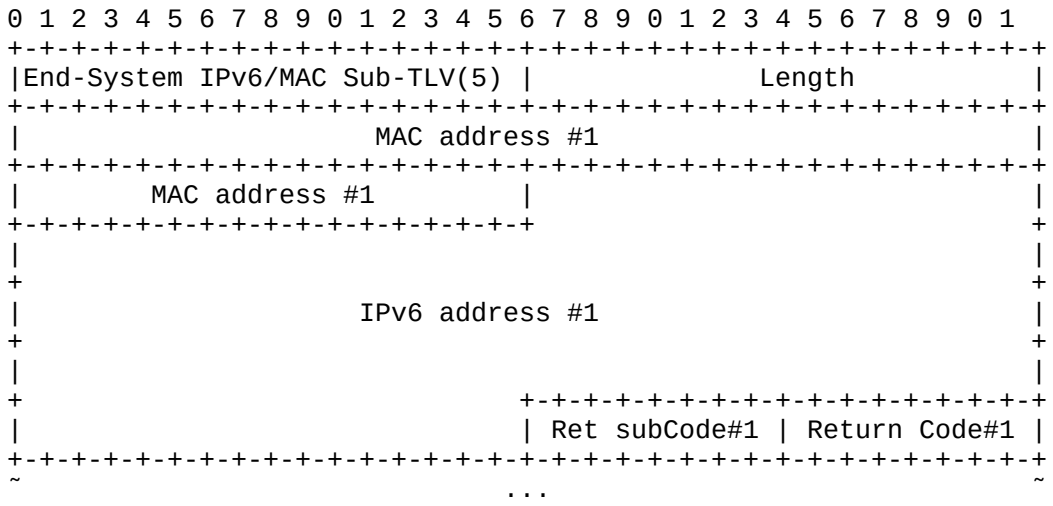


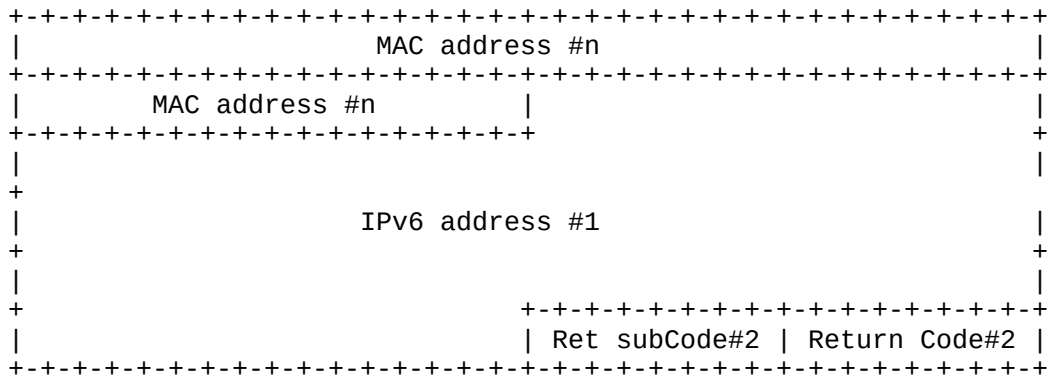
IP Address : IP Address of the End-System, that user is interested to validate.

Return Code: Return Code specifying status of End-System at Overlay End Point

9.1.3. Sub-TLV for Validating End-System MAC and IP Address

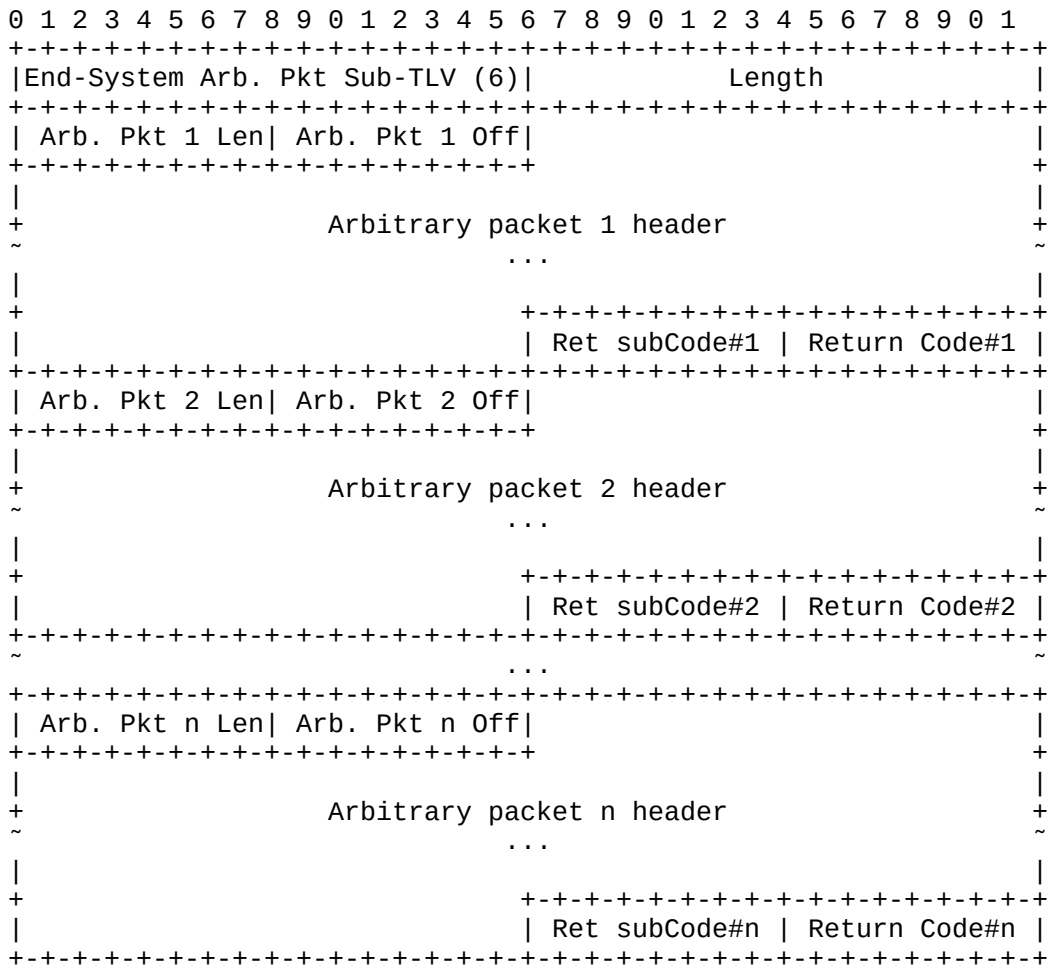






IP Address : IP Address of the End-System, that user is interested to validate.
MAC Address: MAC Address of the End-System, that user is interested to validate.
Return Code: Return Code specifying status of End-System at Overlay End Point

9.1.4. Sub-TLV for Validating End-System Arbitrary packet


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
End-System Arb. Pkt Sub-TLV (6)
Length
Arb. Pkt 1 Len
Arb. Pkt 1 Off
Arbitrary packet 1 header
...
Ret subCode#1
Return Code#1
Arb. Pkt 2 Len
Arb. Pkt 2 Off
Arbitrary packet 2 header
...
Ret subCode#2
Return Code#2
...
Arb. Pkt n Len
Arb. Pkt n Off
Arbitrary packet n header
...
Ret subCode#n
Return Code#n

Field Name explanation

Field Name	Explanation
Arb. Pkt Len	Length in bytes of the arbitrary packet header that follows. (not including the Arbitrary packet offset field)
Arb. Pkt Off	Offset from the start of a regular ethernet frame that the arbitrary data represents. This offset does not include the preamble or start-of-frame delimiter. A value of 0 represents that the data that follows is the fir Arb. Pkt Len bytes of an Ethernet frame starting by the first octect of its DA. A value of 12 means the first 2 octects of the Arbitrary Packet represent the ethertype of the test payload.
Arbitrary Paket	Arbitrary Paket: Arbitrary packet to verify on the remote end. This is a raw bitstream starting by its Destination MAC address -if the Offset is 0- and includes ethertypes, vlan-tags, DSCP values and any other part of the packet that could be used to match against an ACL, flow table or other traffic classification/filtering/forwarding element. This arbitraty packet must be of length Arb Plt Len and represents the ethernet packet present at Arbitrary Packet Offset bytes from the first byte of the Destination MAC address.
Ret subCode	return sub-code specifying the forwarding actions or drops at the Overlay End Point
Return Code	Return Code specifying status of End-System at Overlay End Point

9.2. Sending End-System Ping Request

When it is desired to check presence of a given End-System, the Echo Request Message is prepared as described in above Section "Procedure for Overlay Segment Ping". This packet should compose of Outer Header, Overlay Header, Inner Header, Generic Overlay Header with TLV representing desired Overlay Type (VXLAN, NVGRE, MPLSoGRE or MPLSoUDP). Apart form this the packet should also have one of the Sub-TLV’s as defined in above section "Sub-TLV for End-System Ping" to identify the type of End-System Ping that user is interested in.

Because of the above mentioned encapsulation, it would be guaranteed that the packet follows the same Data Path as that of any End-User data going over the given Overlay Segment.

User need to fill in MAC, IP, MAC/IP combination or the Arbitrary packet for the End-System(s) that needs to be validated at the Overlay End Point in the respective Sub-TLV for End-System Ping.

9.3. Receiving End-System Ping Request

On receiving the End-System Ping Request the processing to trap this Packet, and sent it to Control Plane is done by Overlay Terminating End-System as define in above Section "Procedure for Overlay Segment Ping". Once the OAM Packet reaches OAM Application, it is identified as End-System Ping Request by virtue of presence any of the Sub-TLV's as defined in Section "Sub-TLV for End-System Ping".

If the Sub-TLV is of Type "End-System MAC Sub-TLV", the Overlay End Point should iterate through the list of MAC Addresses and verify the presence of individual MAC Address in its Flow Table or MAC Table for the given Overlay Segment.

If the MAC Address is present, it should set the respective End-System's Return Code field in the Sub-TLV to 1 "End-System-Present".

If the MAC Address is not present, it should set respective the End-System's Return Code filed in the Sub-TLV to 2 "End-System-Not-Present".

If the Sub-TLV is of Type "End-System IP Sub-TLV", the Overlay End Point should iterate through the list of IP Addresses and verify the presence of individual IP Address in its Flow Table or Route Table for the given Overlay Segment.

If the IP Address is present, it should set the respective End-System's Return Code field in the Sub-TLV to 1 "End-System-Present".

If the IP Address is not present, it should set respective the End-System's Return Code filed in the Sub-TLV to 2 "End-System-Not-Present".

If the Sub-TLV is of Type "End-System MAC and IP Sub-TLV", the Overlay End Point should iterate through the list of MAC/IP Addresses and verify the presence of individual MAC/IP Combination in its Flow Table or MAC and IP Table for the given Overlay Segment.

If the IP and MAC Address is present, it should set the respective End-System's Return Code field in the Sub-TLV to 1 "End-System-Present".

If the IP and MAC Address is not present, it should set respective the End-System's Return Code filed in the Sub-TLV to 2 "End-System-Not-Present".

If the Sub-TLV is of Type "Arbitrary packet Sub-TLV", the Overlay End Point should iterate through the list of arbitrary packets and verify the presence of individual MAC/Ethertype/VLAN/IP/DSCP/etc Combination in its Flow Table or forwarding tables for the given Overlay Segment. Unused bytes (from a non-zero offset field or short arbitrary packet) should be filled in with 0x00 for whatever fields/bits are needed in order for the system to perform a flow or forwarding table lookup.

If the arbitrary packet is present, it should set the respective End-System's Return Code field in the Sub-TLV to 1 "End-System-Present".

If the arbitrary packet is deemed not present, it should set respective the End-System's Return Code filed in the Sub-TLV to 2 "End-System-Not-Present".

In general, for the TEPs supporting more advanced diagnostics and/or packet match simulation capabilities, the return sub-code SHOULD be set based on the expected fate of the packet according to the following guidelines.

If the provided information (be it MAC, IPv4, IPv6, a combination of MAC/IPv4, MAC/IPv6 or an arbitrary packet) is enough to determine the fate of a hypothetical packet with those addresses and other arbitrary fields, then the expected action SHOULD be reported back to the originator.

If the fate of the packet can not be properly determined, then the respective End-System's sub-Return code should be set to 0, "Cannot determine action"

If the provided information is enough to determine that the packet would be forwarded to the End-System, then the corresponding sub-Return code should be set to 1, "End system action forward"

If the provided information can determine that the packet would be floded (for example, due to a MAC address not present in the forwarding tables and requiring flooding to all ports), then the corresponding sub-Return code should be set to 2, "End system action flood"

If the information provided can determine that the packet would be dropped by ACL rules configured in the system, then the corresponding sub-Return code should be set to 3, "End system action dropped by rules"

Finally, if the information provided can determine that the packet would be dropped by other rules (for example, a configuration setting to disable the flooding of unkwnon packets or such as an anti-spoof filter) then the correspoinding sub-Return code should be set to 4, "End system action dropped by others"

9.4. Sending End-System Ping Reply

The procedure for sending End-System Echo Reply is same as defined in above section "Sending VXLAN Echo Reply". The replier MUST fill Sub-TLV with proper Return Code and sub-code for each element in the End-System Sub-TLV.

9.5. Receiving End-System Ping Reply

An Originating Overlay End Point should only receive Echo Reply for End-System Ping, in response to an Echo Request that it sent. By virtue of presence of End-System Sub-TLV it would identify the status of respective End-System, and report it to the user. The other part of the handling is similar to section "Receiving VXLAN Echo Reply"

10. Security Considerations

TBD

11. Management Considerations

None

12. Acknowledgements

This document is the outcome of many discussions among many people, including Saurabh Shrivastava, Krishna Ram Kuttuva Jeyaram and Suresh Boddapati of Nuage Networks, Jorge Rabadan of Alcatel-Lucent, Inc and Rahul Kasralikar of Juniper Networks, Inc.

13. IANA Considerations

Action-1: This specification reserves a IANA UDP Port Number to be used when sending the Overlay OAM Packet

Action-2: This specification reserves a IANA Ethernet unicast Address for VXLAN/NVGRE Exception handling. This Address needs to be reserved from the block. "IANA Ethernet Address block - Unicast Use"

14. References

14.1. Normative References

[I-D.draft-ietf-mpls-in-udp]
Xu, , Sheth, , Yong, , Pignataro, , Yongbing , , and Li, "Encapsulating MPLS in UDP", May 2013.

[I-D.draft-lasserre-nvo3-framework]
Lasserre, M., Balus, F., Morin, T., Bitar, N., and Y. Rekhter, "Framework for DC Network Virtualization", September 2011.

[I-D.draft-singh-nvo3-nvgre-router-alert]
Singh, K., Jain, P., Balus, F., and W. Henderickx, "NVGRE Router Alert Option", May 2013.

[I-D.draft-singh-nvo3-vxlan-router-alert]
Singh, K., Jain, P., Balus, F., and W. Henderickx, "VxLAN Router Alert Option", May 2013.

[I-D.draft-sridharan-virtualization-nvgre]
Sridharan, M., Duda, K., Ganga, I., Greenberg, A., Lin, G., Pearson, M., Thaler, P., Tumuluri, C., Venkataramiah, N., and Y. Wang, "NVGRE: Network Virtualization using Generic Routing Encapsulation", February 2013.

[RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, January 2001.

[RFC4023] Worster, T., Rekhter, Y., and E. Rosen, "Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)", RFC 4023, March 2005.

[RFC4365] Rosen, E., "Applicability Statement for BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4365, February 2006.

[RFC4379] Kompella, K. and G. Swallow, "Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures", RFC 4379, February 2006.

[RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, August 2014.

14.2. Informative References

[RFC1812] Baker, F., "Requirements for IP Version 4 Routers", RFC 1812, June 1995.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4330] Mills, D., "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC 4330, January 2006.

Authors' Addresses

Pradeep Jain
Nuage Networks
755 Ravendale Drive
Mountain View, CA 94043
USA

Email: pradeep@nuagenetworks.net

Kanwar Singh
Nuage Networks
755 Ravendale Drive
Mountain View, CA 94043
USA

Email: kanwar@nuagenetworks.net

Diego Garcia del Rio
Nuage Networks
755 Ravendale Drive
Mountain View, CA 94043
USA

Email: diego@nuagenetworks.net

Wim Henderickx
Alcatel-Lucent
Copernicuslaan 50
Antwerp 2018
Belgium

Email: wim.henderickx@alcatel-lucent.be

Vinay Bannai
PayPal
2211 N. First St,
San Jose 95131
USA

Email: vbannai@paypal.com

Ravi Shekhar
Juniper Networks
1194 North Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: rshekhar@juniper.net

Anil Lohiya
Juniper Networks
1194 North Mathilda Ave.
Sunnyvale, CA 94089
USA

Email: alohiya@juniper.net

NVO3 WG
Internet-Draft
Intended status: Standards Track
Expires: September 3, 2015

E. Nordmark
C. Appanna
A. Lo
Arista Networks

March 2, 2015

Layer-Transcending Traceroute for Overlay Networks like VXLAN
draft-nordmark-nvo3-transcending-traceroute-00

Abstract

Tools like traceroute have been very valuable for the operation of the Internet. Part of that value comes from being able to display information about routers and paths over which the user of the tool has no control, but the traceroute output can be passed along to someone else that can further investigate or fix the problem.

In overlay networks such as VXLAN and NVGRE the prevailing view is that since the overlay network has no control of the underlay there needs to be special tools and agreements to enable extracting traces from the underlay. We argue that enabling visibility into the underlay and using existing tools like traceroute has been overlooked and would add value in many deployments of overlay networks.

This document specifies an approach that can be used to make traceroute transcend layers of encapsulation including details for how to apply this to VXLAN. The technique can be applied to other encapsulations used for overlay networks. It can also be implemented using current commercial silicon.

Table of Contents

1. Introduction

Tools like traceroute have been very valuable for the operation of the Internet. Part of that value comes from being able to display information about routers and paths over which the user of the tool has no control, but the traceroute output can be passed along to someone else that can further investigate or fix the problem. The output of traceroute can be included in an email or a trouble ticket to report the problem. This provide a lot more information than the mere indication that A can't communicate with B, in particular when the failures are transient. The ping tool provides some of the same benefits in being able to return ICMP errors such as host unreachable messages.

This document shows how those tools can be used to gather information for both the overlay and underlay parts of an end-to-end path by providing the option to have some packets use a uniform time-to-live (ttl) model for the tunnels, and associated ICMP error handling. These changes are limited to the tunnel ingress and egress points.

The desire to make traceroute provide useful information for overlay network is not an argument against also using a layered approach for OAM as specified in e.g., [I-D.tissa-lime-yang-oam-model]. Such approaches are quite appropriate for continuos monitoring at different layers and across different domains. A layer transcending traceroute complements the ability to do layered and/or continuos monitoring.

The traceroute tool relies on receiving ICMP errors [RFC0792] in combination with using different IP time-to-live values. That results in the packet making it further and further towards the destination with ICMP ttl exceeded errors being received from each hop. That provides the user the working path even if the packets are black holed eventually, and also provides any errors like ICMP host unreachable. The fundamental assumption is that the ttl is decremented for each hop and that the resulting ICMP ttl exceeded errors are delivered back to the host.

When some encapsulation is used to tunnel packets there is an architectural question how those tunnels should be viewed from the rest of the network. Different models were described first for diffserv in [RFC2983] and then applied to MPLS in [RFC3270] and expanded to MPLS ttl handling in [RFC3443] and those models apply to other forms of direct or indirect IP in IP tunnels. Those RFCs define two models for ttl that are of interest to us:

- A pipe model, where the tunnel is invisible to the rest of the network in that it looks like a direct connection between the

tunnel ingress and egress.

- o A uniform model, where the ttl decrements uniformly for hops outside and inside the tunnel.

The tunneling mechanisms discussed in NVO3 (such as VXLAN [RFC7348], NVGRE [I-D.sridharan-virtualization-nvgre], GENEVE [I-D.gross-geneve], and GUE [I-D.herbert-gue]), have either been specified to provide the pipe model of a tunnel or are silent on the setting of the outer ttl. Those protocols can be extended to have an optional uniform tunnel model when the payload is IP, following the same model as in [RFC3443]. Note that these encapsulations carry Ethernet frames hence are not even aware that the payload is IP. However, IP is the bulk of what is carried over such tunnels and the ingress NVE can inspect the IP part of the Ethernet frame.

However, for general application traffic the pipe model is fine and might even be expected by some applications. In general, when the source and destination IP are in the same IP subnet the ttl should not be decremented. Thus it makes sense to have a way to selectively enable the uniform model perhaps based on some method to identify packets associated with traceroute or some marker in the packet itself that the traceroute tool can set.

2. Solution Overview

The pieces needed to accomplish this are:

- o One or more ways to select the uniform model packets at the tunnel ingress.
- o Tunnel ingress copying out the original ttl from a selected packet to the outer IP header, and then doing a check and decrement of that ttl.
- o If that ttl check results in ttl expiry at the tunnel ingress, then deliver an ICMP ttl exceeded packet back to the host.
- o A mechanism by which the tunnel egress knows which packets should have uniform model, for instance a bit in the encapsulation header.
- o The tunnel egress copying in the ttl (for identified packets) from the outer header to the inner IP header, then doing a check and decrement of that ttl.

- o If ttl check results in ttl expiry at the tunnel egress, then deliver an ICMP error back to the original host (or, perhaps better, to tunnel ingress the same way as underlay routers do).

- o IP routers in the underlay will deliver any ICMP errors to the source IP address of the packet. For tunneled packets that will be the tunnel ingress. Hence the tunnel ingress needs to be able to take such ICMP errors and form corresponding ICMP errors that are sent back to the host. The requirement in [RFC1812] ensures that the ICMP errors will contain enough headers to form such an ICMP error.

The idea to reflect (some) ICMP errors from inside a tunnel back to the original source goes back to IPv6 in IPv4 encapsulation as specified in [RFC1933] and [RFC2473]. However, those drafts did not advocate using a uniform ttl model for the tunnels but did handle ICMP packet too big and other unreachable messages. Those drafts specify how to reflect ICMP errors received from underlay routers to ICMP errors sent to the original host. The addition of handling ICMP ttl exceeded errors for uniform tunnel model is straight forward.

The information carried in the ICMP errors are quite limited - the original packet plus an ICMP type and code. However, there are extension mechanisms specified in [RFC4884] and used for MPLS in [RFC4950] which include TLVs with additional information. If there are additional information to include for overlay networks that information could be added by defining new ICMP Extensions Objects based on [RFC4884]. Such extensions are for further study.

3. Goals and Requirements

The following goals and requirements apply:

- o No changes needed in the underlay.

- o Optional changes on the decapsulating end.

- o ECMP friendly. If the underlay employs equal cost multipath routing then one should be able to use this mechanism to trace the same path as a given TCP or UDP flow is using. In addition, one should be able to explore different ECMP paths by varying the IP addresses and port numbers in the packets originated by traceroute on the host.

- o Provide output which makes it possible to compare a regular overlay traceroute with the layer-transcending output.

4. Definition Of Terms

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

The terminology such as NVE, and TS are used as specified in [RFC7365]:

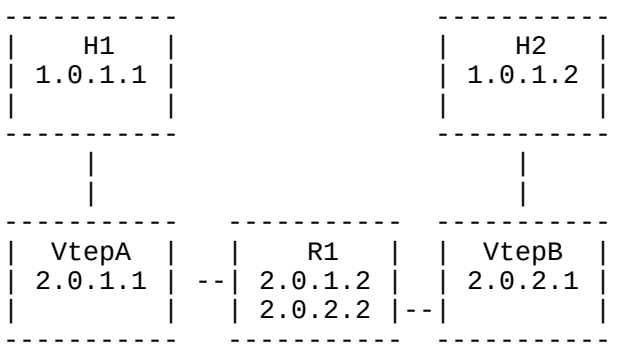
- Network Virtualization Edge (NVE): An NVE is the network entity that sits at the edge of an underlay network and implements L2 and/or L3 network virtualization functions.
- Tenant System (TS): A physical or virtual system that can play the role of a host or a forwarding element such as a router, switch, firewall, etc.
- Virtual Access Points (VAPs): A logical connection point on the NVE for connecting a Tenant System to a virtual network.
- Virtual Network (VN): A VN is a logical abstraction of a physical network that provides L2 or L3 network services to a set of Tenant Systems.
- Virtual Network Context (VN Context) Identifier: Field in an overlay encapsulation header that identifies the specific VN the packet belongs to.

We use the VTEP term in [RFC7348] as synonymous with NVE, and VNI as synonymous to VN Context Identifier.

5. Example Topologies

The following example topologies illustrate different cases where we want a tracing capability. The examples are for overlay technologies such as VXLAN which provide a layer 2 overlay on IP. The cases for layer 3 overlay on top of IP are simpler and not shown in this document.

The VXLAN term VTEP is used as synonymous to NVO3’s NVE term.



Simple L2 overlay

The figure above shows two hosts connected using an underlay which provides a layer two service. Thus H1 and H2 are in the same subnet and unaware of the existence of the underlay. Thus a normal ping or traceroute would not be able to provide any information about the nature of a failure; either packets get through or they do not. When the packets get through traceroute would output something like:

```
traceroute to 1.0.1.2 (1.0.1.2), 30 hops max, 60 byte packets
 1  1.0.2.1 (1.0.2.1)  1.104 ms  1.235 ms  1.729 ms
```

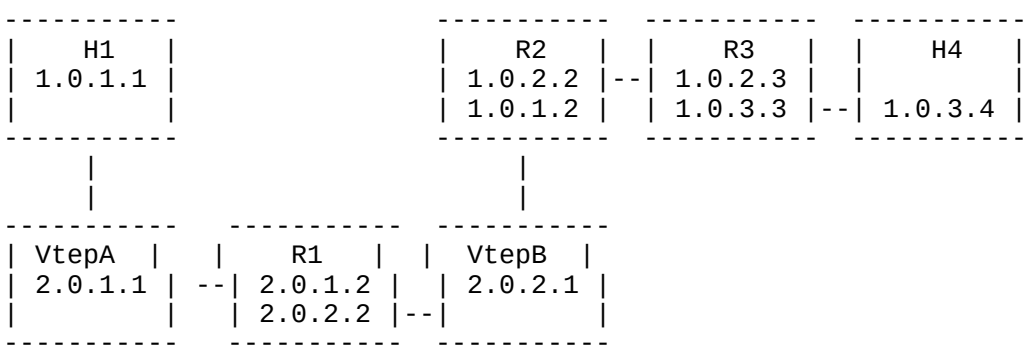
In this case it would be desirable to be able to traceroute from H1 to H2 (and vice versa) and observe VtepA, R1, VtepB and H2. Thus in the case of packets getting through traceroute would output:

```
traceroute to 1.0.1.2 (1.0.1.2), 30 hops max, 60 byte packets
 1  2.0.1.1 (2.0.1.1)  1.104 ms  1.235 ms  1.729 ms
 2  2.0.1.2 (2.0.1.2)  2.106 ms  2.007 ms  2.156 ms
 3  2.0.2.1 (2.0.2.1)  35.034 ms  24.490 ms  21.626 ms
 4  1.0.1.2 (1.0.1.2)  40.830 ms  44.694 ms  75.620 ms
```

Note that the underlay and overlay might exist in completely separate addressing domains. Thus H1 might not be able to reach any of the underlay addresses. And the underlay IP addresses might overlap the overlay IP addresses. For example, it would be completely valid to see e.g. VtepA having the same IP address as H1. The user of this tool need to understand that the utility of the traceroute output is to get information to determine whether the issue is in the underlay or overlay, and be able to pass the underlay information to the operator of the underlay.

In overlay networks without any ARP/ND optimizations ARP/ND packets would be flooded between the tunnel endpoints. Thus if there is some communication failure between H1 and H2, then H1 above might not have

an ARP entry for H2. This results in traceroute not being able to output any data. This implies that in order to use traceroute to trouble shoot the issue one would need some workaround, such as installing some temporary ARP entries on the hosts.



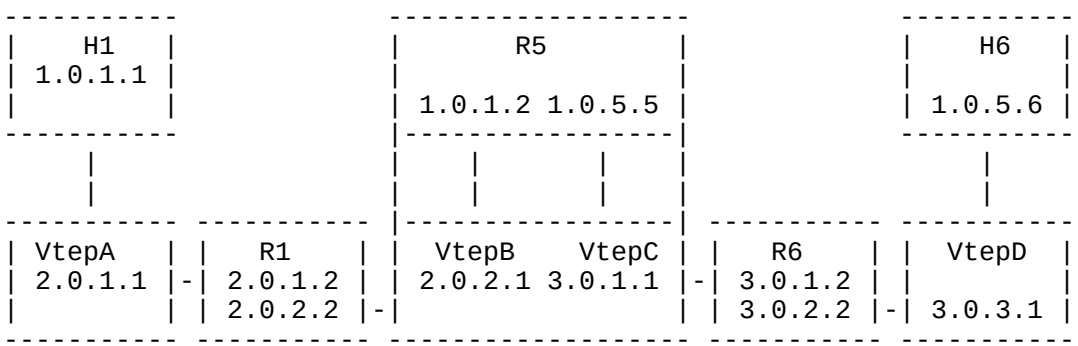
L2 overlay as part of larger network

The figure above has a overlay router the nexthop as seen by H1. In this case a normal overlay traceroute would be able to display the overlay path i.e.

```
traceroute to H4, 30 hops max, 60 byte packets
 1  R2
 2  R3
 3  H4
```

The layer-transcending traceroute would show the combination of the underlay and overlay paths i.e.,

```
traceroute to H4, 30 hops max, 60 byte packets
 1  VtepA
 2  R1
 3  VtepB
 4  R2
 5  R3
 6  H4
```



Multiple L2 overlays in path

The figure above has multiple overlay network segments, that are connected in one router which provides the tunnel endpoints for both overlay segments plus routing for the overlay. A more general picture would be to have an overlay routed path between the two NVEs e.g., VtepB and VtepC connected to different routers in the overlay. However, such a drawing in ASCII art doesn't fit on the page.

An normal overlay traceroute in the above topology would show the overlay router i.e.,

```
traceroute to H6, 30 hops max, 60 byte packets
 1  R5
 2  H6
```

The layer-transcending traceroute would show the combination of the underlay and overlay paths i.e.,

```
traceroute to H6, 30 hops max, 60 byte packets
 1  VtepA
 2  R1
 3  VtepB
 4  R5
 5  VtepC
 6  R6
 7  VtepD
 8  H6
```

Note that the R3 device, which include VtepB and VtepC, appears as three hops in the traceroute output. That is needed to be able to correlate the output with the overlay output which has R3. That correlation would be hard if the R3 device only appeared as VtepB in the LTTON output. The three-hop representation also stays invariant whether or not the NVEs and overlay router are implemented by a

single device or multiple devices.

6. Controlling and selecting ttl behavior

The network admin needs to be able to control who can use the layer transcending traceroute, since the operator might not want to disclose the underlay topology to all its users all the time. There are different approaches for this such as designating particular ports (Virtual Access Points in NVO3 terminology) on a NVE to have uniform ttl tunnel model. We have found it useful to be able to enable this capability on a per port and/or virtual network basis, in addition to having a global setting per NVE.

When enabled on the NVEs the user on the TS needs to be able to control which traffic is subject to which tunnel mode. The normal traffic would use the pipe ttl tunnel model and only explicit trace applications are likely to want to use the uniform ttl tunnel model. Hence it makes sense to use some marker in the packets sent by the TS to select those packets for uniform model on the NVE. Such a mechanism should usable so that the user can perform both a regular traceroute and a LTTON.

Potentially different fields in the packets originated by traceroute on the TS can be used to mark the packets for uniform ttl tunnel model. However, many of those fields such as source and destination port numbers and protocol might be used in hashing for ECMP. The marking that can be used without impacting ECMP is the DSCP field in the packet. That field can be set with an option (--tos) in at least some existing traceroute implementations.

Note that when DSCP is used for such marking it is a configured choice subject to agreement between the operator of the TS and NVE. The matching on the NVE should ignore the ECN bits as to not interfere with ECN.

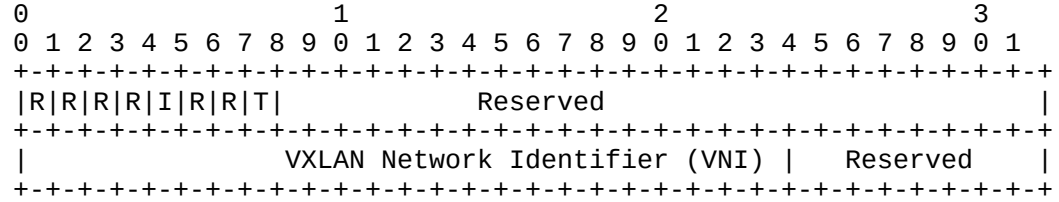
However, the DSCP value used in the overlay might have an impact on the forwarding of the packets. In such a case one can use an alternative selector such as the UDP source port number. That has the downside of affecting the has values used for ECMP and link aggregation port selection.

7. Introducing a ttl copyin flag in the encapsulation header

When this approach is applied to VXLAN [RFC7348] the decapsulating NVE has to be able to identify packets that have to be processed in the uniform ttl tunnel model way. For that purpose we define a new

flag which is sent by the encapsulating NVE on selected packets, and is used by the decapsulating NVE to perform the ttl copyin, decrement and check.

In addition to the one I-flag defined in [RFC7348] we define a new T-flag to capture this the trace behavior at the decapsulating tunnel endpoint.



New fields:

T-flag: When set indicates that decapsulator should take the outer ttl and copy it to the inner ttl, and then check and decrement the resulting ttl.

8. Encapsulation Behavior

If the uniform ttl model is enabled for the input, and the received naked packet matches the selector, then the ingress NVE will perform these additional operations as part of encapsulating an IPv4 or IPv6 packet:

- Examine the IPv4 TTL (or IPv6 hopcount, respectively) on receipt and if 1 or less, then drop the packet and send an ICMPv4 (or ICMPv6) ttl exceeded back to the original host. Since the NVE is operating on a L2 packet, it might not have any layer 3 interfaces or routes for the originating host. Thus it sends the packet back to the source L2 address of the packet back out the ingress port - without any IP address lookup.

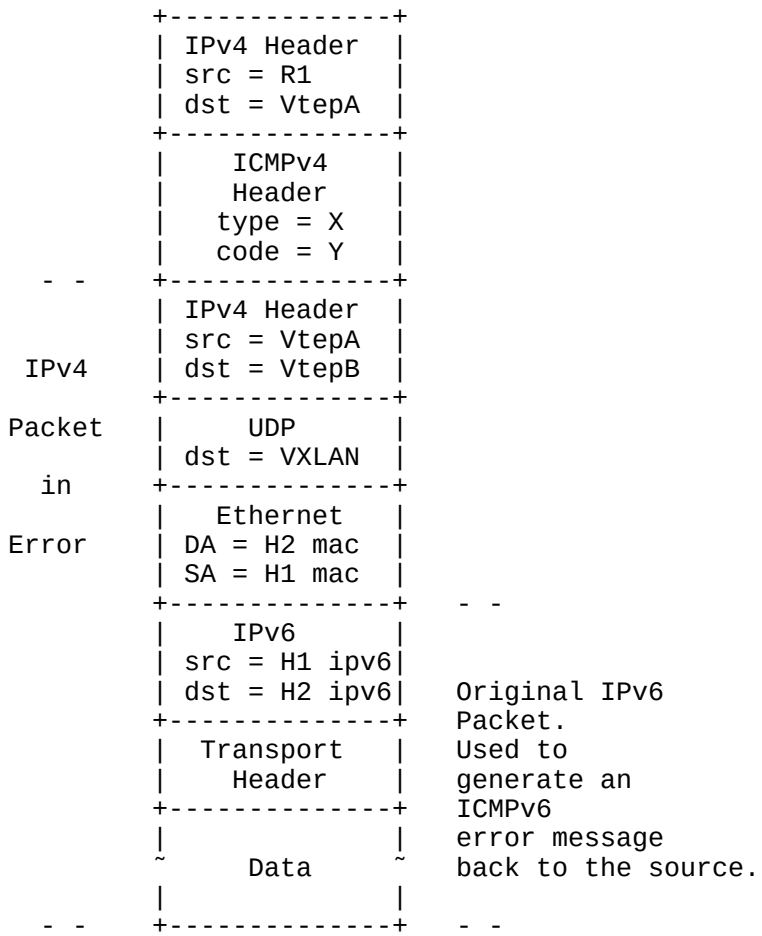
- If ttl did not expire, then decrement the above ttl/hopcount and place it in the outer IP header. Encapsulate and send the packet as normal.

- If some other errors prevent sending the packet (such as unknown VN Context Id, no flood list configured), then the NVE SHOULD send an ICMP host unreachable back to the host.

The ingress NVE will receive ICMP errors from underlay routers and the egress NVE; whether due to ttl exceeded or underlay issues such

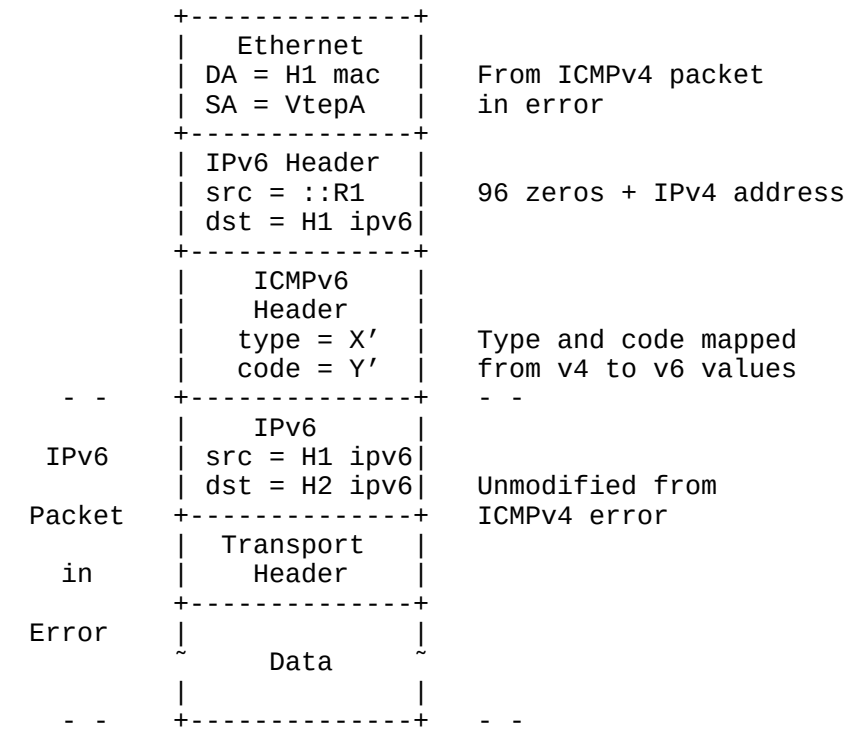
as host unreachable, or packet too big errors. The NVE should take such errors, and in addition to any local syslog etc, generate an ICMP error sent back to the host. The principle for this is specified in [RFC1933] and [RFC2473]. Just like in those specifications, for the inner and outer IP header could be off different version. A common case of that might be an IPv6 overlay with an IPv4 underlay. That case requires some changes in the ICMP type and code values in addition to recreating the packets. The place where LTTON differs from those specifications is that there is an NVO3 header and (for L2 over L3) and L2 header in the packet.

The figures below show an example of ICMP header re-generation at VtepA for the case of IPv6 overlay with IPv4 underlay. The case of IPv4 over IPv4 is similar and simpler since the ICMP header is the same for both overlay and underlay. The example uses VXLAN encapsulation to provide the concrete details, but the approach applies to other NVO3 proposals.



ICMPv4 Error Message Returned to Encapsulating Node

The above underlay ICMPv4 is used to form an overlay ICMPv6 packet by extracting the Ethernet DA from the inner Ethernet SA, and forming an IPv6 header where the source address is based on the source address of the ICMPv4 error. The ICMPv6 type and code values are set based on the ICMPv4 type and code values.



Generated ICMPv6 Error Message for Overlay Source

In the case of IPv6 over IPv4 the above example setting of the IPv6 source address results in this type of traceroute output:

```
traceroute to 2000:0:0:40::2, 30 hops max, 80 byte packets
 1  ::2.0.1.1 (::2.0.1.1)  1.231 ms  1.004 ms  1.126 ms
 2  ::2.0.1.2 (::2.0.1.2)  1.994 ms  2.301 ms  2.016 ms
 3  ::2.0.2.1 (::2.0.2.1)  18.846 ms  30.582 ms  19.776 ms
 4  2000:0:0:40::2 (2000:0:0:40::2)  48.964 ms  60.131 ms  53.895 ms
```

9. Decapsulating Behavior

If this uniform ttl model is enabled on the decapsulating NVE, and the overlay header indicates that uniform ttl model applies (the T-bit in the case of VXLAN), then the NVE will perform these additional operations as part of decapsulating a packet where the inner packet is an IPv4 or IPv6 packet:

- Examine the outer IPv4 TTL (or outer IPv6 hopcount, respectively) on receipt and if 1 or less, then drop the packet and send an

outer ICMPv4 (or ICMPv6) ttl exceeded back to the source of the outer packet i.e., the ingress NVE. This ICMP packet should look the same as an ICMP error generated by an underlay router, and the requirement in [RFC1812] on the size of the packet in error applies.

- o If ttl did not expire, then decrement the above ttl/hopcount and place it in the inner IP header. If the inner IP header is IPv4 then update the IPv4 header checksum. Then decapsulate and send the packet as for other decapsulated packets.

- o If some other errors prevent sending the packet (such as unknown VN Context Id), then the NVE SHOULD send an ICMP host unreachable instead of a ttl exceeded error.

10. Other ICMP errors

The technique for selecting ttl behavior specified in this draft can also be used to trigger other ICMPv4 and ICMPv6 errors. For example, [RFC1933] specifies how ICMP packet too big from underlay routers can be used to report over ICMP packet too big errors to the original source. Other errors that are more specific to the overlay protocol might also be useful, such as not being able to find a VNI ID for the incoming port,vlan, or not being able to flood the packet if the packet is a Broadcast, Unknown unicast, or Multicast packet.

11. Security Considerations

The considerations in [I-D.ietf-nvo3-security-requirements] apply.

In addition, the use of the uniform ttl tunnel model will result in ICMP errors being generated by underlay routers and consumed by NVEs. That presents an attack vector which does not exist in a pipe ttl tunnel model. However, ICMP errors should be rate limited [RFC1812]. Implementations should also take appropriate measures in rate limiting the input rate for ICMP errors that are processed by limited CPU resources.

Some implementations might handle the trace packets (with uniform ttl model) in software while the pipe ttl model packets can be handled in hardware. In such a case the implementation should have mechanisms to avoid starvation of limited CPU resources due to these packets.

12. IANA Considerations

TBD

13. Acknowledgements

The authors acknowledge the helpful comments from David Black and Diego Garcia del Rio.

14. References

14.1. Normative References

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC1812] Baker, F., "Requirements for IP Version 4 Routers", RFC 1812, June 1995.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, August 2014.

[RFC7365] Lasserre, M., Balus, F., Morin, T., Bitar, N., and Y. Rekhter, "Framework for Data Center (DC) Network Virtualization", RFC 7365, October 2014.

14.2. Informative References

[I-D.gross-geneve]
Gross, J., Sridhar, T., Garg, P., Wright, C., Ganga, I., Agarwal, P., Duda, K., Dutt, D., and J. Hudson, "Geneve: Generic Network Virtualization Encapsulation", draft-gross-geneve-02 (work in progress), October 2014.

[I-D.herbert-gue]
Herbert, T. and L. Yong, "Generic UDP Encapsulation", draft-herbert-gue-02 (work in progress), October 2014.

[I-D.ietf-nvo3-security-requirements]
Hartman, S., Zhang, D., Wasserman, M., Qiang, Z., and M.

Zhang, "Security Requirements of NVO3", draft-ietf-nvo3-security-requirements-04 (work in progress), January 2015.

[I-D.sridharan-virtualization-nvgre] Garg, P. and Y. Wang, "NVGRE: Network Virtualization using Generic Routing Encapsulation", draft-sridharan-virtualization-nvgre-07 (work in progress), November 2014.

[I-D.tissa-lime-yang-oam-model] Senevirathne, T., Finn, N., Kumar, D., Salam, S., and Q. Wu, "Generic YANG Data Model for Operations, Administration, and Maintenance (OAM)", draft-tissa-lime-yang-oam-model-03 (work in progress), November 2014.

[RFC1933] Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 1933, April 1996.

[RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, December 1998.

[RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000.

[RFC3270] Le Faucheur, F., Wu, L., Davie, B., Davari, S., Vaananen, P., Krishnan, R., Cheval, P., and J. Heinanen, "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services", RFC 3270, May 2002.

[RFC3443] Agarwal, P. and B. Akyol, "Time To Live (TTL) Processing in Multi-Protocol Label Switching (MPLS) Networks", RFC 3443, January 2003.

[RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", RFC 4884, April 2007.

[RFC4950] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "ICMP Extensions for Multiprotocol Label Switching", RFC 4950, August 2007.

Authors' Addresses

Erik Nordmark
Arista Networks
Santa Clara, CA
USA

Email: nordmark@arista.com

Chandra Appanna
Arista Networks
Santa Clara, CA
USA

Email: achandra@arista.com

Alton Lo
Arista Networks
Santa Clara, CA
USA

Email: altonlo@arista.com