NVO3 Working Group
INTERNET-DRAFT
Intended Status: Informational

H. Chen, Ed.
P. Ashwood-Smith
L. Xia
Huawei Technologies
R. Iyengar
T. Tsou
Huawei Technologies USA
A. Sajassi
Cisco Technologies
M. Boucadair
C. Jacquenet
France Telecom
M. Daikoku
KDDI corporation
A. Ghanwani
Dell
R. Krishnan
Brocade

Expires: April 21, 2016

October 19, 2015

NVO3 Operations, Administration, and Maintenance Requirements
draft-ashwood-nvo3-oam-requirements-04

Abstract

This document provides framework and requirements for Network Virtualization Overlay (NVO3) Operations, Administration, and Maintenance (OAM).

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

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

Table of Contents

1. Introduction

This document provides framework and requirements for Network Virtualization Overlay (NVO3) Operations, Administration, and Maintenance (OAM). Given that this OAM subject is far from new and has been under extensive investigation by various IETF working groups (and several other standards bodies) for many years, this document draws from existing work, starting with [RFC6136]. As a result, sections of [RFC6136] have been reused with minor changes with the permission of the authors.

NVO3 OAM requirements are expected to be a subset of IETF/IEEE etc. work done so far; however, we begin with a full set of requirements and expect to prune them through several iterations of this document.

1.1. OSI Definitions of OAM

The scope of OAM for any service and/or transport/network infrastructure technologies can be very broad in nature. OSI has defined the following five generic functional areas commonly abbreviated as "FCAPS" [NM-Standards]:

- Fault Management,
- Configuration Management,
- Accounting Management,
- Performance Management, and
- Security Management.

This document focuses on the Fault, Performance and to a limited extent the Configuration Management aspects. Other functional aspects of FCAPS and their relevance (or not) to NVO3 are for further study.

Fault Management can typically be viewed in terms of the following categories:

- Fault Detection;
- Fault Verification;
- Fault Isolation;
- Fault Notification and Alarm Suppression;

- Fault Recovery.

Fault detection deals with mechanism(s) that can detect both hard failures such as link and device failures, and soft failures, such as software failure, memory corruption, misconfiguration, etc. Fault detection relies upon a set of mechanisms that first allow the observation of an event, then the use of a protocol to dynamically notify a network/system operator (or management system) about the event occurrence, then the use of diagnostic tools to assess the nature and severity of the fault.

After verifying that a fault has occurred along the data path, it is important to be able to isolate the fault to the level of a given device or link. Therefore, a fault isolation mechanism is needed in Fault Management. A fault notification mechanism should be used in conjunction with a fault detection mechanism to notify the devices upstream and downstream to the fault detection point. The fault notification mechanism should also notify NMS systems.

The terms "upstream" and "backward" are used here to denote the direction(s) from which data traffic is flowing. The terms "downstream" and "forward" denote the direction(s) to which data traffic is forwarded.

For example, when there is a client/server relationship between two layered networks (e.g., the NVO3 layer is a client of the outer IP server layer, while the inner IP layer is a client of the NVO3 server layer 2), fault detection at the server layer may result in the following fault notifications:

- Sending a forward fault notification from the server layer to the client layer network(s) using the fault notification format appropriate to the client layer.

- Sending a backward fault notification to the server layer, if applicable, in the reverse direction.

- Sending a backward fault notification to the client layer, if applicable, in the reverse direction.

Finally, fault recovery deals with recovering from the detected failure by switching to an alternate available data path (depending on the nature of the fault) using alternate devices or links. In fact, the controller can provision another virtual network, thus automatically resolving the reported problem.

The controller may also directly monitor the status of virtual network components such as Network Virtualization Edge elements

(NVEs) [RFC7365] in order to respond to their failures. In addition to forward and backward fault notifications, the controller may deliver notifications to a higher level orchestration component, e.g., one responsible for Virtual Machine (VM) provisioning and management.

Note, given that the IP network on which NVO3 resides is usually self healing, it is expected that recovery by the NVO3 layer would not normally be required, although there may be a requirement for that layer to log that the problem has been detected and resolved. The special cases of a static IP overlay network, or possibly of a centrally controlled IP overlay network, may, however, require NVO3 involvement in fault recovery.

Performance Management deals with mechanism(s) that allow determining and measuring the performance of the network/services under consideration. Performance Management can be used to verify the compliance to both the service-level and network-level metric objectives/specifications. Performance Management typically consists of measuring performance metrics, e.g., Frame Loss, Frame Delay, Frame Delay Variation (aka Jitter), Frame Throughput, Frame Discard, etc., across managed entities when the managed entities are in available state. Performance Management is suspended across unavailable managed entities.

1.2. Requirements Language

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].

1.3. Relationship with Other OAM Work

This document leverages requirements that originate with other OAM work, specifically the following:

- [RFC6136] provides a template and some of the high level requirements and introductory wording.
- [IEEE802.1Q-2011] is expected to provide a subset of the requirements for NVO3 both at the Tenant level and also within the L3 Overlay network.
- [Y.1731] is expected to provide a subset of the requirements for NVO3 at the Tenant level.
- Section 3.3.2.1 of [NVO3-DP-Reqs] lists several requirements specifically concerning ECMP/LAG.

2. 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].

The terminology defined in [RFC7365] and [NVO3-DP-Reqs] is used throughout this document. We introduce no new terminology.

3. NVO3 Reference Model

Figure 1 below reproduces the generic NVO3 reference model as per [RFC7365].

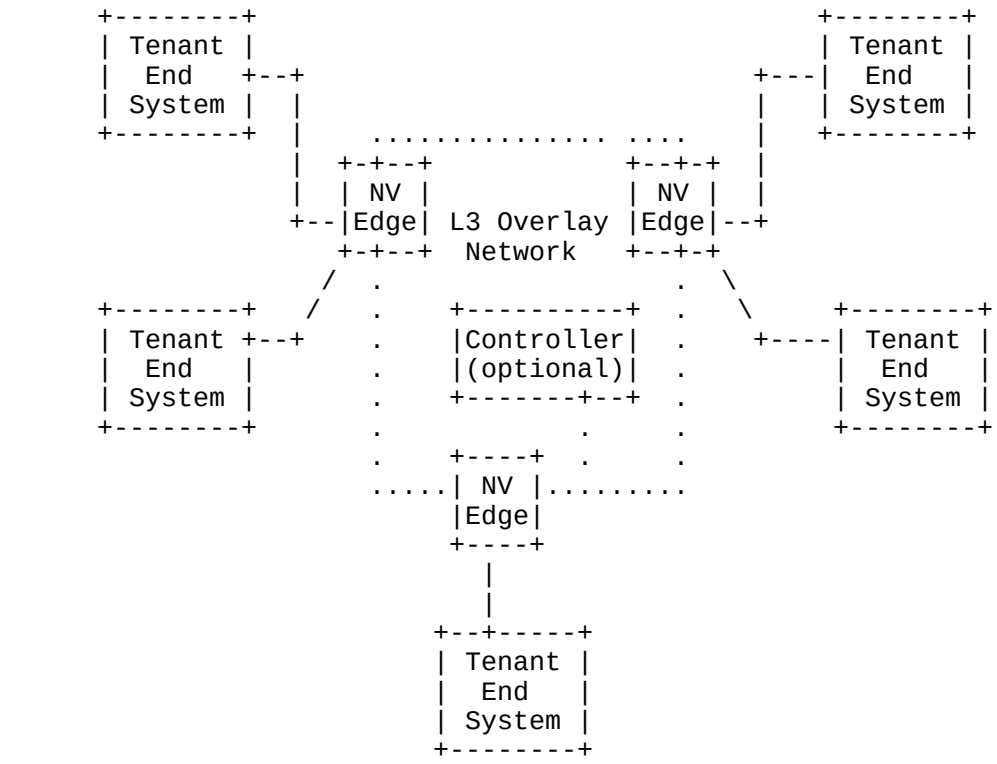


Figure 1: Generic NVO3 Reference Model

Figure 2 below, reproduces the Generic reference model for the NV Edge (NVE) as per [NVO3-DP-Reqs].

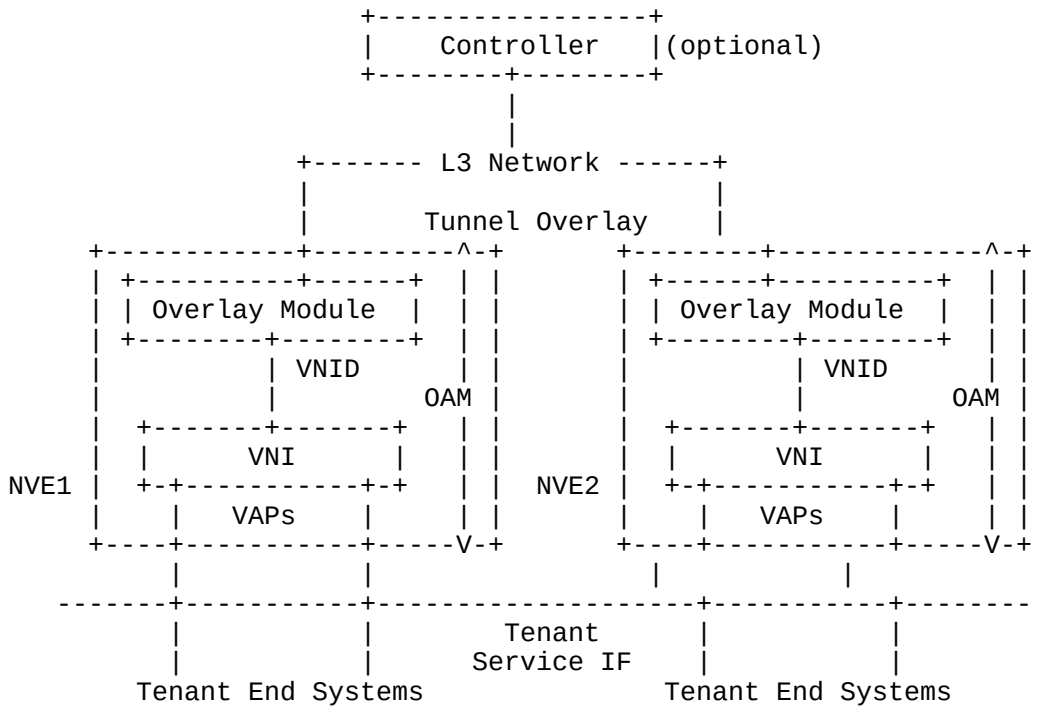


Figure 2: Generic reference model for the NV Edge (NVE)

4. OAM Framework for NVO3

Figure 1 shows the generic reference model for a DC network virtualization over an L3 (or L3VPN) infrastructure while Figure 2 showed the generic reference model for the Network Virtualization (NV) Edge. As shown in both figure 1 and figure 2, the Controller is an optional element that can participate to the support and the operation of OAM functions.

L3 network(s) or L3 VPN networks (either IPv6 or IPv4, or a combination thereof), provide transport for an emulated layer 2 created by NV Edge devices. Unicast and multicast tunneling methods (de-multiplexed by Virtual Network Identifier (VNID)) are used to provide connectivity between the NV Edge devices. The NV Edge devices then present an emulated layer 2 network to the Tenant End Systems at a Virtual Network Interface (VNI) through Virtual Access Points (VAPs). The NV Edge devices map layer 2 unicast to layer 3 unicast point-to-point tunnels and may either map layer 2 multicast to layer 3 multicast tunnels or may replicate packets onto multiple layer 3 unicast tunnels.

4.1. OAM Layering

The emulated layer 2 network is provided by the NV Edge devices to which the Tenant End Systems are connected. This network of NV Edges can be operated by a single service provider or can span across multiple administrative domains. Likewise, the L3 Overlay Network can be operated by a single service provider or span across multiple administrative domains.

While each of the layers is responsible for its own OAM, each layer may consist of several different administrative domains. Figure 3 shows an example.

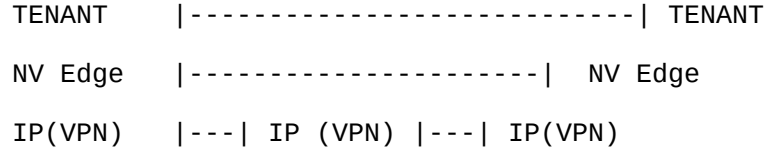


Figure 3: Example NVO3 OAM Layering

For example, at the bottom, at the L3 IP overlay network layer IP(VPN) and/or Ethernet OAM mechanisms are used to probe link by link, node to node etc. OAM addressing here means physical node loopback or interface addresses.

Further up, at the NV Edge layer, NVO3 OAM messages are used to probe the NV Edge to NV Edge tunnels and NV Edge entity status. OAM addressing here likely means the physical node loopback together with the VNI (to de-multiplex the tunnels).

Finally, at the Tenant layer, the IP and/or Ethernet OAM mechanisms are again used but here they are operating over the logical L2/L3 provided by the NV-Edge through the VAP. OAM addressing at this layer deals with the logical interfaces on Vswitches and Virtual Machines.

4.2. OAM Domains

Complex OAM relationships exist as a result of the hierarchical layering of responsibility and of breaking up of end-to-end responsibility.

The OAM domain above NVO3, is expected to be supported by existing IP and L2 OAM methods and tools.

The OAM domain below NVO3, is expected to be supported by existing IP/L2 and MPLS OAM methods and tools. Where this layer is actually

multiple domains spliced together, the existing methods to deal with these boundaries are unchanged. Note however that exposing LAG/ECMP detailed behavior may result in additional requirements to this domain, the details of which will be specified in the future versions of this draft.

When we refer to an OAM domain in this document, or just 'domain', we therefore refer to a closed set of NV Edges or MEPs and the tunnels which interconnect them.

Note, whether for the scenario of inter-domain or multi-layer, each domain (or layer) is responsible for its own OAM, no correlation of OAM function exists between each domain (or layer). When an E2E connection in Tenant layer spans across multiple domains and has multiple underlay layers of NV Edge layer and L3 IP (VPN) layer, current OAM implementation for the E2E connection of Tenant layer such as Fault or Performance Management can only be performed per domain and layer manually and more manual labor is needed. An automatic coordination process among OAM functions of each domain or layer may be useful here for improving efficiency and intelligence.

In the case where a gateway device is use to connect two different domains (whether for changing the encapsulation or other reasons), it is necessary to provide mechanisms to monitor the path through the gateway which involves the removal of one overlay header and the creation of a new one.

5. NVO3 OAM Requirements

5.1. Discovery

R1) NVO3 OAM MUST allow an NV Edge device to dynamically discover other NV Edge devices that share the same VNI within a given NVO3 domain. This may be based on a discovery mechanism used to set up data path forwarding between NVEs.

5.2. Connectivity Fault Management

5.2.1. Connectivity Fault Detection

R2) NVO3 OAM MUST allow proactive connectivity monitoring between two or more NV Edge devices that support the same VNIs within a given NVO3 domain. NVO3 OAM MAY act as a protection trigger. That is, automatic recovery from transmission facility failure by switchover to a redundant replacement facility may be triggered by notifications from NVO3 OAM.

R3) NVO3 OAM MAY allow monitoring/tracing of all possible paths in the underlay network between a specified set of two or more NV Edge devices. Using this feature, equal cost paths that traverse LAG and/or ECMP may be differentiated.

5.2.2. Connectivity Fault Verification

R4) NVO3 OAM MUST allow connectivity fault verification between two or more NV Edge devices that support the same VNI within a given NVO3 domain.

5.2.3. Connectivity Fault localization

R5) NVO3 OAM MUST allow connectivity fault localization between two or more NV Edge devices that support the same VNI within a given NVO3 domain.

5.2.4. Connectivity Fault Notification and Alarm Suppression

R6) NVO3 OAM MUST support fault notification to be triggered as a result of the faults occurring in the underneath network infrastructure. This fault notification SHOULD be used for the suppression of redundant service-level alarms.

5.3. Connectivity Performance Management

5.3.1. Frame Loss

R7) NVO3 OAM MUST support measurement of per VNI frame loss between two NV Edge devices that support the same VNI within a given NVO3 domain.

5.3.2. Frame Delay

R8) NVO3 OAM MUST support measurement of per VNI two-way frame delay between two NV edge devices that support the same VNI within a given NVO3 domain.

R9) NVO3 OAM MUST support measurement of per VNI one-way frame delay between two NV Edge devices that support the same VNI within a given NVO3 domain.

5.3.3. Frame Delay Variation

R10) NVO3 OAM MUST support measurement of per VNI frame delay variation between two NV Edge devices that support the same VNI

within a given NVO3 domain.

5.3.4. Frame Throughput

R11) NVO3 OAM MAY support measurement of per VNI frame throughput (in frames and bytes) between two NV Edge devices that support the same VNI within a given NVO3 domain. This feature could be an effective way to confirm whether or not assigned path bandwidth conforms to service level agreement before providing the path between two NV Edge devices.

5.3.5. Frame Discard

R12) NVO3 OAM MAY support measurement of per VNI frame discard between two NV Edge devices that support the same VNI within a given NVO3 domain. This feature MAY be effective to monitor bursty traffic between two NV Edge devices.

5.4. Continuity Check

NVO3 OAM MUST provide functions that allow any arbitrary NV edge device to perform a Continuity Check to any other NV edge device.

NVO3 OAM MUST provide functions that allow any arbitrary NV edge device to perform a Continuity Check to any other NV edge device using a path associated with a specified flow.

NVO3 OAM SHOULD provide functions that allow any arbitrary NV edge device to perform a Continuity Check to any other NV edge device over any section of any selectable least-cost path.

NVO3 OAM SHOULD provide the ability to perform a Continuity Check on sections of any selectable path within the network.

5.5. Availability

A service may be considered unavailable if the service frames/packets do not reach their intended destination (e.g., connectivity is down) or the service is degraded (e.g., frame loss and/or frame delay and/or delay variation threshold is exceeded). Entry and exit conditions may be defined for the unavailable state. Availability itself may be defined in the context of a service type. Since availability measurement may be associated with connectivity, frame loss, frame delay, and frame delay variation measurements, no additional requirements are specified currently.

5.6. Data Path Forwarding

R13) NVO3 OAM frames MUST be forwarded along the same path (i.e., links (including LAG members) and nodes) as the NVO3 data frames.

R14) NVO3 OAM frames MAY provide a mechanism to exercise/trace all data paths that result due to ECMP/LAG hops in the underlay network, if these paths have been known.

NVO3 OAM frame MUST be possible arranged to follow the path taken by a specific flow.

NVE MUST have the ability to identify frames that require OAM processing.

The Controller element MAY be involved in the out-of-band OAM design and deployment. Indeed, the Controller is expected to maintain an up-to-date global, systemic view of all the network paths and their associated status (e.g., available, idle, unavailable, faulty, in maintenance, etc.)

5.7. Scalability

R15) NVO3 OAM MUST be scalable such that an NV edge device can support proactive OAM for each VNI that is supported by the device.

5.8. Extensibility

R16) NVO3 OAM should be extensible such that new functionality and information elements related to this functionality can be introduced in the future.

R17) NVO3 OAM MUST be defined such that devices not supporting the OAM are able to forward the OAM frames in a similar fashion as the regular NVO3 data frames/packets.

5.9. Security

R18) NVO3 OAM frames MUST be prevented from leaking outside their NVO3 domain.

R19) NVO3 OAM frames from outside an NVO3 domain MUST be prevented from entering the said NVO3 domain when such OAM frames belong to the same level or to a lower-level OAM. (Trivially met because hierarchical domains are independent technologies.)

R20) NVO3 OAM frames from outside an NVO3 domain MUST be transported transparently inside the NVO3 domain when such OAM frames belong to a higher-level NVO3 domain. (Trivially met

because hierarchical domains are independent technologies).

5.10. Transport Independence

Similar to transport requirement from [RFC6136], we expect NVO3 OAM will leverage the OAM capabilities of the transport layer (e.g., IP underlay).

R21) NVO3 OAM MAY allow adaptation/interworking with its IP underlay OAM functions. For example, this would be useful to allow fault notifications from the IP layer to be sent to the NVO3 layer. Likewise, LAG/ECMP-originated notifications may affect the NVO3 OAM decision process.

5.11. Application Independence

R22) NVO3 OAM MAY be independent of the application technologies and specific application OAM capabilities.

5.12. Prioritization

R23) NVO3 OAM messages MUST be preferentially treated in NVE and between NVEs, since NVO3 OAM MAY be used to trigger protection switching. As noted above (R2), protection switching is the automatic replacement of a failed transmission facility with a working one providing equal or greater capacity, typically within a few tens of milliseconds from fault detection.

5.13. Logging and Traceability Requirements

Logging is required at the Network Virtualization Authority (NVA) and the Network Virtualization Edge (NVE) [and the NVO3 Gateway, but the framework does not mention such a beast] in support of fault management and configuration management.

R24) All logs MUST contain a (sufficiently accurate) timestamp to allow the reporting functional instance (i.e., NVA, NVE) to precisely determine the sequence of events. Clocks on different functional instances SHOULD be synchronized to allow similar accuracy when comparing logs from different devices.

R25) All logs MUST contain information that unambiguously identifies the reporting functional instance

R26) Implementations MUST be capable of reporting the following fault-related events:

1. Loss and resumption of connectivity

These reports SHOULD identify the affected VNI(s), but when the loss affects a large number of VNIs simultaneously the report SHOULD identify the underlying entity (e.g., route) if available.

2. Loss and resumption of NVE responsiveness

These reports will be generated by adjacent NVAs or NVEs. They MUST identify the NVE concerned.

3. NVA or NVE change of operational state

These reports will be generated by the NVA or NVE concerned. They MUST indicate the old and new operational states and the cause.

4. Loss and resumption of a VAP

These reports will be generated by adjacent NVAs or NVEs. They MUST identify the VAP concerned.

R27) Implementations MUST be capable of reporting the following events in support of configuration management and auditing. It MUST be possible to generate the reports at both the originating and executing entities. The report generated at the originating entity MUST identify the executing entity and the report at the executing entity MUST identify the originating entity. Both reports MUST indicate the result of the transaction.

1. Virtual Access Point (VAP) creation or deletion

These reports MUST identify the VAP, the Tenant System, and the port supporting the VAP.

2. VNI creation or deletion

These reports MUST identify the VNI and the VAP.

3. VNI renumbering

These reports MUST identify the VAP and the old and new VNI numbers.

4. Reachability and forwarding information update

These reports MUST identify the previous and new file identifiers. (Assumption: reachability and forwarding information is passed as files, which are retained at the originating and executing

entities for a fixed period for auditing purposes.)

R28) As a general requirement, implementations MUST provide a means whereby the operator can impose rate limits on the generation of specific reports. Implementations MUST further permit the operator to totally suppress reporting of specific events. However, if any report types have been suppressed, non-suppressible reports MUST be generated at regular intervals (e.g., once an hour) indicating what report types have been suppressed.

5.14. Live Traffic Monitoring

NVO3 OAM implementations MAY provide methods to utilize live traffic for troubleshooting and performance monitoring.

6. Items for Further Discussion

This section identifies a set of operational items which may be elaborated further if these items fall within the scope of the NVO3.

- VNID renumbering support
  - Means to change the VNID assigned to a given instance MUST be supported.
  - System convergence subsequent to VNID renumbering MUST NOT take longer than a few seconds, to minimize impact on the tenant systems.
  - A NVE MUST be able to map a VNID with a virtual network context.
- VNI migration and management operations
  - Means to delete an existing VNI MUST be supported.
  - Means to add a new VNI MUST be supported.
  - Means to merge several VNIs MAY be supported.
  - Means to retrieve reporting data per VNI MUST be supported.
  - Means to monitor the network resources per VNI MUST be supported.
- Support of planned maintenance operations on the NVO3 infrastructure

    * Graceful procedure to allow for planned maintenance operation on NVE MUST be supported. This includes undoing any configuration changes made for maintenance purposes after completion of the maintenance.

- Support for communication among virtual networks

    * For global reachability purposes, communication among virtual networks MUST be supported. This can be enforced using a NAT function.

- Activation of new network-related services to the NVO3

    * Means to assist in activating new network services (e.g., multicast) without impacting running service SHOULD be supported.

- Inter-operator NVO3 considerations

    * As NVO3 may be deployed over inter-operator infrastructure, coordinating OAM actions in each individual domain are required to ensure an end-to-end OAM. In particular, this assumes existence of agreements on the measurement and monitoring methods, fault detection and repair actions, extending QoS classes (e.g., DSCP mapping policies), etc.

- An automatic coordination process among OAM functions of different domains or layers which an E2E connection in Tenant layer is tunneled on

    * NVO3 OAM MAY support the automatic coordination of OAM functions among different domains or layers which belong to one Tenant layer E2E connection. The automatic coordination means OAM function in client layer or one domain triggers associated OAM functions in server layer or neighbouring domain. This triggered action performs at the domain boundaries, which is also the MEPs of the domain. Which OAM function in client layer or one domain can trigger which OAM functions in server layer or neighbouring domain depends on specific condition, and can be very flexible. But the basic rule is that the OAM functions performed simultaneously in different domains or layers can be synthesized together to get the final result.

    * The OAM MEPs of domains MUST have the capability to know if it they need to perform the above automatic coordination process. This can be achieved by many ways, i.e., by configuration, by checking the flag field in OAM frames.

* When the OAM MEPs perform the automatic coordination, a specific global characteristic information MUST be carried and mapped between OAM frames used in different domains or layers, and be kept the same alone the whole tenant layer E2E connection. The global characteristic information can be the tenant network identifier (e.g., VNID), ICMP sequence number, etc. It is used for identifying a set of correlated OAM results obtained from these domains or layers. This set of OAM results is then synthesized together to get the final diagnose result.

* NVO3 OAM MUST support a Collection Point for collecting all the OAM results and synthesizing them. It can be the SDN controller, NVA, or NMS. An E2E OAM function in tenant network can trigger several OAM functions in different underlay networks, a Collection Point is needed to collect all the OAM results from different OAM MEPs of different domains or layers and synthesizes them.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

Security requirements are specified in Section 5.9. For general NVO3 security considerations, please refer to [NVO3-Security].

9. Acknowledgements

The authors are grateful for the contributions of David Black, Dennis Qin, Erik Smith, Deepark Kumar, Dapeng Liu, and Ziye Yang to this latest version.

10. References

10.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

10.2 Informative References

[IEEE802.1Q-2011] "IEEE Standard for Local and metropolitan area

networks - Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks", 2011.

[NM-Standards] "ITU-T Recommendation M.3400 (02/2000) - TMN Management Functions", February 2000.

[NVO3-DP-Reqs] Bitar, N., Lasserre, M., Balus, F., Morin, T., Jin, L. and Khasnabish, B., "NVO3 Data Plane Requirements", draft-ietf-nvo3-dataplane-requirements-03(work in progress), April 2014.

[NVO3-Security] Hartman, S., Zhang, D., Wasserman, M., Qiang, Z. and Zhang, M., "Security Requirements of NVO3", draft-ietf-nvo3-security-requirements-05(work in progress), June 2015.

[RFC6136] Sajassi, A. and D. Mohan, "Layer 2 Virtual Private Network(L2VPN) Operations, Administration, and Maintenance(OAM) Requirements and Framework", March 2011.

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

[Y.1731] "ITU-T Recommendation Y.1731 (02/08) - OAM functions and mechanisms for Ethernet based networks", February 2008.

Authors' Addresses

Hao Chen
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

Phone: +86-25-56624440
EMail: philips.chenhao@huawei.com

Peter Ashwood-Smith
Huawei Technologies
303 Terry Fox Drive, Suite 400
Kanata, Ontario K2K 3J1
Canada

Phone: +1 613 595-1900
Email: Peter.AshwoodSmith@huawei.com

Liang Xia (Frank)
Huawei Technologies

Email: Frank.xialiang@huawei.com

Ranga Iyengar
Huawei Technologies USA
2330 Central Expy
Santa Clara, CA 95050
USA

Email: ranga.Iyengar@huawei.com

Tina Tsou
Huawei Technologies USA
2330 Central Expy
Santa Clara, CA 95050
USA

Email: Tina.Tsou.Zouting@huawei.com

Ali Sajassi
Cisco Technologies
170 West Tasman Drive
San Jose, CA 95134
USA

Email: sajassi@cisco.com

Mohamed Boucadair
France Telecom
Rennes, 35000
France

Email: mohamed.boucadair@orange.com

Christian Jacquenet
France Telecom
Rennes, 35000
France

Email: christian.jacquenet@orange.com

Masahiro Daikoku
KDDI corporation
3-10-10, Iidabashi, Chiyoda-ku

Tokyo 1028460
Japan

Email: ms-daikoku@kddi.com

Anoop Ghanwani
Dell
5450 Great America Pkwy
Santa Clara, CA
USA

Email: anoop@alumni.duke.edu

Ram Krishnan
Brocade
130 Holger Way
San Jose, CA 95134
USA

Email: ramk@brocade.com

Network Working Group
Internet-Draft
Intended status: Standard Track

L. Yong
Huawei USA
T. Herbert
Facebook

Expires: September 2016

March 14, 2016

Generic UDP Encapsulation (GUE) for Secure Transport
draft-hy-gue-4-secure-transport-03

Abstract

This document specifies the ability of Generic UDP Encapsulation (GUE) to provide secure transport over IP networks and the Internet, including GUE header integrity protection and origin authentication and GUE payload encryption. These are optional features of GUE.

Status of This Document

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/.

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

This Internet-Draft will expire on September 14, 2016.

Table of Contents

1. Introduction

Generic UDP Encapsulation [GUE] is the protocol that specifies tunneling a network protocol over an IP network or Internet and a UDP tunnel. The tunneled network protocol is encapsulated in GUE header [GUE] at a tunnel encapsulator, transported as a regular IP packet, and decapsulated at the tunnel decapsulator.

This draft specifies the security capabilities for GUE. One security capability is to provide origin authentication and integrity protection of the GUE header at tunnel end points to guarantee isolation between tunnels and mitigate Denial of Service attacks. Another capability is payload encryption that prevents the payload from eavesdropping, tampering, or message forgery. These security capabilities are specified as optional features of GUE.

In theory, uses of GUE could leverage other existing tunnel mechanisms that provides secure transport over Internet such as DTLS [RFC6347] and IPsec[RFC4301]. Section 6 discusses the weakness to rely on another tunnel mechanism for GUE secure transport.

2. Terminology

The terms defined in GUE [GUE] are used in this document.

2.1. Requirements Language

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].

3. GUE Security

This document proposes to allocate two flag bits from GUE optional flag field as the Security flag for GUE integrity protection and authentication validation. GUE header format with Security Flag is shown in Figure 1. The second and third bits in GUE optional flag field are allocated for Security mechanism.

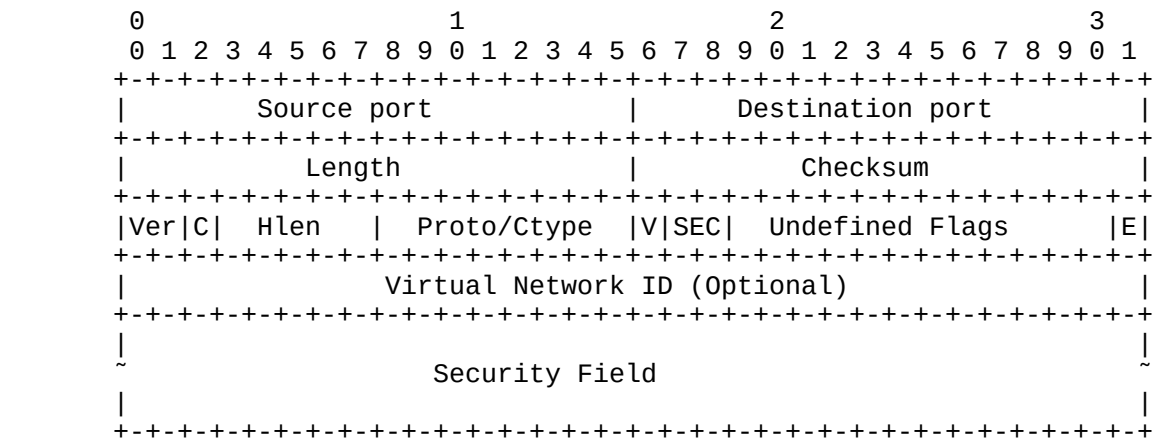


Figure 1 GUE Header Format with Security Flag

- ’V’ Virtualization flag: This flag is designated for network virtualization overlay (NVO)[GUE4NVO].

- ’SEC’ Security flags: Indicates presence of security field. To provide security capability, the flags MUST be set. Different sizes are allowed to allow different methods and extensibility. The use of the security field is expected to be negotiated out of band between two tunnel end points. Potential uses of the security field are discussed in Section of Security Considerations.

    - 00 - No security field

    - 01 - 64 bit security field

    - 10 - 128 bit security field

    - 11 - 256 bit security field

The usage of the key fields in the GUE header [GUE] for the security mechanism is described as below:

- Ver: Set to 0x0. Security option is designed for GUE version 0.

- ’C’ Control flag: When it set, control message presents and control processing MUST occur after security validation.

- Hlen: length of optional fields (byte)/4. Note that Payload Transform function does not require a private field.

o Proto/ctype: Contain the protocol of the encapsulated payload packet, i.e. next header when the C bit is not set. Contains a control message type when the C bit is set. The next header begins at the offset provided by Hlen.

'E' Extension flag. It is the last bit in the GUE header. For usage of Extension field see [GUE]. The security transport does not require use of any extension field.

UDP header field must be set per [GUE]. The checksum and length implementation MUST be compliant with GUE implementation [GUE].

4. GUE Payload Encryption

The payload of a GUE packet can be secured using Datagram Transport Layer Security [RFC6347]. An encapsulator would apply DTLS to the GUE payload so that the payload packets are encrypted and the GUE header remains in plaintext.

To differ encrypted payload from plaintext payload, the document proposes allocating one flag from GUE optional flag field for payload transformation indication and adding a 32 bit field when Payload Transform flag is set. Following format shows GUE header with Payload Transform flag, i.e. 'T' is set.

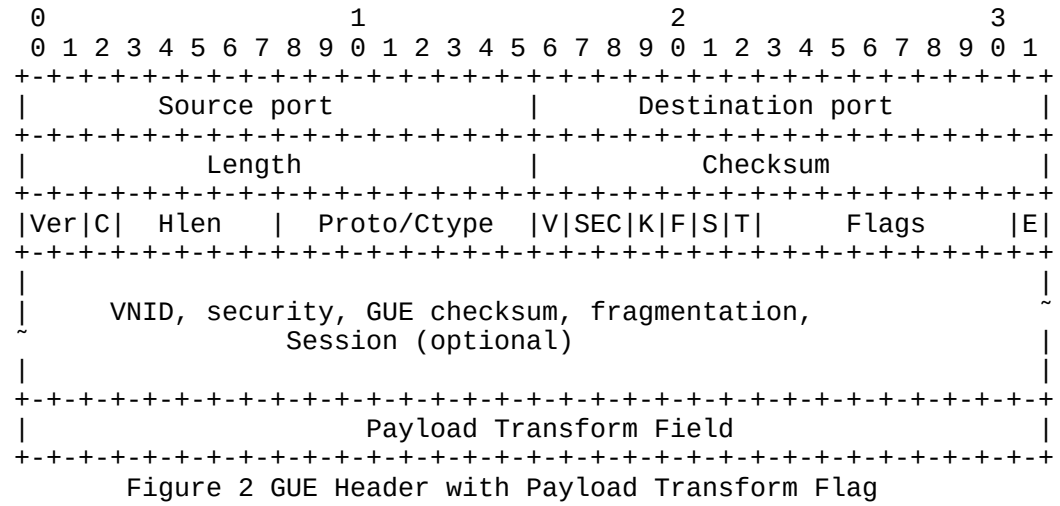


Figure 2 GUE Header with Payload Transform Flag

A 32 bits field in GUE header is for the payload transform function and MUST be presented when Payload Transform flag T is set and MUST

NOT be presented when clear. The format of Payload Transform Field is in Figure 3.

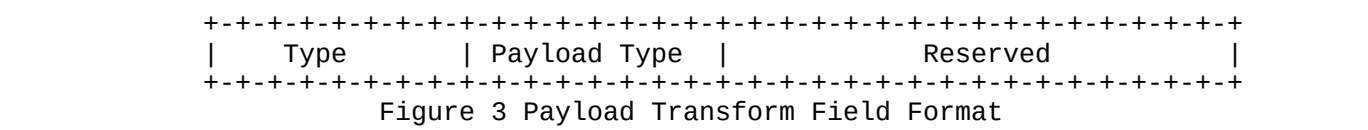


Figure 3 Payload Transform Field Format

Type: Payload Transform Type or Code point. Each payload transform mechanism must have one code point registered in IANA. This document specifies:

0x01: for DTLS [RFC6347]

0x80~0xFF: for private payload transform types

A private payload transform type can be used for experimental purpose or vendor proprietary mechanism.

Payload Type: used to indicate the encrypted payload type. When encryption flag is set, next protocol in the base header should set to 59 ("No next header") for a data message and zero for a control message. The payload type (IP protocol or control message type) of the unencrypted paytload must be encoded in this field.

The benefit of this rule is to prevent a middle box from inspecting the encrypted payload according to GUE next protocol. Assumption here is that a middle box may understand GUE base header but does not understand GUE option flag definitions.

Reserved field for DTLS type MUST set to Zero. For other transformation type, the reserved field may be specified for a purpose.

The usage of the key fields in the GUE header [GUE] for the payload encryption mechanism is described as below:

- Ver: Set to 0x0. Payload transform option applies to version 0x0.
- Control flag: When it set, control message presents SHOULD be encrypted except GUE base header.
- Hlen: length of optional fields (byte)/4
- Proto/CType: Set to 59. The payload type is encoded in the payload encryption option field.

- o 'E' Extension flag. It is the last bit in the GUE header. For usage of Extension field see [GUE]. The payload encryption does not require use of any extension field.

UDP header usage for payload encryption mechanism: UDP dst port SHOULD be filled with GUE port [GUE]; UDP src port MAY be filled with entropy or a random value. The checksum and length implementation MUST be compliant with GUE implementation [GUE].

5. Encapsulation/Decapsulation Operations

GUE secure transport mechanism applies to both IPv4 and IPv6 underlay networks. The outer IP address MUST be tunnel egress IP address (dst) and tunnel ingress IP address (src). GUE security option provides origin authentication and integrity to GUE based tunnel; GUE payload encryption provides payload privacy over an IP delivery network or Internet. Two functions are processed separately at tunnel end points. A GUE tunnel can use both functions or use one of them.

When both encryption and security are required, an encapsulator must perform payload encryption first and then encapsulate the encrypted packet with security flag and encryption flag set in GUE header; the security field must be filled according Section 3 above; the encryption field must be filled according to Section 4 above; the decapsulator must decapsulate the packet first, then perform the authentication validation; if the validation is successful, it performs the payload decryption according the encryption information in the payload encryption field in the header; if the validation fails, the decapsulator must discard the packet and may generate an alert to the management system. These processing rules also apply when only one function, either encryption or security, is enabled, except another function is not performed as stated above.

If GUE fragmentation is used in concert with the GUE security option and/or payload transform option, the security and playload transformation are performed after fragmentation at the encapsulator and before reassembly at the decapsulator.

In order to get flow entropy from the payload, the encapsulator needs to get the flow entropy first before performing the payload encryption; then the flow entropy is inserted into UDP src port in the GUE header.

DTLS [RFC6347] provides packet fragmentation capability. To avoid packet fragmentation performed multiple times at GUE encapsulator, GUE encapsulator SHOULD only perform the packet fragmentation at

packet encapsulation process, i.e., not in payload encryption process. The encryption process should apply to GUE control packets.

DTLS usage [RFC6347] is limited to a single DTLS session for any specific tunnel encapsulator/decapsulator pair (identified by source and destination IP addresses). Both IP addresses MUST be unicast addresses - multicast traffic is not supported when DTLS is used. A GUE tunnel decapsulator implementation that supports DTLS can establish DTLS session(s) with one or multiple tunnel encapsulators, and likewise a GUE tunnel encapsulator implementation can establish DTLS session(s) with one or multiple decapsulators.

6. Considerations of Using Other Security Tunnel Mechanisms

GUE may rely on other secure tunnel mechanisms such as DTLS [RFC6347] for securing the whole GUE packet or IPsec [RFC4301] to achieve the secure transport over an IP network or Internet.

IPsec [RFC4301] was designed as a network security mechanism, and therefore it resides at the network layer. As such, if the tunnel is secured with IPsec, the UDP header would not be visible to intermediate routers anymore in either IPsec tunnel or transport mode. The big drawback here prohibits intermediate routers to perform load balance based on the flow entropy in UDP header. In addition, this method prohibits any middle box function on the path.

By comparison, DTLS [RFC6347] was designed with application security and can better preserve network and transport layer protocol information than IPsec [RFC4301]. Using DTLS to secure the GUE tunnel, both GUE header and payload will be encrypted. In order to differentiate plaintext GUE header from encrypted GUE header, the destination port of the UDP header between two must be different, which essentially requires another standard UDP port for GUE with DTLS. The drawback on this method is to prevent a middle box operation to GUE tunnel on the path.

Use of two independent tunnel mechanisms such as GUE and DTLS to carry a network protocol over an IP network adds some overlap and process complex. For example, fragmentation will be done twice.

As the result, a GUE tunnel SHOULD use the security mechanisms specified in this document to provide secure transport over an IP network or Internet when it is needed. GUE tunnel can be used as secure transport mechanism over an IP network and Internet.

7. Security Considerations

Encapsulation of network protocol in GUE should not increase security risk, nor provide additional security in itself. GUE requires that the source port for UDP packets should be randomly seeded to mitigate some possible denial service attacks.

If the integrity and privacy of data packets being transported through GUE is a concern, GUE security and payload encryption SHOULD be used to remove the concern. If the integrity is the only concern, the tunnel may consider use of GUE security only for optimization. Likewise, if the privacy is the only concern, the tunnel may use GUE encryption function only.

If GUE payload already provides secure mechanism, e.g., the payload is IPsec packets, it is still valuable to consider use of GUE secure mechanisms for the payload header privacy and the tunnel integrity.

7.1. GUE Security Field Usage

The GUE security field should be used to provide integrity and authentication of the GUE header. Security negotiation (interpretation of security field, key management, etc.) is expected to be negotiated out of band between two communicating hosts. Two possible uses for this field are outlined below, a more precise specification is deferred to other documents.

7.1.1. Cookies

The security field may be used as a cookie. This would be similar to cookie mechanism described in L2TP [RFC3931], and the general properties should be the same. The cookie may be used to validate the encapsulation. The cookie is a shared value between an encapsulator and decapsulator which should be chosen randomly and may be changed periodically. Different cookies may used for logical flows between the encapsulator and decapsulator, for instance packets sent with different VNIs in network virtualization [GUE4NVO] might have different cookies.

7.1.2. Secure hash

Strong authentication of the GUE header can be provided using a secure hash. This may follow the model of the TCP authentication option [RFC5925]. In this case the security field holds a message digest for the GUE header (e.g. 16 bytes from MD5). The digest might be done over static fields in IP and UDP headers per negotiation (addresses, ports, and protocols). In order to provide enough

entropy, a random salt value in each packet might be added, for instance the security field could be a 256 bit value which contains 128 bits of salt value, and a 128 bit digest value. The use of secure hashes requires shared keys which are presumably negotiated and rotated as needed out of band.

8. IANA Considerations

This document requires IANA to allocate:

Two bits in GUE option flag field for GUE Security.
One bit in GUE option flag field for GUE Payload Transform.

This document requires IANA to have a new registry for Encryption Type and require two code points for:

0x01: for DTLS [RFC6347]

0x80~0xFF: for private payload transform

9. References

9.1. Normative References

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

[RFC3931] Lau, J., Townsley, W., et al, "Layer Tow Tunneling Protocol - Version 3 (L2TPv3)", RFC3931, 1999

[RFC4301] Kent, S., Seo, K., "Security Architecture for the Internet Protocol", RFC4301, December 2005

[RFC5925] Touch, J., et al, "The TCP Authentication Option", RFC5925, June 2010

[RFC6347] Rescoria, E., Modadugu, N., "Datagram Transport Layer Security Version 1.2", RFC6347, 2012.

[GUE] Herbert, T., Yong, L., et al "Generic UNP Encapsulation", draft-ietf-nvo3-gue-00, work in progress.

9.2. Informative References

[GUE4NVO] Yong, L., and Herbert T., "Generic UNP Encapsulation for NVO", draft-hy-nvo3-gue-4-nvo-01, work in progress.

10. Authors' Addresses

Lucy Yong
Huawei USA

Email: lucy.yong@huawei.com

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA 94052
US

Email: tom@herbertland.com

Network Working Group
Internet-Draft
Intended status: Standard Track

L. Yong
Huawei USA
T. Herbert
Facebook
O. Zia
Microsoft

Expires: April 2017 October 28, 2016

Generic UDP Encapsulation (GUE) for Network Virtualization Overlay
draft-hy-nvo3-gue-4-nvo-04

Abstract

This document describes network virtualization overlay encapsulation scheme by use of Generic UDP Encapsulation (GUE) [GUE]. It allocates one GUE optional flag and defines a 32 bit field for Virtual Network Identifier (VNID).

Status of This Document

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/.

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

This Internet-Draft will expire on April 28, 2017.

Table of Contents

1. Introduction

Network Virtualization Overlay (NVO3) [RFC7365] provides a framework for a virtual network solution over an IP network in a DC with multi-tenant environment. Virtual network packets, i.e. tenant packets, between any pair of Network Virtualization Edges (NVE) are encapsulated at ingress NVE, sent from ingress NVE to egress NVE as IP packets, and decapsulated at egress NVE. This is known as a tunnel mechanism. This draft specifies use of Generic UDP Encapsulation (GUE) [GUE] for NVO3 packet encapsulation.

GUE [GUE] as a generic UDP encapsulation provides several merits for NVO3 encapsulation. Hence, underlay IP network treats it the same was as other UDP applications, that are well supported by both IPv4 and IPv6 underlay networks. GUE provides strong security transport options [GUEEXT] that NVO3 can leverage. In addition, GUE supports other options that NVO3 may use such as private data and extensibility. In addition, GUE control flag can be used for NVO3 OAM message.

This document requests one flag (1 bit) from GUE optional flag field for Network Virtualization Overlay (NVO3) indication and specifies a 32 bit field for virtual network identifier in GUE optional fields. It describes use of GUE security options in NVO3.

2. Terminology

The terms defined in [GUE], [RFC7365] are used in this document.

2.1. Requirements Language

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].

3. Generic UDP Encapsulation (GUE) for NVO3

Generic UDP Encapsulation adds a 32 bit basic GUE header after UDP header. GUE header contains some key fields that a UDP tunnel application needs. These key fields are version, control message indication (c), Header Length (HLen), and Protocol Type (or ctype). It also contains some optional flags that are reserved for optional features at a UDP tunnel.

This document proposes to allocate one flag bit from GUE optional flags for the Network Virtualization Overlay (NVO3) and defines a 32 bit field for NVO3 in GUE optional fields when the flag bit is set. GUE based NVO3 encapsulation format is shown in Figure 1.

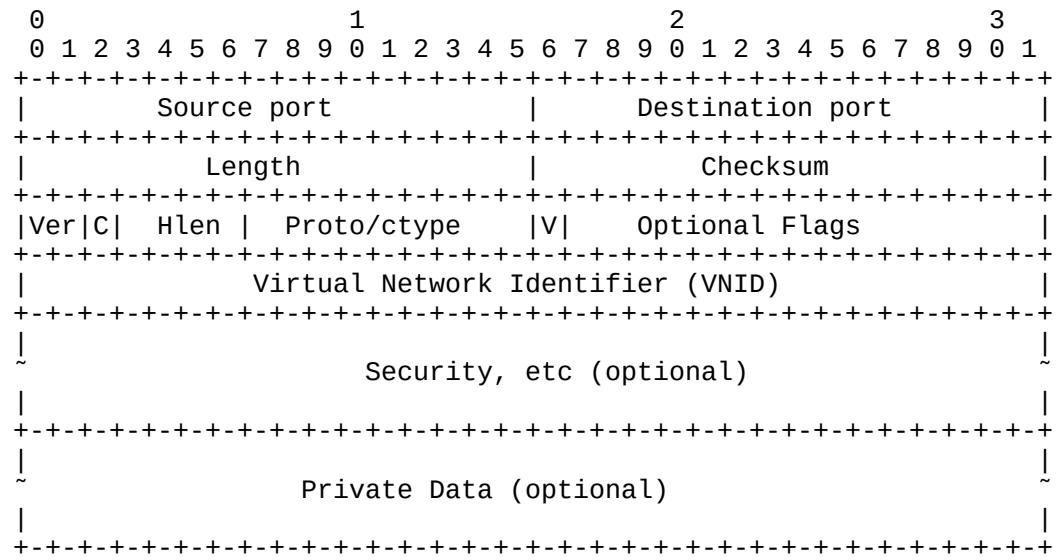


Figure 1 GUE based NVO3 Encapsulation Format

- 'V': Virtualization flag. Indicates presence of the Virtual Network Identifier (VNID) field in GUE optional fields. This flag MUST be set when GUE is used for network virtualization overlay (NVO3).

- Virtual Network ID (4 octets): a 32 bit field is used to identify a virtual network that the packet belongs to. This field MUST be present when 'V' virtualization flag is set; and MUST NOT present when 'V' flag is clear.

NVO3 implementation may carry private data in the private data field. It must follow the rules specified in [GUE] when inserting private data in GUE header.

NVO3 may allocate other flags and fields in GUE header for NVO3 purpose and MUST follow the flag/field allocation rules specified in [GUE].

The usage of the key fields in the GUE header [GUE] for NVO3 encapsulation is described as below:

- Ver: Set to 0x0 to indicate version zero of GUE. Packets received by the decapsulator with non-zero version MUST be dropped.

- Control flag: When set, indicates that the packet contains a control message. OAM packets for the virtual network instance can be carried as a control message. NOV3 OAM packet format and mechanisms will be specified in a separated document.

- Hlen: length of (optional fields + private field (byte))/4.

- Proto/ctype: Contain the protocol of the encapsulated payload packet, i.e. next header or control message type (ctype) when Control flag is set. The next header begins at the offset provided by Hlen. For network virtualization, the payload protocol can be Ethernet, IPv4, IPv6, or 59 (NULL). The VNID can be used with ctype to direct control message for the VN layer.

UDP header field MUST be set per [GUE]. The checksum and length implementation MUST be compliant with GUE implementation [GUE].

NVO3 can use GUE specified optional functions to improve the transport such as GUE security option [GUEEXT], GUE checksum option [GUEEXT], etc. When using a GUE specified option, NVO3 implementation MUST be compliant with the corresponding specification.

4. Encapsulation/Decapsulation Operations

The network virtualization encapsulation by use of GUE applies to both IPv4 and IPv6 underlay networks. The outer source and destination IP addresses MUST be ingress NVE and egress NVE IP addresses respectively. Ingress NVE adds UDP and GUE headers on the payload packet with the required fields as described in Section 3. NVE encapsulation and decapsulation process MUST be compliant with GUE implementation specification [GUE]. If ingress and egress NVE implement GUE options, they MUST be compliant with the corresponding GUE option specification.

4.1. Multi-Tenant Segregation

Ingress NVE MUST set option ’V’ and insert Virtual Network Identifier (VNID) into the corresponding option field when encapsulating tenant packets. A GUE tunnel can carry the payload packets that are from different tenant networks simultaneously.

Egress NVE MUST use the VNID in GUE header to identify the tenant network that the payload packet is associated to and forward to the packet to corresponding tenant network. All 32 bits can be used for VNID.

4.2. Tenant Broadcast and Multicast Packets

If tenant packet is L2 broadcast/multicast, or L3 multicast packet, depending on which multicast solution NV03 deploys [NVO3MFRWK], the packet may be carried by a set of point-to-point GUE tunnels, or a point-to-multipoint GUE tunnel. In the latter case, multicast IP address is used as outer destination address.

The mapping of inner broadcast/multicast group to IP multicast group can be manually configured or based on an algorithm, which is outside the scope of this document.

4.3. Fragmentation

To gain the performance and simplification, NVO3 SHOULD avoid packet fragmentation. Manual configuration or negotiation with tenant systems can ensure that the MTU of the physical networks is greater than or equal to the MTU of the encapsulated network plus GUE header. It is strongly RECOMMENDED Path MTU Discovery [RFC1191] [RFC1981] to be used by setting the DF bit in the IP header when GUE packets are carried by IPv4 (this is the default with IPv6). In a case, it can’t avoid packet fragmentation; GUE fragmentation option can be used [GUEEXT].

4.4. GUE Header Security

NVO3 is expected to operate in multi-tenant environment, so security is extreme important. Security can be provided by DC networking and/or by NVO3. NVO3 can use GUE security options [GUEEXT]. When NVO3 use GUE security option, ingress NVE has to set the security flag and insert a key value in the security field [GUEEXT], egress NVE has to validate the key prior to packet decapitation process. If the key validation fails, the packet will be dropped [GUEEXT]. The key value used between ingress and egress NVE can be managed by NVA or generated algorithm at NVEs. This mechanism will be described in future version.

4.5. Tenant Packet Encryption

To prevent tenant packet from eavesdropping, tampering, or message forgery, NVO3 can adopt GUE payload encryption mechanism. To encrypt tenant packets, ingress NVE sets GUE payload transform flag and adds

32 bit payload transform field in GUE header. The payload type MUST be filled at the payload transform field and the protocol field in GUE base header MUST be set to 59 "No next header"[GUEEXT]. Both ingress NVE and egress NVE MUST implement the encryption mechanism as described in [GUEEXT].

5. IANA Considerations

The document request IANA to allocate the first bit in the registry of GUE optional flag fields for Network Virtualization Flag and register 32 bit field on GUE option field registry for Network Virtualization Identifier (VNID).

6. Security Considerations

When Network Virtualization Edge (NVE) uses the UDP tunnel mechanism specified in GUE [GUE], it faces the same security concern stated in Section of Security Considerations in [GUE] and can leverage GUE secure transport mechanisms [GUEEXT] for secure transport over the underlay IP network.

GUE provides two optional security functions. One is origin authentication and integrity protection between encapsulator and decapsulator, which protects Denial of Service (DoS) attacks; another is GUE payload encryption, which prevents the payload from eavesdropping, tampering, or message forgery. The two functions can be used together or independently according to the deployment environment. NVO3 virtual network identifier (VNID) is encoded in GUE header that can be protected by origin authentication.

7. References

7.1. Normative References

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

[RFC7365] Lasserre, M., el al, "Framework for Data Center (DC) Network Virtualization".

[GUE] Herbert T., Yong, L., Zia, O., "Generic UNP Encapsulation", draft-ietf-nvo3-gue, work in progress.

[GUEEXT] Herbert, T., Yong, L., Templin, F., "Extensions for Generic UDP Encapsulation", draft-herbert-gue-extensions, work in progress.

7.2. Informative Reference

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[NVO3MFRWK] Ghanwani, A., Dunbar, L., et al "A Framework for Multicast in Network Virtualization Overlays", draft-ietf-nov3-mcast-framework, work in progress.

8. Authors' Addresses

Lucy Yong
Huawei USA
5340 Legacy Dr.
Plano, TX 75024
US

Email: lucy.yong@huawei.com

Tom Herbert
Google
1600 Amphitheatre Parkway
Mountain View, CA
US

Email: therbert@google.com

Osama Zia
Microsoft
1 Microsoft Way
Redmond, WA 98029
US

Email: osamaz@microsoft.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 10, 2019

J. Gross, Ed.
I. Ganga, Ed.
Intel
T. Sridhar, Ed.
VMware

October 07, 2018

Geneve: Generic Network Virtualization Encapsulation
draft-ietf-nvo3-geneve-08

Abstract

Network virtualization involves the cooperation of devices with a wide variety of capabilities such as software and hardware tunnel endpoints, transit fabrics, and centralized control clusters. As a result of their role in tying together different elements in the system, the requirements on tunnels are influenced by all of these components. Flexibility is therefore the most important aspect of a tunnel protocol if it is to keep pace with the evolution of the system. This draft describes Geneve, a protocol designed to recognize and accommodate these changing capabilities and needs.

Table of Contents

1. Introduction

Networking has long featured a variety of tunneling, tagging, and other encapsulation mechanisms. However, the advent of network virtualization has caused a surge of renewed interest and a corresponding increase in the introduction of new protocols. The large number of protocols in this space, ranging all the way from VLANs [IEEE.802.1Q_2014] and MPLS [RFC3031] through the more recent VXLAN [RFC7348], NVGRE [RFC7637], often leads to questions about the need for new encapsulation formats and what it is about network virtualization in particular that leads to their proliferation.

While many encapsulation protocols seek to simply partition the underlay network or bridge between two domains, network virtualization views the transit network as providing connectivity between multiple components of a distributed system. In many ways this system is similar to a chassis switch with the IP underlay network playing the role of the backplane and tunnel endpoints on the edge as line cards. When viewed in this light, the requirements placed on the tunnel protocol are significantly different in terms of the quantity of metadata necessary and the role of transit nodes.

Current work such as VL2 [VL2] and the NVO3 working group [I-D.ietf-nvo3-dataplane-requirements] have described some of the properties that the data plane must have to support network virtualization. However, one additional defining requirement is the need to carry system state along with the packet data. The use of some metadata is certainly not a foreign concept - nearly all protocols used for virtualization have at least 24 bits of identifier space as a way to partition between tenants. This is often described as overcoming the limits of 12-bit VLANs, and when seen in that context, or any context where it is a true tenant identifier, 16 million possible entries is a large number. However, the reality is that the metadata is not exclusively used to identify tenants and encoding other information quickly starts to crowd the space. In fact, when compared to the tags used to exchange metadata between line cards on a chassis switch, 24-bit identifiers start to look quite small. There are nearly endless uses for this metadata, ranging from storing input ports for simple security policies to service based context for interposing advanced middleboxes.

Existing tunnel protocols have each attempted to solve different aspects of these new requirements, only to be quickly rendered out of date by changing control plane implementations and advancements. Furthermore, software and hardware components and controllers all

have different advantages and rates of evolution - a fact that should be viewed as a benefit, not a liability or limitation. This draft describes Geneve, a protocol which seeks to avoid these problems by providing a framework for tunneling for network virtualization rather than being prescriptive about the entire system.

1.1. Requirements Language

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].

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. Terminology

The NVO3 framework [RFC7365] defines many of the concepts commonly used in network virtualization. In addition, the following terms are specifically meaningful in this document:

Checksum offload. An optimization implemented by many NICs which enables computation and verification of upper layer protocol checksums in hardware on transmit and receive, respectively. This typically includes IP and TCP/UDP checksums which would otherwise be computed by the protocol stack in software.

Clos network. A technique for composing network fabrics larger than a single switch while maintaining non-blocking bandwidth across connection points. ECMP is used to divide traffic across the multiple links and switches that constitute the fabric. Sometimes termed "leaf and spine" or "fat tree" topologies.

ECMP. Equal Cost Multipath. A routing mechanism for selecting from among multiple best next hop paths by hashing packet headers in order to better utilize network bandwidth while avoiding reordering a single stream.

Geneve. Generic Network Virtualization Encapsulation. The tunnel protocol described in this draft.

LRO. Large Receive Offload. The receive-side equivalent function of LSO, in which multiple protocol segments (primarily TCP) are coalesced into larger data units.

NIC. Network Interface Card. A NIC could be part of a tunnel endpoint or transit device and can either process Geneve packets or aid in the processing of Geneve packets.

OAM. Operations, Administration, and Management. A suite of tools used to monitor and troubleshoot network problems.

Transit device. A forwarding element along the path of the tunnel making up part of the Underlay Network. A transit device MAY be capable of understanding the Geneve packet format but does not originate or terminate Geneve packets.

LSO. Large Segmentation Offload. A function provided by many commercial NICs that allows data units larger than the MTU to be passed to the NIC to improve performance, the NIC being responsible for creating smaller segments of size less than or equal to the MTU with correct protocol headers. When referring specifically to TCP/IP, this feature is often known as TSO (TCP Segmentation Offload).

Tunnel endpoint. A component performing encapsulation and decapsulation of packets, such as Ethernet frames or IP datagrams, in Geneve headers. As the ultimate consumer of any tunnel metadata, endpoints have the highest level of requirements for parsing and interpreting tunnel headers. Tunnel endpoints may consist of either software or hardware implementations or a combination of the two. Endpoints are frequently a component of an NVE but may also be found in middleboxes or other elements making up an NVO3 Network.

VM. Virtual Machine.

2. Design Requirements

Geneve is designed to support network virtualization use cases, where tunnels are typically established to act as a backplane between the virtual switches residing in hypervisors, physical switches, or middleboxes or other appliances. An arbitrary IP network can be used as an underlay although Clos networks composed using ECMP links are a common choice to provide consistent bisectional bandwidth across all connection points. Figure 1 shows an example of a hypervisor, top of rack switch for connectivity to physical servers, and a WAN uplink connected using Geneve tunnels over a simplified Clos network. These tunnels are used to encapsulate and forward frames from the attached components such as VMs or physical links.

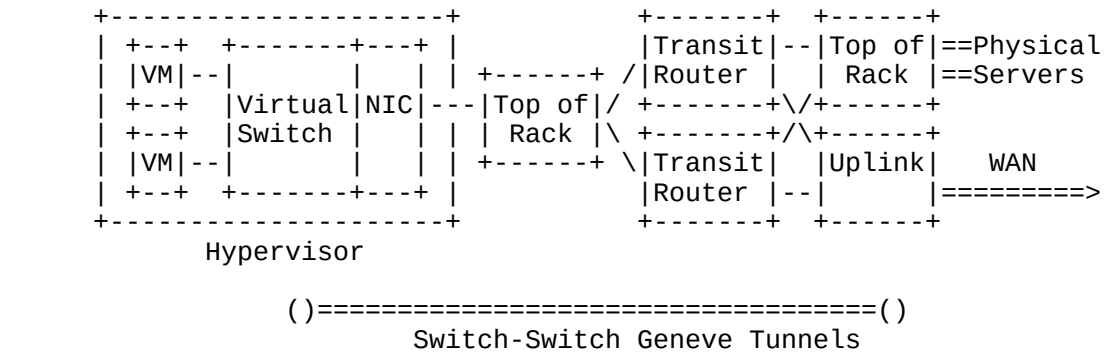


Figure 1: Sample Geneve Deployment

To support the needs of network virtualization, the tunnel protocol should be able to take advantage of the differing (and evolving) capabilities of each type of device in both the underlay and overlay networks. This results in the following requirements being placed on the data plane tunneling protocol:

- o The data plane is generic and extensible enough to support current and future control planes.
- o Tunnel components are efficiently implementable in both hardware and software without restricting capabilities to the lowest common denominator.
- o High performance over existing IP fabrics.

These requirements are described further in the following subsections.

2.1. Control Plane Independence

Although some protocols for network virtualization have included a control plane as part of the tunnel format specification (most notably, the original VXLAN spec prescribed a multicast learning-based control plane), these specifications have largely been treated as describing only the data format. The VXLAN packet format has actually seen a wide variety of control planes built on top of it.

There is a clear advantage in settling on a data format: most of the protocols are only superficially different and there is little advantage in duplicating effort. However, the same cannot be said of control planes, which are diverse in very fundamental ways. The case for standardization is also less clear given the wide variety in requirements, goals, and deployment scenarios.

As a result of this reality, Geneve aims to be a pure tunnel format specification that is capable of fulfilling the needs of many control planes by explicitly not selecting any one of them. This simultaneously promotes a shared data format and increases the chances that it will not be obsoleted by future control plane enhancements.

2.2. Data Plane Extensibility

Achieving the level of flexibility needed to support current and future control planes effectively requires an options infrastructure to allow new metadata types to be defined, deployed, and either finalized or retired. Options also allow for differentiation of products by encouraging independent development in each vendor's core specialty, leading to an overall faster pace of advancement. By far the most common mechanism for implementing options is Type-Length-Value (TLV) format.

It should be noted that while options can be used to support non-wirespeed control packets, they are equally important on data packets as well to segregate and direct forwarding (for instance, the examples given before of input port based security policies and service interposition both require tags to be placed on data packets). Therefore, while it would be desirable to limit the extensibility to only control packets for the purposes of simplifying the datapath, that would not satisfy the design requirements.

2.2.1. Efficient Implementation

There is often a conflict between software flexibility and hardware performance that is difficult to resolve. For a given set of functionality, it is obviously desirable to maximize performance. However, that does not mean new features that cannot be run at that speed today should be disallowed. Therefore, for a protocol to be efficiently implementable means that a set of common capabilities can be reasonably handled across platforms along with a graceful mechanism to handle more advanced features in the appropriate situations.

The use of a variable length header and options in a protocol often raises questions about whether it is truly efficiently implementable in hardware. To answer this question in the context of Geneve, it is important to first divide "hardware" into two categories: tunnel endpoints and transit devices.

Endpoints must be able to parse the variable header, including any options, and take action. Since these devices are actively participating in the protocol, they are the most affected by Geneve.

However, as endpoints are the ultimate consumers of the data, transmitters can tailor their output to the capabilities of the recipient. As new functionality becomes sufficiently well defined to add to endpoints, supporting options can be designed using ordering restrictions and other techniques to ease parsing.

Transit devices MAY be able to interpret the options, however, as non-terminating devices, transit devices do not originate or terminate the Geneve packet, hence MUST NOT insert or delete options, which is the responsibility of Geneve endpoints. The participation of transit devices in interpreting options is OPTIONAL.

Further, either tunnel endpoints or transit devices MAY use offload capabilities of NICs such as checksum offload to improve the performance of Geneve packet processing. The presence of a Geneve variable length header SHOULD NOT prevent the tunnel endpoints and transit devices from using such offload capabilities.

2.3. Use of Standard IP Fabrics

IP has clearly cemented its place as the dominant transport mechanism and many techniques have evolved over time to make it robust, efficient, and inexpensive. As a result, it is natural to use IP fabrics as a transit network for Geneve. Fortunately, the use of IP encapsulation and addressing is enough to achieve the primary goal of delivering packets to the correct point in the network through standard switching and routing.

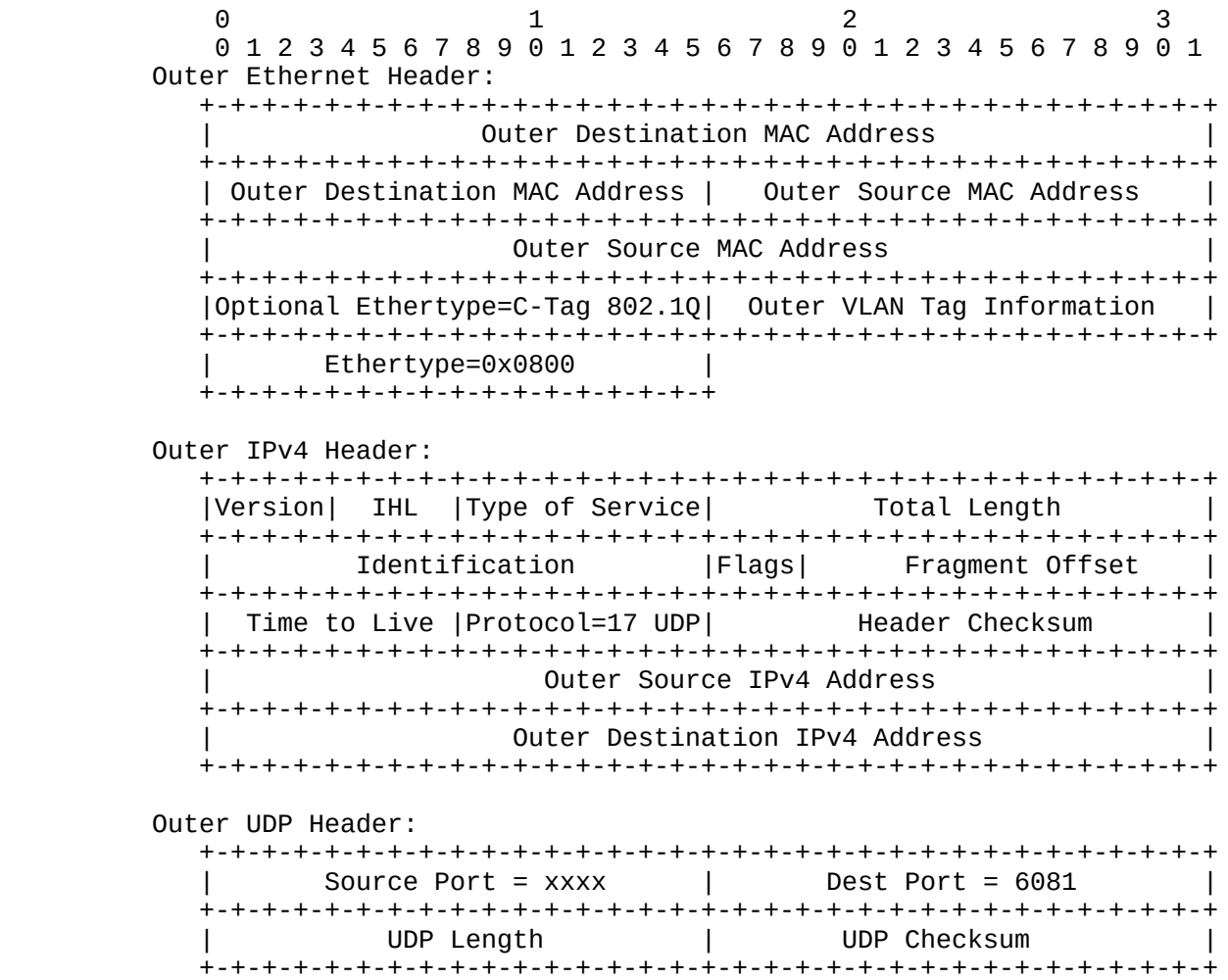
In addition, nearly all underlay fabrics are designed to exploit parallelism in traffic to spread load across multiple links without introducing reordering in individual flows. These equal cost multipathing (ECMP) techniques typically involve parsing and hashing the addresses and port numbers from the packet to select an outgoing link. However, the use of tunnels often results in poor ECMP performance without additional knowledge of the protocol as the encapsulated traffic is hidden from the fabric by design and only endpoint addresses are available for hashing.

Since it is desirable for Geneve to perform well on these existing fabrics, it is necessary for entropy from encapsulated packets to be exposed in the tunnel header. The most common technique for this is to use the UDP source port, which is discussed further in Section 3.3.

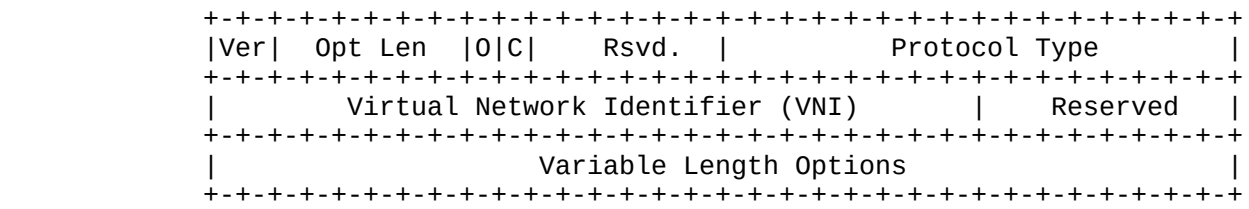
3. Geneve Encapsulation Details

The Geneve packet format consists of a compact tunnel header encapsulated in UDP over either IPv4 or IPv6. A small fixed tunnel header provides control information plus a base level of functionality and interoperability with a focus on simplicity. This header is then followed by a set of variable options to allow for future innovation. Finally, the payload consists of a protocol data unit of the indicated type, such as an Ethernet frame. Section 3.1 and Section 3.2 illustrate the Geneve packet format transported (for example) over Ethernet along with an Ethernet payload.

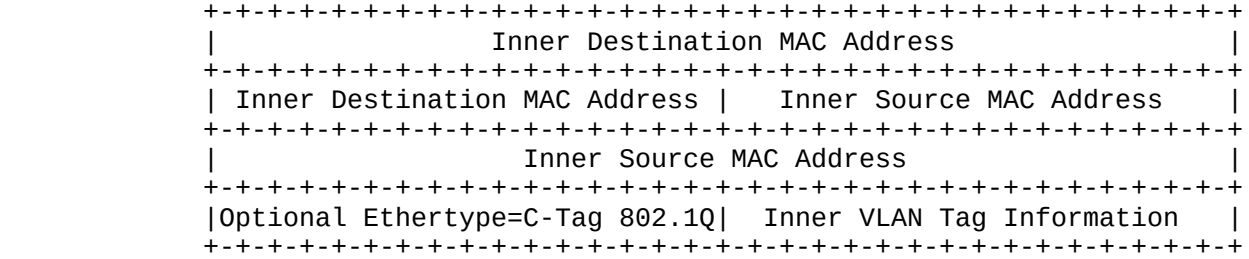
3.1. Geneve Packet Format Over IPv4



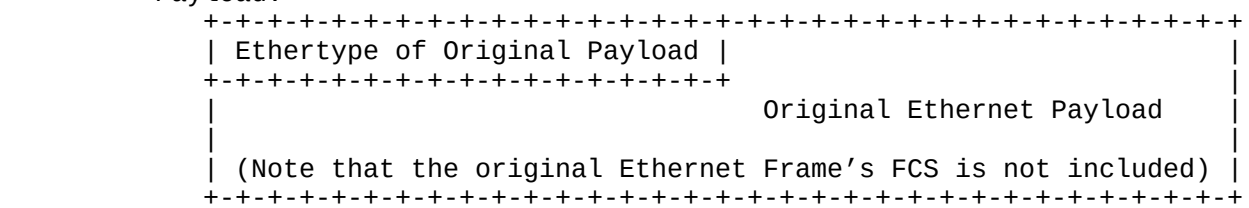
Geneve Header:



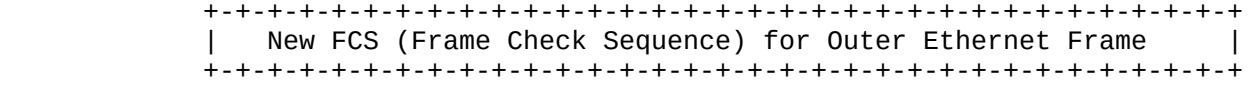
Inner Ethernet Header (example payload):



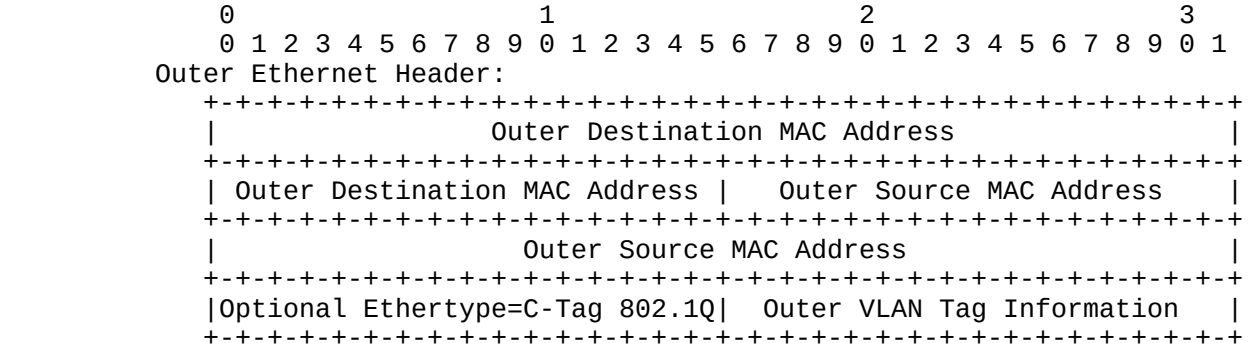
Payload:

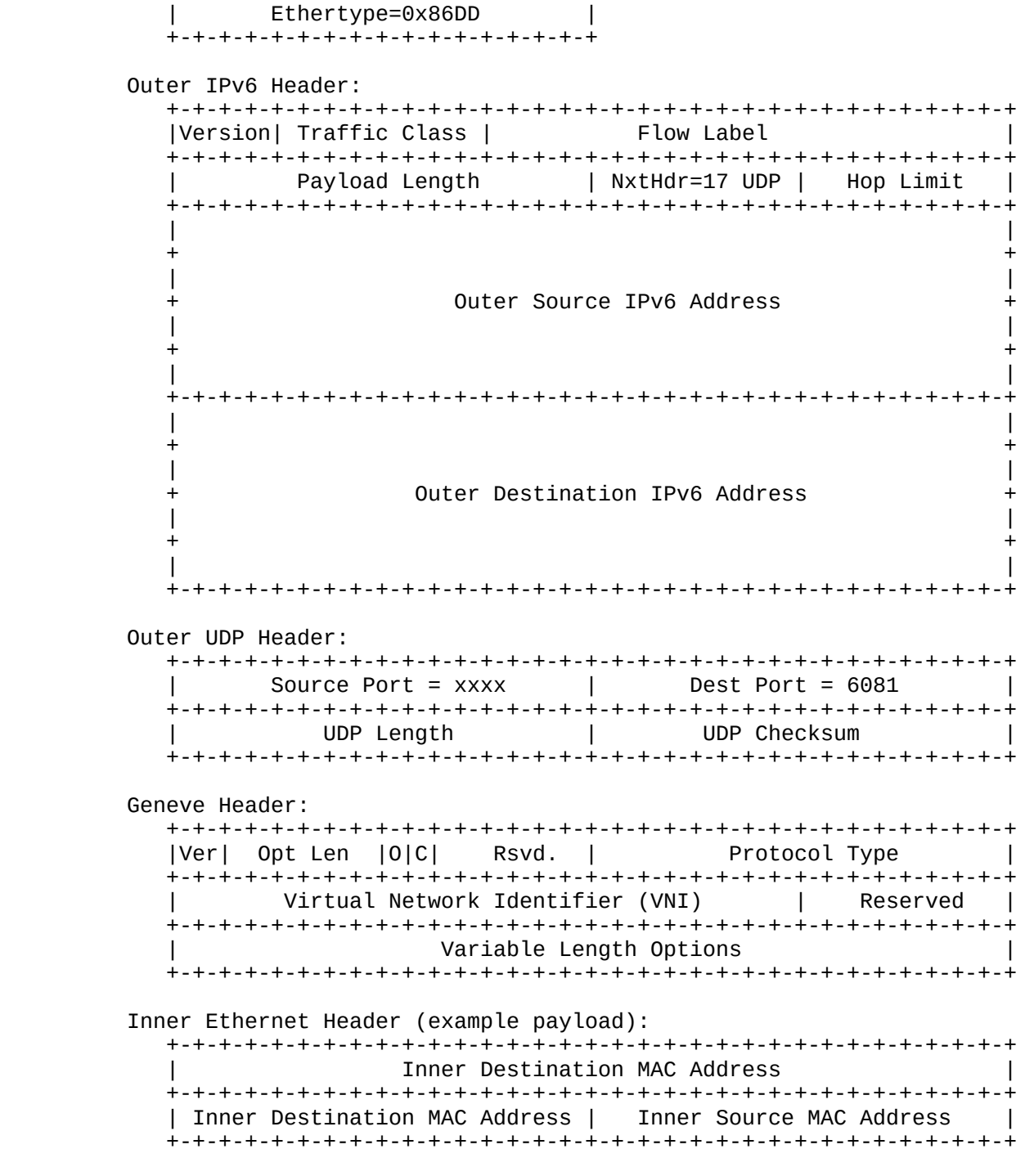


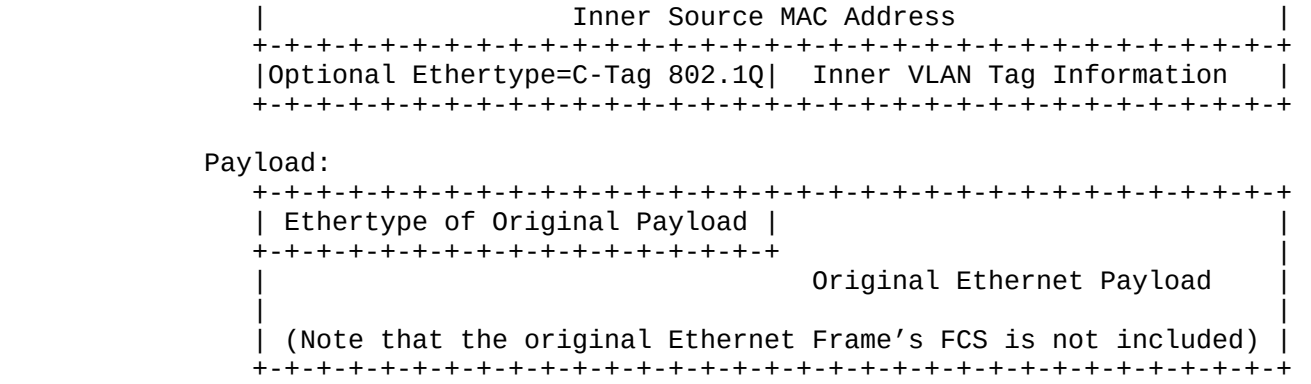
Frame Check Sequence:



3.2. Geneve Packet Format Over IPv6




Ethertype=0x86DD
Outer IPv6 Header:
Version
Traffic Class
Flow Label
Payload Length
NxtHdr=17 UDP
Hop Limit
Outer Source IPv6 Address
Outer Destination IPv6 Address
Outer UDP Header:
Source Port = xxxx
Dest Port = 6081
UDP Length
UDP Checksum
Geneve Header:
Ver
Opt Len
O
C
Rsvd.
Protocol Type
Virtual Network Identifier (VNI)
Reserved
Variable Length Options
Inner Ethernet Header (example payload):
Inner Destination MAC Address
Inner Destination MAC Address
Inner Source MAC Address



3.3. UDP Header

The use of an encapsulating UDP [RFC0768] header follows the connectionless semantics of Ethernet and IP in addition to providing entropy to routers performing ECMP. The header fields are therefore interpreted as follows:

Source port: A source port selected by the originating tunnel endpoint. This source port SHOULD be the same for all packets belonging to a single encapsulated flow to prevent reordering due to the use of different paths. To encourage an even distribution of flows across multiple links, the source port SHOULD be calculated using a hash of the encapsulated packet headers using, for example, a traditional 5-tuple. Since the port represents a flow identifier rather than a true UDP connection, the entire 16-bit range MAY be used to maximize entropy.

Dest port: IANA has assigned port 6081 as the fixed well-known destination port for Geneve. Although the well-known value should be used by default, it is RECOMMENDED that implementations make this configurable. The chosen port is used for identification of Geneve packets and MUST NOT be reversed for different ends of a connection as is done with TCP.

UDP length: The length of the UDP packet including the UDP header.

UDP checksum: The checksum MAY be set to zero on transmit for

packets encapsulated in both IPv4 and IPv6 [RFC6935]. When a packet is received with a UDP checksum of zero it MUST be accepted and decapsulated. If the originating tunnel endpoint optionally encapsulates a packet with a non-zero checksum, it MUST be a correctly computed UDP checksum. Upon receiving such a packet, the egress endpoint MUST validate the checksum. If the checksum is not correct, the packet MUST be dropped, otherwise the packet MUST be accepted for decapsulation. It is RECOMMENDED that the UDP checksum be computed to protect the Geneve header and options in situations where the network reliability is not high and the packet is not protected by another checksum or CRC.

3.4. Tunnel Header Fields

Ver (2 bits): The current version number is 0. Packets received by an endpoint with an unknown version MUST be dropped. Non-terminating devices processing Geneve packets with an unknown version number MUST treat them as UDP packets with an unknown payload.

Opt Len (6 bits): The length of the options fields, expressed in four byte multiples, not including the eight byte fixed tunnel header. This results in a minimum total Geneve header size of 8 bytes and a maximum of 260 bytes. The start of the payload headers can be found using this offset from the end of the base Geneve header.

O (1 bit): OAM packet. This packet contains a control message instead of a data payload. Control messages are sent between Geneve endpoints. Endpoints MUST NOT forward the payload and transit devices MUST NOT attempt to interpret or process it. Since these are infrequent control messages, it is RECOMMENDED that endpoints direct these packets to a high priority control queue (for example, to direct the packet to a general purpose CPU from a forwarding ASIC or to separate out control traffic on a NIC). Transit devices MUST NOT alter forwarding behavior on the basis of this bit, such as ECMP link selection.

C (1 bit): Critical options present. One or more options has the critical bit set (see Section 3.5). If this bit is set then tunnel endpoints MUST parse the options list to interpret any critical options. On endpoints where option parsing is not supported the packet MUST be dropped on the basis of the 'C' bit in the base header. If the bit is not set tunnel endpoints MAY strip all options using 'Opt Len' and forward the decapsulated packet. Transit devices MUST NOT drop packets on the basis of this bit.

The critical bit allows hardware implementations the flexibility to handle options processing in the hardware fastpath or in the exception (slow) path without the need to process all the options. For example, a critical option such as secure hash to provide Geneve header integrity check must be processed by tunnel endpoints and typically processed in the hardware fastpath.

Rsvd. (6 bits): Reserved field which MUST be zero on transmission and ignored on receipt.

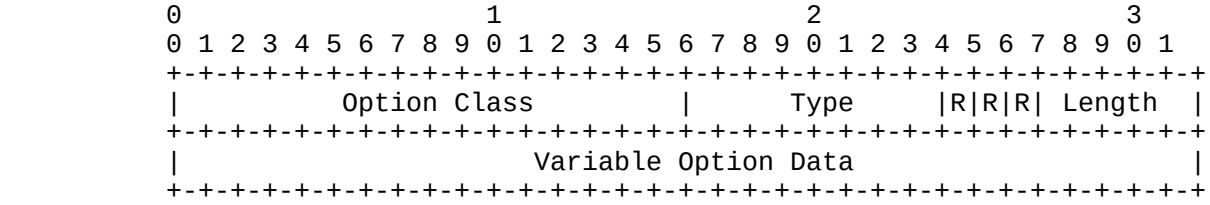
Protocol Type (16 bits): The type of the protocol data unit appearing after the Geneve header. This follows the EtherType [ETYPES] convention with Ethernet itself being represented by the value 0x6558.

Virtual Network Identifier (VNI) (24 bits): An identifier for a unique element of a virtual network. In many situations this may represent an L2 segment, however, the control plane defines the forwarding semantics of decapsulated packets. The VNI MAY be used as part of ECMP forwarding decisions or MAY be used as a mechanism to distinguish between overlapping address spaces contained in the encapsulated packet when load balancing across CPUs.

Reserved (8 bits): Reserved field which MUST be zero on transmission and ignored on receipt.

Transit devices MUST maintain consistent forwarding behavior irrespective of the value of 'Opt Len', including ECMP link selection. These devices SHOULD be able to forward packets containing options without resorting to a slow path.

3.5. Tunnel Options



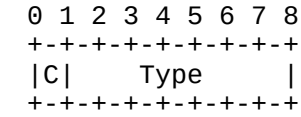
Geneve Option

The base Geneve header is followed by zero or more options in Type-Length-Value format. Each option consists of a four byte option header and a variable amount of option data interpreted according to the type.

Option Class (16 bits): Namespace for the 'Type' field. IANA will be requested to create a "Geneve Option Class" registry to allocate identifiers for organizations, technologies, and vendors that have an interest in creating types for options. Each organization may allocate types independently to allow experimentation and rapid innovation. It is expected that over time certain options will become well known and a given implementation may use option types from a variety of sources. In addition, IANA will be requested to reserve specific ranges for standardized and experimental options.

Type (8 bits): Type indicating the format of the data contained in this option. Options are primarily designed to encourage future extensibility and innovation and so standardized forms of these options will be defined in a separate document.

The high order bit of the option type indicates that this is a critical option. If the receiving endpoint does not recognize this option and this bit is set then the packet MUST be dropped. If the critical bit is set in any option then the 'C' bit in the Geneve base header MUST also be set. Transit devices MUST NOT drop packets on the basis of this bit. The following figure shows the location of the 'C' bit in the 'Type' field:



The requirement to drop a packet with an unknown critical option applies to the entire tunnel endpoint system and not a particular component of the implementation. For example, in a system comprised of a forwarding ASIC and a general purpose CPU, this does not mean that the packet must be dropped in the ASIC. An implementation may send the packet to the CPU using a rate-limited control channel for slow-path exception handling.

R (3 bits): Option control flags reserved for future use. MUST be zero on transmission and ignored on receipt.

Length (5 bits): Length of the option, expressed in four byte multiples excluding the option header. The total length of each option may be between 4 and 128 bytes. A value of 0 in the Length field implies an option with only the option header without the variable option data. Packets in which the total length of all options is not equal to the 'Opt Len' in the base header are invalid and MUST be silently dropped if received by an endpoint.

Variable Option Data: Option data interpreted according to 'Type'.

3.5.1. Options Processing

Geneve options are intended to be originated and processed by tunnel endpoints. However, options MAY be interpreted by transit devices along the tunnel path. Transit devices not processing Geneve headers SHOULD process Geneve packets as any other UDP packet and maintain consistent forwarding behavior.

In tunnel endpoints, the generation and interpretation of options is determined by the control plane, which is out of the scope of this document. However, to ensure interoperability between heterogeneous devices some requirements are imposed on options and the devices that process them:

- Receiving endpoints MUST drop packets containing unknown options with the 'C' bit set in the option type. Conversely, transit devices MUST NOT drop packets as a result of encountering unknown options, including those with the 'C' bit set.

- Some options may be defined in such a way that the position in the option list is significant. Options or their ordering, MUST NOT be changed by transit devices.

- An option MUST NOT affect the parsing or interpretation of any other option.

When designing a Geneve option, it is important to consider how the option will evolve in the future. Once an option is defined it is reasonable to expect that implementations may come to depend on a specific behavior. As a result, the scope of any future changes must be carefully described upfront.

Unexpectedly significant interoperability issues may result from changing the length of an option that was defined to be a certain size. A particular option is specified to have either a fixed length, which is constant, or a variable length, which may change over time or for different use cases. This property is part of the definition of the option and conveyed by the 'Type'. For fixed length options, some implementations may choose to ignore the length field in the option header and instead parse based on the well known length associated with the type. In this case, redefining the length will impact not only parsing of the option in question but also any options that follow. Therefore, options that are defined to be fixed length in size MUST NOT be redefined to a different length. Instead, a new 'Type' should be allocated.

4. Implementation and Deployment Considerations

4.1. Encapsulation of Geneve in IP

As an IP-based tunnel protocol, Geneve shares many properties and techniques with existing protocols. The application of some of these are described in further detail, although in general most concepts applicable to the IP layer or to IP tunnels generally also function in the context of Geneve.

4.1.1. IP Fragmentation

To prevent fragmentation and maximize performance, the best practice when using Geneve is to ensure that the MTU of the physical network is greater than or equal to the MTU of the encapsulated network plus tunnel headers. Manual or upper layer (such as TCP MSS clamping) configuration can be used to ensure that fragmentation never takes place, however, in some situations this may not be feasible.

It is strongly RECOMMENDED that Path MTU Discovery ([RFC1191], [RFC1981]) be used by setting the DF bit in the IP header when Geneve packets are transmitted over IPv4 (this is the default with IPv6). The use of Path MTU Discovery on the transit network provides the encapsulating endpoint with soft-state about the link that it may use to prevent or minimize fragmentation depending on its role in the virtualized network. For example, recommendations/guidance for handling fragmenation in similar overlay encapsulation services like PWE3 are provided in section 5.3 of [RFC3985].

Note that some implementations may not be capable of supporting fragmentation or other less common features of the IP header, such as options and extension headers.

4.1.2. DSCP and ECN

When encapsulating IP (including over Ethernet) packets in Geneve, there are several considerations for propagating DSCP and ECN bits from the inner header to the tunnel on transmission and the reverse on reception.

[RFC2983] provides guidance for mapping DSCP between inner and outer IP headers. Network virtualization is typically more closely aligned with the Pipe model described, where the DSCP value on the tunnel header is set based on a policy (which may be a fixed value, one based on the inner traffic class, or some other mechanism for grouping traffic). Aspects of the Uniform model (which treats the inner and outer DSCP value as a single field by copying on ingress and egress) may also apply, such as the ability to remark the inner

header on tunnel egress based on transit marking. However, the Uniform model is not conceptually consistent with network virtualization, which seeks to provide strong isolation between encapsulated traffic and the physical network.

[RFC6040] describes the mechanism for exposing ECN capabilities on IP tunnels and propagating congestion markers to the inner packets. This behavior MUST be followed for IP packets encapsulated in Geneve.

4.1.3. Broadcast and Multicast

Geneve tunnels may either be point-to-point unicast between two endpoints or may utilize broadcast or multicast addressing. It is not required that inner and outer addressing match in this respect. For example, in physical networks that do not support multicast, encapsulated multicast traffic may be replicated into multiple unicast tunnels or forwarded by policy to a unicast location (possibly to be replicated there).

With physical networks that do support multicast it may be desirable to use this capability to take advantage of hardware replication for encapsulated packets. In this case, multicast addresses may be allocated in the physical network corresponding to tenants, encapsulated multicast groups, or some other factor. The allocation of these groups is a component of the control plane and therefore outside of the scope of this document. When physical multicast is in use, the 'C' bit in the Geneve header may be used with groups of devices with heterogeneous capabilities as each device can interpret only the options that are significant to it if they are not critical.

4.1.4. Unidirectional Tunnels

Generally speaking, a Geneve tunnel is a unidirectional concept. IP is not a connection oriented protocol and it is possible for two endpoints to communicate with each other using different paths or to have one side not transmit anything at all. As Geneve is an IP-based protocol, the tunnel layer inherits these same characteristics.

It is possible for a tunnel to encapsulate a protocol, such as TCP, which is connection oriented and maintains session state at that layer. In addition, implementations MAY model Geneve tunnels as connected, bidirectional links, such as to provide the abstraction of a virtual port. In both of these cases, bidirectionality of the tunnel is handled at a higher layer and does not affect the operation of Geneve itself.

4.2. Constraints on Protocol Features

Geneve is intended to be flexible to a wide range of current and future applications. As a result, certain constraints may be placed on the use of metadata or other aspects of the protocol in order to optimize for a particular use case. For example, some applications may limit the types of options which are supported or enforce a maximum number or length of options. Other applications may only handle certain encapsulated payload types, such as Ethernet or IP. This could be either globally throughout the system or, for example, restricted to certain classes of devices or network paths.

These constraints may be communicated to tunnel endpoints either explicitly through a control plane or implicitly by the nature of the application. As Geneve is defined as a data plane protocol that is control plane agnostic, the exact mechanism is not defined in this document.

4.2.1. Constraints on Options

While Geneve options are more flexible, a control plane may restrict the number of option TLVs as well as the order and size of the TLVs, between tunnel endpoints, to make it simpler for a data plane implementation in software or hardware to handle [I-D.ietf-nvo3-encap]. For example, there may be some critical information such as a secure hash that must be processed in a certain order to provide lowest latency.

A control plane may negotiate a subset of option TLVs and certain TLV ordering, as well may limit the total number of option TLVs present in the packet, for example, to accommodate hardware capable of processing fewer options [I-D.ietf-nvo3-encap]. Hence, a control plane needs to have the ability to describe the supported TLVs subset and their order to the tunnel end points. In the absence of a control plane, alternative configuration mechanisms may be used for this purpose. The exact mechanism is not defined in this document.

4.3. NIC Offloads

Modern NICs currently provide a variety of offloads to enable the efficient processing of packets. The implementation of many of these offloads requires only that the encapsulated packet be easily parsed (for example, checksum offload). However, optimizations such as LSO and LRO involve some processing of the options themselves since they must be replicated/merged across multiple packets. In these situations, it is desirable to not require changes to the offload logic to handle the introduction of new options. To enable this,

some constraints are placed on the definitions of options to allow for simple processing rules:

- When performing LSO, a NIC MUST replicate the entire Geneve header and all options, including those unknown to the device, onto each resulting segment. However, a given option definition may override this rule and specify different behavior in supporting devices. Conversely, when performing LRO, a NIC MAY assume that a binary comparison of the options (including unknown options) is sufficient to ensure equality and MAY merge packets with equal Geneve headers.

- Options MUST NOT be reordered during the course of offload processing, including when merging packets for the purpose of LRO.

- NICs performing offloads MUST NOT drop packets with unknown options, including those marked as critical.

There is no requirement that a given implementation of Geneve employ the offloads listed as examples above. However, as these offloads are currently widely deployed in commercially available NICs, the rules described here are intended to enable efficient handling of current and future options across a variety of devices.

4.4. Inner VLAN Handling

Geneve is capable of encapsulating a wide range of protocols and therefore a given implementation is likely to support only a small subset of the possibilities. However, as Ethernet is expected to be widely deployed, it is useful to describe the behavior of VLANs inside encapsulated Ethernet frames.

As with any protocol, support for inner VLAN headers is OPTIONAL. In many cases, the use of encapsulated VLANs may be disallowed due to security or implementation considerations. However, in other cases trunking of VLAN frames across a Geneve tunnel can prove useful. As a result, the processing of inner VLAN tags upon ingress or egress from a tunnel endpoint is based upon the configuration of the endpoint and/or control plane and not explicitly defined as part of the data format.

5. Interoperability Issues

Viewed exclusively from the data plane, Geneve does not introduce any interoperability issues as it appears to most devices as UDP packets. However, as there are already a number of tunnel protocols deployed in network virtualization environments, there is a practical question of transition and coexistence.

Since Geneve is a superset of the functionality of the most common protocols used for network virtualization (VXLAN, NVGRE) it should be straightforward to port an existing control plane to run on top of it with minimal effort. With both the old and new packet formats supporting the same set of capabilities, there is no need for a hard transition - endpoints directly communicating with each other use any common protocol, which may be different even within a single overall system. As transit devices are primarily forwarding packets on the basis of the IP header, all protocols appear similar and these devices do not introduce additional interoperability concerns.

To assist with this transition, it is strongly suggested that implementations support simultaneous operation of both Geneve and existing tunnel protocols as it is expected to be common for a single node to communicate with a mixture of other nodes. Eventually, older protocols may be phased out as they are no longer in use.

6. Security Considerations

As encapsulated within an UDP/IP packet, Geneve does not have any inherent security mechanisms. As a result, an attacker with access to the underlay network transporting the IP packets has the ability to snoop or inject packets. Legitimate but malicious tunnel endpoints may also spoof identifiers in the tunnel header to gain access to networks owned by other tenants.

Within a particular security domain, such as a data center operated by a single service provider, the most common and highest performing security mechanism is isolation of trusted components. Tunnel traffic can be carried over a separate VLAN and filtered at any untrusted boundaries. In addition, tunnel endpoints should only be operated in environments controlled by the service provider, such as the hypervisor itself rather than within a customer VM.

When crossing an untrusted link, such as the public Internet, IPsec [RFC4301] may be used to provide authentication and/or encryption of the IP packets formed as part of Geneve encapsulation.

Geneve does not otherwise affect the security of the encapsulated packets. As per the guidelines of BCP72 [RFC3552], the following sections describe potential security risks that may be applicable to Geneve deployments and approaches to mitigate such risks. It is also noted that not all such risks are applicable to all Geneve deployment scenarios, i.e., only a subset may be applicable to certain deployments. So an operator has to make an assessment based on their network environment and determine the risks that are applicable to their specific environment and use appropriate mitigation approaches as applicable.

6.1. Data Confidentiality

Geneve is a network virtualization overlay encapsulation protocol designed to establish tunnels between network virtualization end points (NVE) over an existing IP network. It can be used to deploy multi-tenant overlay networks over an existing IP underlay network in a public or private data center. The overlay service is typically provided by a service provider, for example a cloud services provider or a private data center operator. Due to the nature of multi-tenancy in such environments, a tenant system may expect data confidentiality to ensure its packet data is not tampered with (active attack) in transit or a target of unauthorized monitoring (passive attack). A tenant may expect the overlay service provider to provide data confidentiality as part of the service or a tenant may bring its own data confidentiality mechanisms like IPsec or TLS to protect the data end to end between its tenant systems.

If an operator determines data confidentiality is necessary in their environment based on their risk analysis, for example as in multi-tenant environments, then an encryption mechanism SHOULD be used to encrypt the tenant data end to end between the NVEs. The NVEs may use existing well established encryption mechanisms such as IPsec, DTLs, etc., The operator may choose not to enable the encryption if, for example, the packet data is already encrypted by the tenant system.

6.1.1. Inter-data center traffic

A tenant system in a customer premises (private data center) may want to connect to tenant systems on their tenant overlay network in a public cloud data center or a tenant may want to have its tenant systems located in multiple geographically separated data centers for high availability. Geneve data traffic between tenant systems across such separated networks should be protected from threats when traversing public networks. Any Geneve overlay data leaving the data center network beyond the operator’s security domain, for example over the public Internet, SHOULD be secured by encryption mechanisms such as IPsec or other VPN mechanisms to protect the communications between the NVEs when they are geographically separated over untrusted network links. Implementation of specific data protection mechanisms employed between data centers is beyond the scope of this document.

6.2. Data Integrity

Geneve encapsulation is used between NVEs to establish overlay tunnels over an existing IP underlay network. In a multi-tenant data center, a rogue or compromised tenant system may try to launch a

passive attack such as monitoring the traffic of other tenants, or an active attack such as spoofing or trying to inject unauthorized Geneve encapsulated traffic into the network. To prevent such attacks, an NVE MUST not propagate Geneve packets beyond the NVE to tenant systems and SHOULD employ packet filtering mechanisms so as not to forward unauthorized traffic between TSs in different tenant networks.

A compromised network node or a transit device within a data center may launch an active attack trying to tamper with the Geneve packet data between NVEs. Malicious tampering of Geneve header fields may cause the packet from one tenant to be forwarded to a different tenant network. If an operator determines the possibility of such threat in their environment, the operator may choose to employ data integrity mechanisms between NVEs. In order to prevent such risks, a data integrity mechanism SHOULD be used in such environments to protect the integrity of Geneve packets including packet headers, options and payload on communications between NVE pairs. A cryptographic data protection mechanism such as IPsec may be used to provide data integrity protection. A data center operator may choose to deploy any other data integrity mechanisms as applicable and supported in their underlay networks.

Geneve supports Geneve Options, so an operator may choose to use a Geneve option TLV to provide a cryptographic data protection mechanism, to verify the data integrity of the Geneve header, Geneve options or the entire Geneve packet including the payload. Implementation of such a mechanism is beyond the scope of this document.

6.3. Authentication of NVE peers

A rogue network device or a compromised NVE in a data center environment might be able to spoof Geneve packets as if it came from a legitimate NVE. In order to mitigate such a risk, an operator SHOULD use an Authentication mechanism, such as IPsec to ensure that the Geneve packet originated from the intended NVE peer, in environments where the operator determines spoofing or rogue devices is a potential threat. Other simpler source checks such as ingress filtering for VLAN/MAC/IP address, reverse path forwarding checks, etc., may be used in certain trusted environments to ensure Geneve packets originated from the intended NVE peer.

6.4. Multicast/Broadcast

In typical data center networks where IP multicasting is not supported in the underlay network, multicasting may be supported using multiple unicast tunnels. The same security requirements as

described in the above sections can be used to protect Geneve communications between NVE peers. If IP multicasting is supported in the underlay network and the operator chooses to use it for multicast traffic among Geneve endpoints, then the operator in such environments may use data protection mechanisms such as IPsec with Multicast extensions [RFC5374] to protect multicast traffic among Geneve NVE groups.

6.5. Control plane communications

A Network Virtualization Authority (NVA) as outlined in [RFC8014] may be used as a control plane for configuring and managing the Geneve NVEs. The data center operator is expected to use security mechanisms to protect the communications between the NVA to NVEs and use authentication mechanisms to detect any rogue or compromised NVEs within their administrative domain. Data protection mechanisms for control plane communication or authentication mechanisms between the NVA and the NVEs is beyond the scope of this document.

7. IANA Considerations

IANA has allocated UDP port 6081 as the well-known destination port for Geneve. Upon publication, the registry should be updated to cite this document. The original request was:

Service Name: geneve
Transport Protocol(s): UDP
Assignee: Jesse Gross <jgross@vmware.com>
Contact: Jesse Gross <jgross@vmware.com>
Description: Generic Network Virtualization Encapsulation (Geneve)
Reference: This document
Port Number: 6081

In addition, IANA is requested to create a "Geneve Option Class" registry to allocate Option Classes. This shall be a registry of 16-bit hexadecimal values along with descriptive strings. The identifiers 0x0-0xFF are to be reserved for standardized options for allocation by IETF Review [RFC5226] and 0xFFF0-0xFFFF for Experimental Use. Otherwise, identifiers are to be assigned to any organization with an interest in creating Geneve options on a First Come First Served basis. The registry is to be populated with the following initial values:

Option Class	Description
0x0000..0x00FF	Unassigned - IETF Review
0x0100	Linux
0x0101	Open vSwitch
0x0102	Open Virtual Networking (OVN)
0x0103	In-band Network Telemetry (INT)
0x0104	VMware
0x0105	Amazon
0x0106	Cisco
0x0107..0xFFEF	Unassigned - First Come First Served
0xFFF0..FFFF	Experimental

8. Contributors

The following individuals were authors of an earlier version of this document and made significant contributions:

Pankaj Garg
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

Email: pankajg@microsoft.com

Chris Wright
Red Hat Inc.
1801 Varsity Drive
Raleigh, NC 27606
USA

Email: chrisw@redhat.com

Puneet Agarwal
Innovium, Inc.
6001 America Center Drive
San Jose, CA 95002
USA

Email: puneet@innovium.com

Kenneth Duda
Arista Networks
5453 Great America Parkway
Santa Clara, CA 95054

USA

Email: kduda@arista.com

Dinesh G. Dutt
Cumulus Networks
140C S. Whisman Road
Mountain View, CA 94041
USA

Email: ddutt@cumulusnetworks.com

Jon Hudson
Independent

Email: jon.hudson@gmail.com

Ariel Hendel
Facebook, Inc.
1 Hacker Way
Menlo Park, CA 94025
USA

Email: ahendel@fb.com

9. Acknowledgements

The authors wish to thank Martin Casado, Bruce Davie and Dave Thaler for their input, feedback, and helpful suggestions.

10. References

10.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

10.2. Informative References

[ETYPES] The IEEE Registration Authority, "IEEE 802 Numbers", 2013, <http://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xml>.

[I-D.ietf-nvo3-dataplane-requirements] Bitar, N., Lasserre, M., Balus, F., Morin, T., Jin, L., and B. Khasnabish, "NVO3 Data Plane Requirements", draft-ietf-nvo3-dataplane-requirements-03 (work in progress), April 2014.

[I-D.ietf-nvo3-encap] Boutros, S., Ganga, I., Garg, P., Manur, R., Mizrahi, T., Mozes, D., Nordmark, E., Smith, M., Aldrin, S., and I. Bagdonas, "NVO3 Encapsulation Considerations", draft-ietf-nvo3-encap-01 (work in progress), October 2017.

[IEEE.802.1Q_2014] IEEE, "IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks", IEEE 802.1Q-2014, DOI 10.1109/ieeestd.2014.6991462, December 2014, <http://ieeexplore.ieee.org/servlet/opac?punumber=6991460>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <https://www.rfc-editor.org/info/rfc1981>.

[RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <https://www.rfc-editor.org/info/rfc2983>.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <https://www.rfc-editor.org/info/rfc3031>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[RFC3985] Bryant, S., Ed. and P. Pate, Ed., "Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture", RFC 3985, DOI 10.17487/RFC3985, March 2005, <https://www.rfc-editor.org/info/rfc3985>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, DOI 10.17487/RFC5374, November 2008, <https://www.rfc-editor.org/info/rfc5374>.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <https://www.rfc-editor.org/info/rfc6040>.

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <https://www.rfc-editor.org/info/rfc6935>.

[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, DOI 10.17487/RFC7348, August 2014, <https://www.rfc-editor.org/info/rfc7348>.

[RFC7365] Lasserre, M., Balus, F., Morin, T., Bitar, N., and Y. Rekhter, "Framework for Data Center (DC) Network Virtualization", RFC 7365, DOI 10.17487/RFC7365, October 2014, <https://www.rfc-editor.org/info/rfc7365>.

[RFC7637] Garg, P., Ed. and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <https://www.rfc-editor.org/info/rfc7637>.

[RFC8014] Black, D., Hudson, J., Kreeger, L., Lasserre, M., and T. Narten, "An Architecture for Data-Center Network Virtualization over Layer 3 (NVO3)", RFC 8014, DOI 10.17487/RFC8014, December 2016, <https://www.rfc-editor.org/info/rfc8014>.

[VL2] Greenberg, A., et al., "VL2: A Scalable and Flexible Data Center Network", ACM SIGCOMM Computer Communication Review, DOI 10.1145/1594977.1592576, 2009, <http://www.sigcomm.org/sites/default/files/ccr/papers/2009/October/1594977-1592576.pdf>.

Authors' Addresses

Jesse Gross (editor)

Email: jesse@kernel.org

Ilango Ganga (editor)
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95054
USA

Email: ilango.s.ganga@intel.com

T. Sridhar (editor)
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: tsridhar@vmware.com

Network Virtualization Overlays (nvo3)
Internet-Draft
Intended status: Standard track
Expires May 1, 2017

T. Herbert
Facebook
L. Yong
Huawei USA
O. Zia
Microsoft

October 28, 2016

Generic UDP Encapsulation
draft-ietf-nvo3-gue-05

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

This Internet-Draft will expire on May 1, 2017.

Copyright Notice

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents

Abstract

This specification describes Generic UDP Encapsulation (GUE), which is a scheme for using UDP to encapsulate packets of different IP protocols for transport across layer 3 networks. By encapsulating packets in UDP, specialized capabilities in networking hardware for efficient handling of UDP packets can be leveraged. GUE specifies basic encapsulation methods upon which higher level constructs, such tunnels and overlay networks for network virtualization, can be constructed. GUE is extensible by allowing optional data fields as part of the encapsulation, and is generic in that it can encapsulate packets of various IP protocols.

Table of Contents

1. Introduction

This specification describes Generic UDP Encapsulation (GUE) which is a general method for encapsulating packets of arbitrary IP protocols within User Datagram Protocol (UDP) [RFC0768] packets. Encapsulating packets in UDP facilitates efficient transport across networks. Networking devices widely provide protocol specific processing and optimizations for UDP (as well as TCP) packets. Packets for atypical IP protocols (those not usually parsed by networking hardware) can be encapsulated in UDP packets to maximize deliverability and to leverage flow specific mechanisms for routing and packet steering.

GUE provides an extensible header format for including optional data in the encapsulation header. This data potentially covers items such as virtual networking identifier, security data for validating or authenticating the GUE header, congestion control data, etc. GUE also allows private optional data in the encapsulation header. This feature can be used by a site or implementation to define local custom optional data, and allows experimentation of options that may eventually become standard.

This document does not define any specific GUE extensions. [GUEEXTENS] specifies a set of core extensions and [GUE4NVO3] defines an extension for using GUE with network virtualization.

The motivation for the GUE protocol is described in section 6.

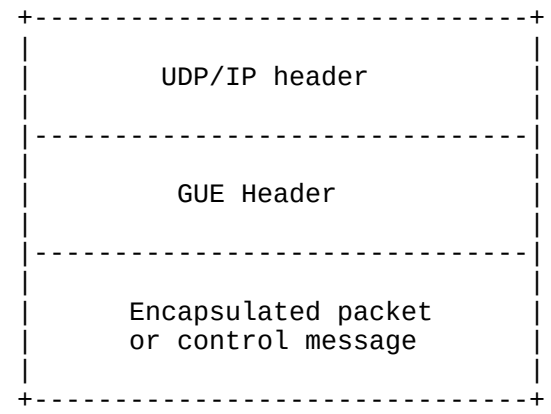
1.1 Terminology

GUE	Generic UDP Encapsulation
GUE Header	A variable length protocol header that is composed of a primary four byte header and zero or more four byte words for optional header data
GUE packet	A UDP/IP packet that contains a GUE header and GUE payload within the UDP payload
Encapsulator	A network node that encapsulates a packet in GUE
Decapsulator	A network node that decapsulates and processes packets encapsulated in GUE
Data message	An encapsulated packet in the GUE payload that is addressed to the protocol stack for an associated protocol
Control message	A formatted message in the GUE payload that is

	implicitly addressed to a decapsulator to monitor or control the state or behavior of a tunnel
Flags	A set of bit flags in the primary GUE header
Extension field	An optional field in a GUE header whose presence is indicated by corresponding flag(s)
C-bit	A single bit flag in the primary GUE header that indicates whether the GUE packet contains a control message or not.
Hlen	A field in the primary GUE header that gives the length of the GUE header
Proto/ctype	A field in the GUE header that holds either the IP protocol number for a data message or a type for a control message
Private data	Optional data in the GUE header that may be used for private purposes
Outer IP header	Refers to the outer most IP header of a packet when encapsulating a packet over IP
Inner IP header	Refers to an encapsulated IP header when an IP packets is encapsulated
Outer packet	Refers to an encapsulating packet
Inner packet	Refers to a packet that is encapsulated
Tunnel	An abstraction of a path across a network that ships packets or protocols across a network that normally wouldn’t support them. Tunnels provide communication paths between two endpoints. Encapsulation is one common technique used to actualize tunnels
Overlay network	A computer network that is built on top of another network
Underlay network	A network over which an overlay network is built

2. Base packet format

A GUE packet is comprised of a UDP packet whose payload is a GUE header followed by a payload which is either an encapsulated packet of some IP protocol or a control message (like an OAM message). A GUE packet has the general format:



The GUE header is variable length as determined by the presence of optional extension fields.

2.1. GUE version

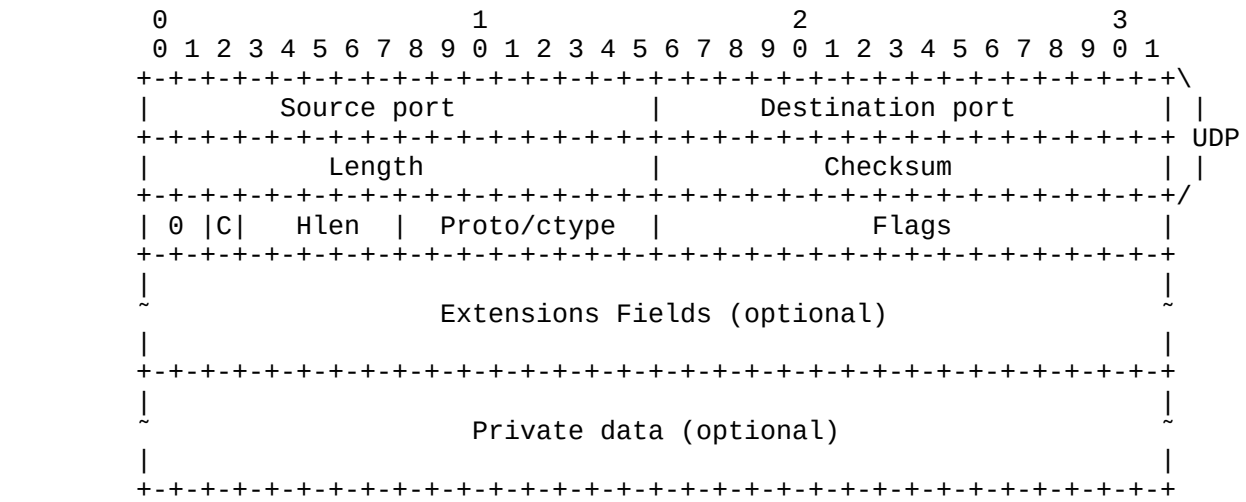
The first two bits of the GUE header contain the GUE protocol version number. The rest of the fields after the GUE version number are defined based on the version number. Versions 0 and 1 are described in this specification; versions 2 and 3 are reserved.

3. Version 0

Version 0 of GUE defines a generic extensible format to encapsulate packets by Internet protocol number.

3.1. Header format

The header format for version 0 of GUE in UDP is:



The contents of the UDP header are:

- Source port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the source port in the local tuple. When connection semantics are not applied this should be set to a flow entropy value for use with ECMP; the properties of flow entropy are described in section 5.11.

- Destination port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the destination port for the tuple. If connection semantics are not applied this is set to the GUE assigned port number, 6080.

- Length: Canonical length of the UDP packet (length of UDP header and payload).

- Checksum: Standard UDP checksum (handling is described in section 5.7).

The GUE header consists of:

- Ver: GUE protocol version (0).

- C: C-bit. When set indicates a control message, not set indicates a data message.

- Hlen: Length in 32-bit words of the GUE header, including optional extension fields but not the first four bytes of the header. Computed as (header_len - 4) / 4. All GUE headers are a multiple of four bytes in length. Maximum header length is 128 bytes.

- Proto/ctype: When the C-bit is set this field contains a control message type for the payload (section 3.2.2). When C-bit is not set, the field holds the Internet protocol number for the encapsulated packet in the payload (section 3.2.1). The control message or encapsulated packet begins at the offset provided by Hlen.

- Flags. Header flags that may be allocated for various purposes and may indicate presence of extension fields. Undefined header flag bits MUST be set to zero on transmission.

- Extension Fields: Optional fields whose presence is indicated by corresponding flags.

- Private data: Optional private data (see section 3.4). If private data is present it immediately follows that last extension field present in the header. The length of this data is determined by subtracting the starting offset from the header length.

3.2. Proto/ctype field

The proto/ctype field contains the type of the GUE payload. This can either be an IP protocol number or a control message type number. Intermediate devices may parse the GUE payload per the number in the proto/ctype field, and header flags cannot affect the interpretation of the proto/ctype field.

3.2.1 Proto field

When the C-bit is not set the proto/ctype field contains an IANA Internet Protocol Number. The protocol number is interpreted relative to the IP protocol that encapsulates the UDP packet (i.e. protocol of the outer IP header).

When the outer IP protocol is IPv4 the proto field may be set to any number except for those that refer to IPv6 extension headers or ICMPv6 options (number 58). An exception is that the destination options extension header using the PadN option may be used with IPv4 as described in section 3.6. The "no next header" protocol number (59) may be used with IPv4 as described below.

When the outer IP protocol is IPv6 the proto field may be set to any defined protocol number except Hop-by-hop options (number 0). If a received GUE packet in IPv6 contains a protocol number that is an extension header (e.g. Destination Options) then the extension header is processed after the GUE header as though the GUE header itself were an extension header.

IP protocol number 59 ("No next header") may be set to indicate that the GUE payload does not begin with the header of an IP protocol. This would be the case, for instance, if the GUE payload were a fragment when performing GUE level fragmentation. The interpretation of the payload is performed through other means (such as flags and extension fields), and intermediate devices must not parse packets based on the IP protocol number in this case.

3.2.2 Ctype field

When the C-bit is set, the proto/ctype field must be set to a valid control message type. A value of zero indicates that the GUE payload requires further interpretation to deduce the control type. This might be the case when the payload is a fragment of a control message, where only the reassembled packet can be interpreted as a control message.

Control message types 1 through 127 may be defined in standards. Types 128 through 255 are reserved to be user defined for experimentation or private control messages.

This document does not specify any standard control message types other than type 0.

3.3. Flags and extension fields

Flags and associated extension fields are the primary mechanism of extensibility in GUE. As mentioned in section 3.1 GUE header flags may indicate the presence of optional extension fields in the GUE header. [GUEXTENS] defines a basic set of GUE extensions.

3.3.1. Requirements

There are sixteen flag bits in the GUE header. A flag may indicate presence of an extension fields. The size of an extension field indicated by a flag must be fixed.

Flags may be paired together to allow different lengths for an extension field. For example, if two flag bits are paired, a field may possibly be three different lengths. Regardless of how flag bits may be paired, the lengths and offsets of optional fields

corresponding to a set of flags must be well defined.

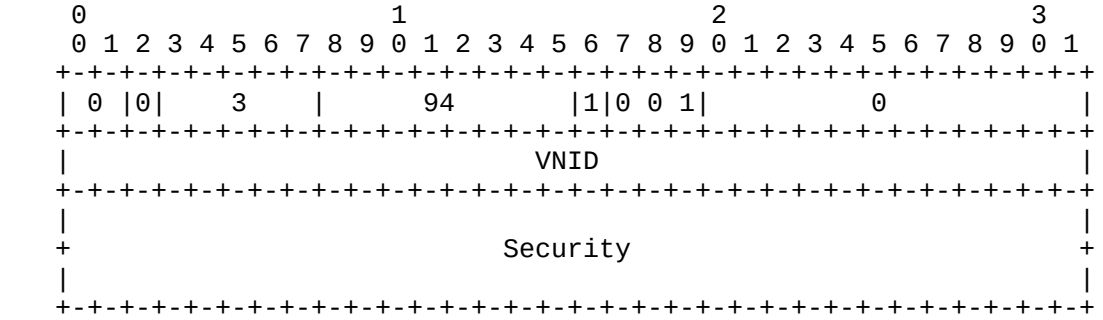
Extension fields are placed in order of the flags. New flags are to be allocated from high to low order bit contiguously without holes. Flags allow random access, for instance to inspect the field corresponding to the Nth flag bit, an implementation only considers the previous N-1 flags to determine the offset. Flags after the Nth flag are not pertinent in calculating the offset of an extension field indicated by the Nth flag. Random access of flags and fields permits processing of optional extensions in an order that is independent of their position in the packet. The processing order of extensions defined in [GUEEXTENS] demonstrates this property.

Flags (or paired flags) are idempotent such that new flags must not cause reinterpretation of old flags. Also, new flags should not alter interpretation of other elements in the GUE header nor how the message is parsed (for instance, in a data message the proto/ctype field always holds an IP protocol number as an invariant).

The set of available flags may be extended in the future by defining a "flag extensions bit" that refers to a field containing a new set of flags.

3.3.2. Example GUE header with extension fields

An example GUE header for a data message encapsulating an IPv4 packet and containing the VNID and Security extension fields (both defined in [GUEXTENS]) is shown below:



In the above example, the first flag bit is set which indicates that the VNID extension is present; this is a 32 bit field. The second through fourth bits of the flags are paired flags that indicate the presence of a security field with seven possible sizes. In this example 001 indicates a sixty-four bit security field.

3.4. Private data

An implementation may use private data for its own use. The private data immediately follows the last extension field in the GUE header and is not a fixed length. This data is considered part of the GUE header and must be accounted for in header length (Hlen). The length of the private data must be a multiple of four and is determined by subtracting the offset of private data in the GUE header from the header length. Specifically:

```
Private_length = (Hlen * 4) - Length(flags)
```

Where "Length(flags)" returns the sum of lengths of all the extension fields present in the GUE header. When there is no private data present, the length of the private data is zero.

The semantics and interpretation of private data are implementation specific. The private data may be structured as necessary, for instance it might contain its own set of flags and extension fields.

An encapsulator and decapsulator MUST agree on the meaning of private data before using it. The mechanism to achieve this agreement is outside the scope of this document but could include implementation-defined behavior, coordinated configuration, in-band communication using GUE control messages, or out-of-band messages.

If a decapsulator receives a GUE packet with private data, it MUST validate the private data appropriately. If a decapsulator does not expect private data from an encapsulator the packet MUST be dropped. If a decapsulator cannot validate the contents of private data per the provided semantics the packet MUST also be dropped. An implementation may place security data in GUE private data which must be verified for packet acceptance.

3.5. Message types

3.5.1. Control messages

Control messages carry formatted message that are implicitly addressed to the decapsulator to monitor or control the state or behavior of a tunnel (OAM). For instance, an echo request and corresponding echo reply message may be defined to test for liveness.

Control messages are indicated in the GUE header when the C-bit is set. The payload is interpreted as a control message with type specified in the proto/ctype field. The format and contents of the control message are indicated by the type and can be variable length.

Other than interpreting the proto/ctype field as a control message type, the meaning and semantics of the rest of the elements in the GUE header are the same as that of data messages. Forwarding and routing of control messages should be the same as that of a data message with the same outer IP and UDP header and GUE flags-- this ensures that control messages can be created that follow the same path as data messages.

3.5.2. Data messages

Data messages carry encapsulated packets that are addressed to the protocol stack for the associated protocol. Data messages are a primary means of encapsulation and can be used to create tunnels for overlay networks.

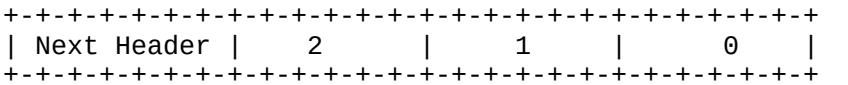
Data messages are indicated in GUE header when the C-bit is not set. The payload of a data message is interpreted as an encapsulated packet of an Internet protocol indicated in the proto/ctype field. The encapsulated packet immediately follows the GUE header.

3.6. Hiding the transport layer protocol number

The GUE header indicates the Internet protocol of the encapsulated packet. This is either contained in the Proto/ctype field of the primary GUE header, or is contained in the Payload Type field of a GUE Transform Field (used to encrypt the payload with DTLS, [GUESEC]). If the protocol number must be obfuscated, that is the transport protocol in use must be hidden from the network, then a trivial destination options can be used at the beginning of the payload.

The PadN destination option can be used to encode the transport protocol as a next header of an extension header (and maintain alignment of encapsulated transport headers). The Proto/ctype field or Payload Type field of the GUE Transform field is set to 60 to indicate that the first encapsulated header is a Destination Options extension header.

The format of the extension header is below:



For IPv4, it is permitted in GUE to use this precise destination option to contain the obfuscated protocol number. In this case next header must refer to a valid IP protocol for IPv4. No other extension headers or destination options are permitted with IPv4.

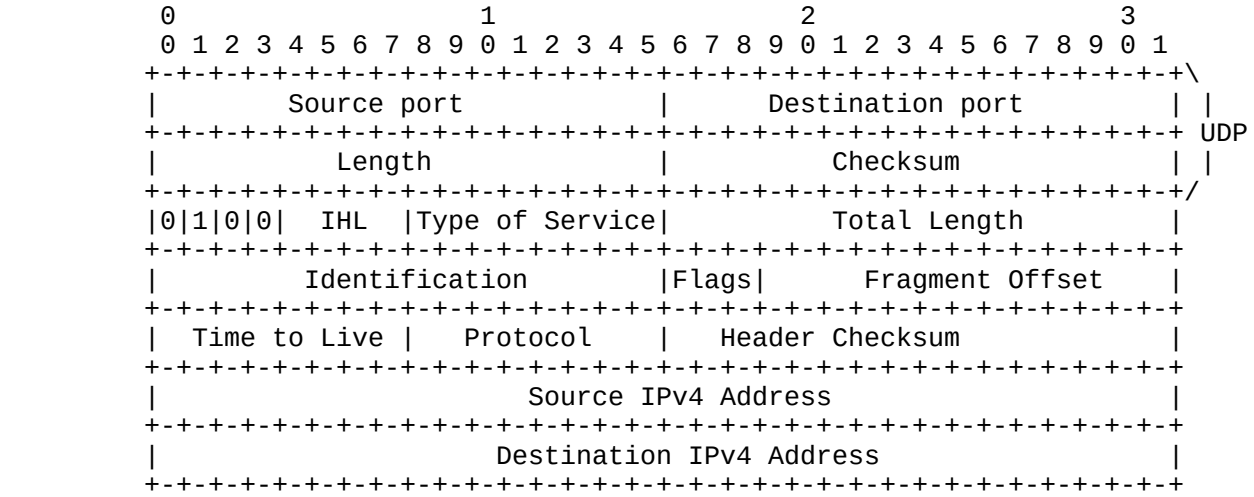
4. Version 1

Version 1 of GUE allows direct encapsulation of IPv4 and IPv6 in UDP. In this version there is no GUE header; a UDP packet encapsulates an IP packet. The first two bits of the UDP payload for GUE are the GUE version and coincide with the first two bits of the version number in the IP header. The first two version bits of IPv4 and IPv6 are 01, so we use GUE version 1 for direct IP encapsulation which makes two bits of GUE version to also be 01.

This technique is effectively a means to compress out the GUE header when encapsulating IPv4 or IPv6 packets and there are no flags or extension fields present. This method is compatible to use on the same port number as packets with the GUE header (GUE version 0 packets). This technique saves encapsulation overhead on costly links for the common use of IP encapsulation, and also obviates the need to allocate a separate port number for IP-over-UDP encapsulation.

4.1. Direct encapsulation of IPv4

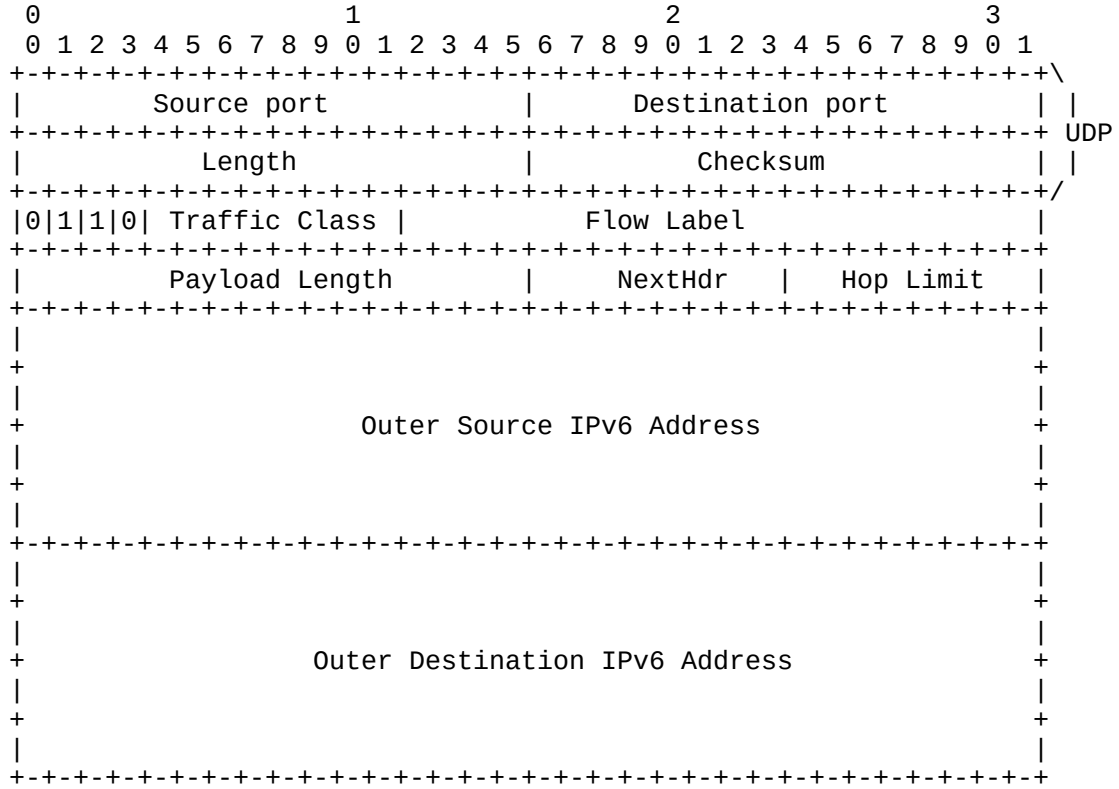
The format for encapsulating IPv4 directly in UDP is demonstrated below:



Note that 0100 value IP version field expresses the GUE version as 1 (bits 01) and IP version as 4 (bits 0100).

4.2. Direct encapsulation of IPv6

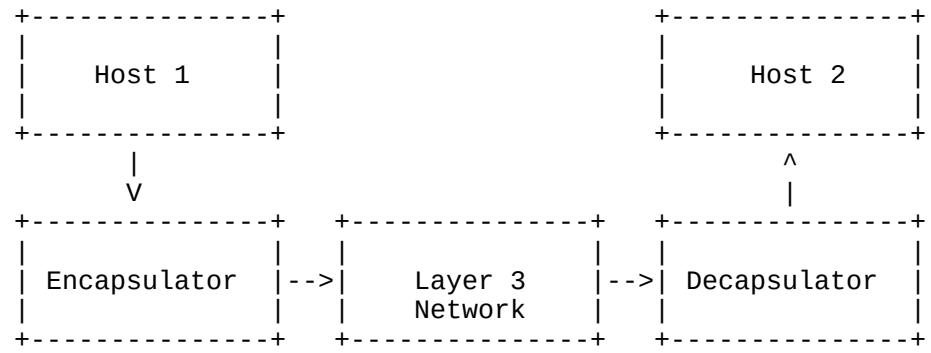
The format for encapsulating IPv4 directly in UDP is demonstrated below:



Note that 0110 value IP version field expresses the GUE version as 1 (bits 01) and IP version as 6 (bits 0110).

5. Operation

The figure below illustrates the use of GUE encapsulation between two hosts. Sever 1 is sending packets to host 2. An encapsulator performs encapsulation of packets from host 1. These encapsulated packets traverse the network as UDP packets. At the decapsulator, packets are decapsulated and sent on to host 2. Packet flow in the reverse direction need not be symmetric; GUE encapsulation is not required in the reverse path.



The encapsulator and decapsulator may be co-resident with the corresponding hosts, or may be on separate nodes in the network.

5.1. Network tunnel encapsulation

Network tunneling can be achieved by encapsulating layer 2 or layer 3 packets. In this case the encapsulator and decapsulator nodes are the tunnel endpoints. These could be routers that provide network tunnels on behalf of communicating hosts.

5.2. Transport layer encapsulation

When encapsulating layer 4 packets, the encapsulator and decapsulator should be co-resident with the hosts. In this case, the encapsulation headers are inserted between the IP header and the transport packet. The addresses in the IP header refer to both the endpoints of the encapsulation and the endpoints for terminating the the transport protocol. Note that the transport layer ports in the encapsulated packet are independent of the UDP ports in the outer packet.

Details about performing transport layer encapsulation are discussed in [TOU].

5.3. Encapsulator operation

Encapsulators create GUE data messages, set the fields of the UDP header, set flags and optional extension fields in the GUE header, and forward packets to a decapsulator.

An encapsulator may be an end host originating the packets of a flow, or may be a network device performing encapsulation on behalf of hosts (routers implementing tunnels for instance). In either case, the intended target (decapsulator) is indicated by the outer destination IP address and destination port in the UDP header.

If an encapsulator is tunneling packets, that is encapsulating packets of layer 2 or layer 3 protocols (e.g. EtherIP, IPIP, ESP tunnel mode), it should follow standard conventions for tunneling of one protocol over another. For instance, if an IP packet is being encapsualated in GUE then diffserv interaction [RFC2983] and ECN propagation for tunnels [RFC6040] should be followed.

5.4. Decapsulator operation

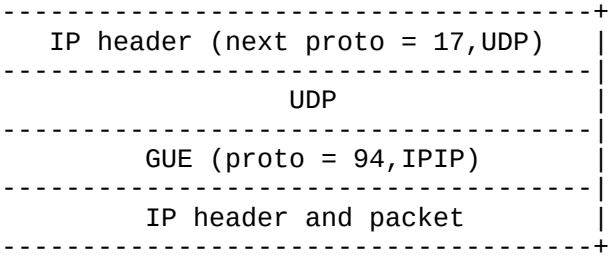
A decapsulator performs decapsulation of GUE packets. A decapsulator is addressed by the outer destination IP address of a GUE packet. The decapsulator validates packets, including fields of the GUE header.

If a decapsulator receives a GUE packet with an unsupported version, unknown flag, bad header length (too small for included extension fields), unknown control message type, bad protocol number, an unsupported Proto/ctype, or an otherwise malformed header, it MUST drop the packet. Such events may be logged subject to configuration and rate limiting of logging messages. No error message is returned back to the encapsulator. Note that set flags in GUE that are unknown to a decapsulator MUST NOT be ignored. If a GUE packet is received by a decapsulator with unknown flags, the packet MUST be dropped.

5.4.1. Processing a received data message

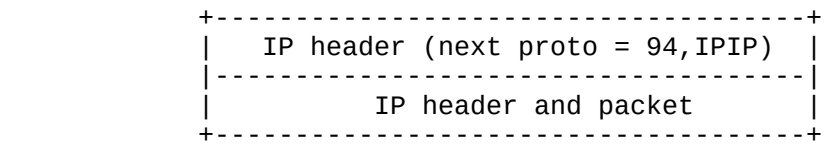
If a valid data message is received the UDP and GUE headers are removed from the packet. The outer IP header remains in tact and the next protocol in the header is set to the protocol from the proto field in the GUE header. The resulting packet is then resubmitted into the protocol stack to process that packet as though it was received with the protocol in the GUE header.

As an example, consider that a data message is received where GUE encapsulates an IP packet. In this case proto field in the GUE header is set 94 for IPIP:



The receiver removes the UDP and GUE headers and sets the next

protocol field in the IP packet to IPIP which is derived from the GUE proto field. The resultant packet would have the format:



This packet is then resubmitted into the protocol stack to be processed as an IPIP packet.

5.4.2. Processing a received control message

If a valid control message is received the packet must be processed as a control message. The specific processing to be performed depends on the ctype in the GUE header.

5.5. Router and switch operation

Routers and switches should forward GUE packets as standard UDP/IP packets. The outer five-tuple should contain sufficient information to perform flow classification corresponding to the flow of the inner packet. A switch should not normally need to parse a GUE header, and none of the flags or extension fields in the GUE header should affect routing.

An intermediate node SHOULD NOT modify a GUE header or GUE payload when forwarding packets since correctly identifying GUE packets in the network based on port numbers is not robust (see [RFC7605]). An intermediate node may encapsulate a GUE packet in another GUE packet, for instance to implement a network tunnel (i.e. by encapsulating an IP packet with a GUE payload in another IP packet as a GUE payload). In this case the router takes the role of an encapsulator, and the corresponding decapsulator is the logical endpoint of the tunnel. When encapsulating a GUE packet within another GUE packet, there are no provisions to automatically copy flags or extension fields to the outer GUE header. Each layer of encapsulation is considered independent.

5.6. Middlebox interactions

A middle box may interpret some flags and extension fields of the GUE header for classification purposes, but is not required to understand any of the flags or extension fields in GUE packets. A middle box must not drop a GUE packet because there are flags unknown to it. The header length in the GUE header allows a middlebox to inspect the payload packet without needing to parse the flags or extension

fields.

5.6.1. Connection semantics

A middlebox may infer bidirectional connection semantics for a UDP flow. For instance a stateful firewall may create a five-tuple rule to match flows on egress, and a corresponding five-tuple rule for matching ingress packets where the roles of source and destination are reversed for the IP addresses and UDP port numbers. To operate in this environment, a GUE tunnel must assume connected semantics defined by the UDP five tuple and the use of GUE encapsulation must be symmetric between both endpoints. The source port set in the UDP header must be the destination port the peer would set for replies. In this case the UDP source port for a tunnel would be a fixed value for a tunnel and not set to be flow entropy as described in section 5.11.

The selection of whether to make the UDP source port fixed or set to a flow entropy value for each packet sent should be configurable for a tunnel.

5.6.2. NAT

IP address and port translation can be performed on the UDP/IP headers adhering to the requirements for NAT with UDP [RFC4787]. In the case of stateful NAT, connection semantics must be applied to a GUE tunnel as described in section 5.6.1. GUE endpoints may also invoke STUN [RFC5389] or ICE [RFC5245] to manage NAT port mappings for encapsulations.

5.7. Checksum Handling

The potential for mis-delivery of packets due to corruption of IP, UDP, or GUE headers must be considered. Historically, the UDP checksum would be considered sufficient as a check against corruption of either the UDP header and payload or the IP addresses. Encapsulation protocols, such as GUE, may be originated or terminated on devices incapable of computing the UDP checksum for packet. This section discusses the requirements around checksum and alternatives that might be used when an endpoint does not support UDP checksum.

5.7.1. Requirements

One of the following requirements must be met:

- UDP checksums are enabled (for IPv4 or IPv6).

- The GUE header checksum is used (defined in [GUEEXTENS]).
- Use zero UDP checksums. This is always permissable with IPv4, in IPv6 they may only be used in accordance with applicable requirements in [GREUDP], [RFC6935], and [RFC6936].

5.7.2. UDP Checksum with IPv4

For UDP in IPv4, the UDP checksum MUST be processed as specified in [RFC768] and [RFC1122] for both transmit and receive. An encapsulator MAY set the UDP checksum to zero for performance or implementation considerations. The IPv4 header includes a checksum that protects against mis-delivery of the packet due to corruption of IP addresses. The UDP checksum potentially provides protection against corruption of the UDP header, GUE header, and GUE payload. Enabling or disabling the use of checksums is a deployment consideration that should take into account the risk and effects of packet corruption, and whether the packets in the network are already adequately protected by other, possibly stronger mechanisms such as the Ethernet CRC. If an encapsulator sets a zero UDP checksum for IPv4 it SHOULD use the GUE header checksum as described in [GUEEXTENS].

When a decapsulator receives a packet, the UDP checksum field MUST be processed. If the UDP checksum is non-zero, the decapsulator MUST verify the checksum before accepting the packet. By default a decapsulator SHOULD accept UDP packets with a zero checksum. A node MAY be configured to disallow zero checksums per [RFC1122]; this may be done selectively, for instance disallowing zero checksums from certain hosts that are known to be sending over paths subject to packet corruption. If verification of a non-zero checksum fails, a decapsulator lacks the capability to verify a non-zero checksum, or a packet with a zero-checksum was received and the decapsulator is configured to disallow, the packet MUST be dropped.

5.7.3. UDP Checksum with IPv6

In IPv6 there is no checksum in the IPv6 header that protects against mis-delivery due to address corruption. Therefore, when GUE is used over IPv6, either the UDP checksum must be enabled, the GUE header checksum must be used, or a zero UDP checksum is used if applicable requirements are met. Setting a zero checksum may be desirable for performance or implementation reasons, in which case the GUE header checksum MUST be used or requirements for using zero UDP checksums in [RFC6935] and [RFC6936] MUST be met. If the UDP checksum is enabled, then the GUE header checksum should not be used since it is mostly redundant.

When a decapsulator receives a packet, the UDP checksum field MUST be

processed. If the UDP checksum is non-zero, the decapsulator MUST verify the checksum before accepting the packet. By default a decapsulator MUST only accept UDP packets with a zero checksum if the GUE header checksum is used and is verified. If verification of a non-zero checksum fails, a decapsulator lacks the capability to verify a non-zero checksum, or a packet with a zero-checksum and no GUE header checksum was received, the packet MUST be dropped.

5.8. MTU and fragmentation

Standard conventions for handling of MTU (Maximum Transmission Unit) and fragmentation in conjunction with networking tunnels (encapsulation of layer 2 or layer 3 packets) should be followed. Details are described in MTU and Fragmentation Issues with In-the-Network Tunneling [RFC4459]

If a packet is fragmented before encapsulation in GUE, all the related fragments must be encapsulated using the same UDP source port. An operator should set MTU to account for encapsulation overhead and reduce the likelihood of fragmentation.

Alternative to IP fragmentation, the GUE fragmentation extension can be used. GUE fragmentation is described in [GUEEXTENS].

5.9. Congestion control

Per requirements of [RFC5405], if the IP traffic encapsulated with GUE implements proper congestion control no additional mechanisms should be required.

In the case that the encapsulated traffic does not implement any or sufficient control, or it is not known whether a transmitter will consistently implement proper congestion control, then congestion control at the encapsulation layer MUST be provided per RFC5405. Note this case applies to a significant use case in network virtualization in which guests run third party networking stacks that cannot be implicitly trusted to implement conformant congestion control.

Out of band mechanisms such as rate limiting, Managed Circuit Breaker [CIRCBRK], or traffic isolation may be used to provide rudimentary congestion control. For finer grained congestion control that allows alternate congestion control algorithms, reaction time within an RTT, and interaction with ECN, in-band mechanisms may be warranted.

5.10. Multicast

GUE packets may be multicast to decapsulators using a multicast destination address in the encapsulating IP headers. Each receiving

host will decapsulate the packet independently following normal decapsulator operations. The receiving decapsulators should agree on the same set of GUE parameters and properties; how such an agreement is reached is outside the scope of this document.

GUE allows encapsulation of unicast, broadcast, or multicast traffic. Flow entropy (the value in the UDP source port) may be generated from the header of encapsulated unicast or broadcast/multicast packets at an encapsulator. The mapping mechanism between the encapsulated multicast traffic and the multicast capability in the IP network is transparent and independent of the encapsulation and is otherwise outside the scope of this document.

5.11. Flow entropy for ECMP

5.11.1. Flow classification

A major objective of using GUE is that a network device can perform flow classification corresponding to the flow of the inner encapsulated packet based on the contents in the outer headers.

Hardware devices commonly perform hash computations on packet headers to classify packets into flows or flow buckets. Flow classification is done to support load balancing of flows across a set of networking resources. Examples of such load balancing techniques are Equal Cost Multipath routing (ECMP), port selection in Link Aggregation, and NIC device Receive Side Scaling (RSS). Hashes are usually either a three-tuple hash of IP protocol, source address, and destination address; or a five-tuple hash consisting of IP protocol, source address, destination address, source port, and destination port. Typically, networking hardware will compute five-tuple hashes for TCP and UDP, but only three-tuple hashes for other IP protocols. Since the five-tuple hash provides more granularity, load balancing can be finer grained with better distribution. When a packet is encapsulated with GUE and connection semantics are not applied, the source port in the outer UDP packet is set to a flow entropy value that corresponds to the flow of the inner packet. When a device computes a five-tuple hash on the outer UDP/IP header of a GUE packet, the resultant value classifies the packet per its inner flow.

Examples of deriving flow entropy for encapsulation are:

- If the encapsulated packet is a layer 4 packet, TCP/IPv4 for instance, the flow entropy could be based on the canonical five-tuple hash of the inner packet.

- If the encapsulated packet is an AH transport mode packet with TCP as next header, the flow entropy could be a hash over a

three-tuple: TCP protocol and TCP ports of the encapsulated packet.

- If a node is encrypting a packet using ESP tunnel mode and GUE encapsulation, the flow entropy could be based on the contents of clear-text packet. For instance, a canonical five-tuple hash for a TCP/IP packet could be used.

[RFC6438] discusses methods to compute and flow entropy value for IPv6 flow labels, those methods can also be used to create flow entropy values for GUE.

5.11.2. Flow entropy properties

The flow entropy is the value set in the UDP source port of a GUE packet. Flow entropy in the UDP source port should adhere to the following properties:

- The value set in the source port should be within the ephemeral port range (49152 to 65535 [RFC6335]). Since the high order two bits of the port are set to one this provides fourteen bits of entropy for the value.

- The flow entropy should have a uniform distribution across encapsulated flows.

- An encapsulator may occasionally change the flow entropy used for an inner flow per its discretion (for security, route selection, etc). To avoid thrashing or flapping the value, the flow entropy used for a flow should not change more than once every thirty seconds (or a configurable value).

- Decapsulators, or any networking devices, should not attempt to interpret flow entropy as anything more than an opaque value. Neither should they attempt to reproduce the hash calculation used by an encapasulator in creating a flow entropy value. They may use the value to match further receive packets for steering decisions, but cannot assume that the hash uniquely or permanently identifies a flow.

- Input to the flow entropy calculation is not restricted to ports and addresses; input could include flow label from an IPv6 packet, SPI from an ESP packet, or other flow related state in the encapsulator that is not necessarily conveyed in the packet.

- The assignment function for flow entropy should be randomly seeded to mitigate denial of service attacks. The seed may be changed periodically.

5.12. Negotiation of acceptable flags and extension fields

An encapsulator and decapsulator must achieve agreement about GUE parameters that will be used in communications. Parameters include GUE versions, flags and optional extension fields that can be used, security algorithms and keys, supported protocols and control messages, etc. This document proposes different general methods to accomplish this, the details of implementing these are considered out of scope.

General methods for this are:

- o Configuration. The parameters used for a tunnel are configured at each endpoint.
- o Negotiation. A tunnel negotiation can be performed. This could be accomplished in-band of GUE using control messages or private data.
- o Via a control plane. Parameters for communicating with a tunnel endpoint can be set in a control plane protocol (such as that needed for nvo3).
- o Via security negotiation. If security is used that would typically imply a key exchange between endpoints. Other GUE parameters may be conveyed as part of that process.

6. Motivation for GUE

This section presents the motivation for GUE with respect to other encapsulation methods.

6.1. Benefits of GUE

- * GUE is a generic encapsulation protocol. GUE can encapsulate protocols that are represented by an IP protocol number. This includes layer 2, layer 3, and layer 4 protocols.
- * GUE is an extensible encapsulation protocol. Standardized optional data such as security, virtual networking identifiers, fragmentation are being defined.
- * GUE allows private data to be sent as part of the encapsulation. This permits experimentation or customization in deployment.
- * GUE allows sending of control messages such as OAM using the same GUE header format (for routing purposes) as normal data messages.

* GUE maximizes deliverability of non-UDP and non-TCP protocols.

* GUE provides a means for exposing per flow entropy for ECMP for atypical protocols such as SCTP, DCCP, ESP, etc.

6.2. Comparison of GUE to other encapsulations

A number of different encapsulation techniques have been proposed for the encapsulation of one protocol over another. EtherIP [RFC3378] provides layer 2 tunneling of Ethernet frames over IP. GRE [RFC2784], MPLS [RFC4023], and L2TP [RFC2661] provide methods for tunneling layer 2 and layer 3 packets over IP. NVGRE [RFC7637] and VXLAN [RFC7348] are proposals for encapsulation of layer 2 packets for network virtualization. IPIP [RFC2003] and Generic packet tunneling in IPv6 [RFC2473] provide methods for tunneling IP packets over IP.

Several proposals exist for encapsulating packets over UDP including ESP over UDP [RFC3948], TCP directly over UDP [TCPUDP], VXLAN [RFC7348], LISP [RFC6830] which encapsulates layer 3 packets, MPLS/UDP [7510], and Generic UDP Encapsulation for IP Tunneling (GRE over UDP)[GREUDP]. Generic UDP tunneling [GUT] is a proposal similar to GUE in that it aims to tunnel packets of IP protocols over UDP.

GUE has the following discriminating features:

- UDP encapsulation leverages specialized network device processing for efficient transport. The semantics for using the UDP source port for flow entropy as input to ECMP are defined in section 5.11.

- GUE permits encapsulation of arbitrary IP protocols, which includes layer 2 3, and 4 protocols.

- Multiple protocols can be multiplexed over a single UDP port number. This is in contrast to techniques to encapsulate protocols over UDP using a protocol specific port number (such as ESP/UDP, GRE/UDP, SCTP/UDP). GUE provides a uniform and extensible mechanism for encapsulating various IP protocols in UDP with minimal overhead (four bytes of additional header).

- GUE is extensible. New flags and extension fields can be defined.

- The GUE header includes a header length field. This allows a network node to inspect an encapsulated packet without needing to parse the full encapsulation header.

- Private data in the encapsulation header allows local

customization and experimentation while being compatible with processing in network nodes (routers and middleboxes).

- o GUE includes both data messages (encapsulation of packets) and control messages (such as OAM).

- o The flags-field model facilitates efficient implementation of extensibility in hardware.

  For instance a TCAM can be use to parse a known set of N flags where the number of entries in the TCAM is 2^N.

  By comparison, the number of TCAM entries needed to parse a set of N arbitrarily ordered TLVS is:

  N! + (N N-1)(N-1)! + (N N-2)(N-2)! + ... + (N 2)2! + (N 1)1!

7. Security Considerations

There are two important considerations of security with respect to GUE.

- o Authentication and integrity of the GUE header

- o Authentication, integrity, and confidentiality of the GUE payload.

Security is integrated into GUE by the use of GUE security related extensions; these are defined in [GUEEXTENS]. These extensions include methods to authenticate the GUE header and encrypt the GUE payload.

IPsec in transport mode may be used to authenticate or encrypt GUE packets (GUE header and payload). Existing network security mechanisms, such as address spoofing detection, DDOS mitigation, and transparent encrypted tunnels can be applied to GUE packets.

A hash function for computing flow entropy (section 5.11) should be randomly seeded to mitigate some possible denial service attacks.

8. IANA Consideration

8.1. UDP source port

A user UDP port number assignment for GUE has been assigned:

```
Service Name: gue
Transport Protocol(s): UDP
Assignee: Tom Herbert <therbert@google.com>
Contact: Tom Herbert <therbert@google.com>
Description: Generic UDP Encapsulation
Reference: draft-herbert-gue
Port Number: 6080
Service Code: N/A
Known Unauthorized Uses: N/A
Assignment Notes: N/A
```

8.2. GUE version number

IANA is requested to set up a registry for the GUE version number. The GUE version number is 2 bits containing four possible values. This document defines version 0 and 1. New values are assigned via Standards Action [RFC5226].

Version number	Description	Reference
0	Version 0	This document
1	Version 1	This document
2..3	Unassigned	

8.3. Control types

IANA is requested to set up a registry for the GUE control types. Control types are 8 bit values. New values for control types 1-127 are assigned via Standards Action [RFC5226].

Control type	Description	Reference
0	Need further interpretation	This document
1..127	Unassigned	
128..255	User defined	This document

8.4. Flag-fields

IANA is requested to create a "GUE flag-fields" registry to allocate flags and extension fields used with GUE. This shall be a registry of bit assignments for flags, length of extension fields for corresponding flags, and descriptive strings. There are sixteen bits for primary GUE header flags (bit number 0-15). New values are assigned via Standards Action [RFC5226].

Flags bits	Field size	Description	Reference
Bit 0	4 bytes	VNID	[GUE4NVO3]
Bit 1..3	001->8 bytes 010->16 bytes 011->32 bytes	Security	[GUEEXTENS]
Bit 4	8 bytes	Fragmen-tation	[GUEEXTENS]
Bit 5	4 bytes	Payload transform	[GUEEXTENS]
Bit 6	4 bytes	Remote checksum offload	[GUEEXTENS]
Bit 7	4 bytes	Checksum	[GUEEXTENS]
Bit 8..15		Unassigned	

New flags are to be allocated from high to low order bit contiguously without holes.

9. Acknowledgements

The authors would like to thank David Liu, Erik Nordmark, Fred Templin, Adrian Farrel, and Bob Briscoe for valuable input on this draft.

10. References

10.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://www.rfc-editor.org/info/rfc768>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, DOI 10.17487/RFC2434, October 1998, <http://www.rfc-editor.org/info/rfc2434>.

[RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <http://www.rfc-editor.org/info/rfc2983>.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <http://www.rfc-editor.org/info/rfc6040>.

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <http://www.rfc-editor.org/info/rfc6935>.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <http://www.rfc-editor.org/info/rfc6936>.

[RFC4459] Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, DOI 10.17487/RFC4459, April 2006, <http://www.rfc-editor.org/info/rfc4459>.

10.2. Informative References

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004, <http://www.rfc-editor.org/info/rfc3828>.

[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, <http://www.rfc-editor.org/info/rfc7348>.

[RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <http://www.rfc-editor.org/info/rfc7605>.

[RFC7637] Garg, P., Ed., and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <http://www.rfc-editor.org/info/rfc7637>.

[RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black, "Encapsulating MPLS in UDP", RFC 7510, DOI 10.17487/RFC7510, April 2015, <http://www.rfc-editor.org/info/rfc7510>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <http://www.rfc-editor.org/info/rfc4340>.

[RFC4787] Audet, F., Ed., and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <http://www.rfc-editor.org/info/rfc4787>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <http://www.rfc-editor.org/info/rfc5389>.

[RFC5285] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <http://www.rfc-editor.org/info/rfc5245>.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <http://www.rfc-editor.org/info/rfc5405>.

[RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <http://www.rfc-editor.org/info/rfc7605>.

[RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011, <http://www.rfc-editor.org/info/rfc6438>.

[RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <http://www.rfc-editor.org/info/rfc2003>.

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, DOI 10.17487/RFC3948, January 2005, <http://www.rfc-editor.org/info/rfc3948>.

[RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <http://www.rfc-editor.org/info/rfc6830>.

[RFC3378] Housley, R. and S. Hollenbeck, "EtherIP: Tunneling Ethernet Frames in IP Datagrams", RFC 3378, DOI 10.17487/RFC3378, September 2002, <http://www.rfc-editor.org/info/rfc3378>.

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <http://www.rfc-editor.org/info/rfc2784>.

[RFC4023] Worster, T., Rekhter, Y., and E. Rosen, Ed., "Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)", RFC 4023, DOI 10.17487/RFC4023, March 2005, <http://www.rfc-editor.org/info/rfc4023>.

[RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <http://www.rfc-editor.org/info/rfc2661>.

[GUEEXTENS] Herbert, T., Yong, L., and Templin, F., "Extensions for Generic UDP Encapsulation" draft-herbert-gue-extensions-00

[GUE4NVO3] Yong, L., Herbert, T., Zia, O., "Generic UDP Encapsulation (GUE) for Network Virtualization Overlay" draft-hy-nvo3-gue-4-nvo-03

[GUESEC] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Secure Transport" draft-hy-gue-4-secure-transport-03

[TCPUDP] Chesire, S., Graessley, J., and McGuire, R., "Encapsulation of TCP and other Transport Protocols over UDP" draft-cheshire-tcp-over-udp-00

[TOU] Herbert, T., "Transport layer protocols over UDP" draft-herbert-transports-over-udp-00

[GREUDP] Crabbe, E., Yong, L., Xu, X., and Herbert, T., "Generic UDP Encapsulation for IP Tunneling" draft-ietf-tsvwg-gre-in-udp-encap-19

[GUT] Manner, J., Varia, N., and Briscoe, B., "Generic UDP Tunnelling (GUT) draft-manner-tsvwg-gut-02.txt"

[CIRCBRK] Fairhurst, G., "Network Transport Circuit Breakers", draft-ietf-tsvwg-circuit-breaker-15

[LCO] Cree, E., https://www.kernel.org/doc/Documentation/networking/checksum-offloads.txt

Appendix A: NIC processing for GUE

This appendix provides some guidelines for Network Interface Cards (NICs) to implement common offloads and accelerations to support GUE. Note that most of this discussion is generally applicable to other methods of UDP based encapsulation.

This appendix is informational and does not constitute a normative part of this document.

A.1. Receive multi-queue

Contemporary NICs support multiple receive descriptor queues (multi-queue). Multi-queue enables load balancing of network processing for a NIC across multiple CPUs. On packet reception, a NIC must select the appropriate queue for host processing. Receive Side Scaling is a common method which uses the flow hash for a packet to index an indirection table where each entry stores a queue number. Flow

Director and Accelerated Receive Flow Steering (aRFS) allow a host to program the queue that is used for a given flow which is identified either by an explicit five-tuple or by the flow's hash.

GUE encapsulation should be compatible with multi-queue NICs that support five-tuple hash calculation for UDP/IP packets as input to RSS. The flow entropy in the UDP source port ensures classification of the encapsulated flow even in the case that the outer source and destination addresses are the same for all flows (e.g. all flows are going over a single tunnel).

By default, UDP RSS support is often disabled in NICs to avoid out of order reception that can occur when UDP packets are fragmented. As discussed above, fragmentation of GUE packets should be mitigated by fragmenting packets before entering a tunnel, GUE fragmentation, path MTU discovery in higher layer protocols, or operator adjusting MTUs. Other UDP traffic may not implement such procedures to avoid fragmentation, so enabling UDP RSS support in the NIC should be a considered tradeoff during configuration.

A.2. Checksum offload

Many NICs provide capabilities to calculate standard ones complement payload checksum for packets in transmit or receive. When using GUE encapsulation there are at least two checksums that may be of interest: the encapsulated packet's transport checksum, and the UDP checksum in the outer header.

A.2.1. Transmit checksum offload

NICs may provide a protocol agnostic method to offload transmit checksum (NETIF_F_HW_CSUM in Linux parlance) that can be used with GUE. In this method the host provides checksum related parameters in a transmit descriptor for a packet. These parameters include the starting offset of data to checksum, the length of data to checksum, and the offset in the packet where the computed checksum is to be written. The host initializes the checksum field to pseudo header checksum.

In the case of GUE, the checksum for an encapsulated transport layer packet, a TCP packet for instance, can be offloaded by setting the appropriate checksum parameters.

NICs typically can offload only one transmit checksum per packet, so simultaneously offloading both an inner transport packet's checksum and the outer UDP checksum is likely not possible.

If an encapsulator is co-resident with a host, then checksum offload

may be performed using remote checksum offload (described in [GUEEXTENS]). Remote checksum offload relies on NIC offload of the simple UDP/IP checksum which is commonly supported even in legacy devices. In remote checksum offload the outer UDP checksum is set and the GUE header includes an option indicating the start and offset of the inner "offloaded" checksum. The inner checksum is initialized to the pseudo header checksum. When a decapsulator receives a GUE packet with the remote checksum offload option, it completes the offload operation by determining the packet checksum from the indicated start point to the end of the packet, and then adds this into the checksum field at the offset given in the option. Computing the checksum from the start to end of packet is efficient if checksum-complete is provided on the receiver.

Another alternative when an encapsulator is co-resident with a host is to perform Local Checksum Offload [LCO]. In this method the inner transport layer checksum is offloaded and the outer UDP checksum can be deduced based on the fact that the portion of the packet cover by the inner transport checksum will sum to zero (or at least the bit wise not of the inner pseudo header).

A.2.2. Receive checksum offload

GUE is compatible with NICs that perform a protocol agnostic receive checksum (CHECKSUM_COMPLETE in Linux parlance). In this technique, a NIC computes a ones complement checksum over all (or some predefined portion) of a packet. The computed value is provided to the host stack in the packet’s receive descriptor. The host driver can use this checksum to "patch up" and validate any inner packet transport checksum, as well as the outer UDP checksum if it is non-zero.

Many legacy NICs don’t provide checksum-complete but instead provide an indication that a checksum has been verified (CHECKSUM_UNNECESSARY in Linux). Usually, such validation is only done for simple TCP/IP or UDP/IP packets. If a NIC indicates that a UDP checksum is valid, the checksum-complete value for the UDP packet is the "not" of the pseudo header checksum. In this way, checksum-unnecessary can be converted to checksum-complete. So if the NIC provides checksum-unnecessary for the outer UDP header in an encapsulation, checksum conversion can be done so that the checksum-complete value is derived and can be used by the stack to validate checksums in the encapsulated packet.

A.3. Transmit Segmentation Offload

Transmit Segmentation Offload (TSO) is a NIC feature where a host provides a large (>MTU size) TCP packet to the NIC, which in turn splits the packet into separate segments and transmits each one. This is useful to reduce CPU load on the host.

The process of TSO can be generalized as:

- Split the TCP payload into segments which allow packets with size less than or equal to MTU.

- For each created segment:

  1. Replicate the TCP header and all preceding headers of the original packet.

  2. Set payload length fields in any headers to reflect the length of the segment.

  3. Set TCP sequence number to correctly reflect the offset of the TCP data in the stream.

  4. Recompute and set any checksums that either cover the payload of the packet or cover header which was changed by setting a payload length.

Following this general process, TSO can be extended to support TCP encapsulation in GUE. For each segment the Ethernet, outer IP, UDP header, GUE header, inner IP header if tunneling, and TCP headers are replicated. Any packet length header fields need to be set properly (including the length in the outer UDP header), and checksums need to be set correctly (including the outer UDP checksum if being used).

To facilitate TSO with GUE it is recommended that extension fields should not contain values that must be updated on a per segment basis-- for example, extension fields should not include checksums, lengths, or sequence numbers that refer to the payload. If the GUE header does not contain such fields then the TSO engine only needs to copy the bits in the GUE header when creating each segment and does not need to parse the GUE header.

A.4. Large Receive Offload

Large Receive Offload (LRO) is a NIC feature where packets of a TCP connection are reassembled, or coalesced, in the NIC and delivered to the host as one large packet. This feature can reduce CPU utilization in the host.

LRO requires significant protocol awareness to be implemented correctly and is difficult to generalize. Packets in the same flow need to be unambiguously identified. In the presence of tunnels or network virtualization, this may require more than a five-tuple match (for instance packets for flows in two different virtual networks may have identical five-tuples). Additionally, a NIC needs to perform

validation over packets that are being coalesced, and needs to fabricate a single meaningful header from all the coalesced packets.

The conservative approach to supporting LRO for GUE would be to assign packets to the same flow only if they have identical five-tuple and were encapsulated the same way. That is the outer IP addresses, the outer UDP ports, GUE protocol, GUE flags and fields, and inner five tuple are all identical.

Appendix B: Implementation considerations

This appendix is informational and does not constitute a normative part of this document.

B.1. Priveleged ports

Using the source port to contain a flow entropy value disallows the security method of a receiver enforcing that the source port be a privileged port. Privileged ports are defined by some operating systems to restrict source port binding. Unix, for instance, considered port number less than 1024 to be privileged.

Enforcing that packets are sent from a privileged port is widely considered an inadequate security mechanism and has been mostly deprecated. To approximate this behavior, an implementation could restrict a user from sending a packet destined to the GUE port without proper credentials.

B.2. Setting flow entropy as a route selector

An encapsulator generating flow entropy in the UDP source port may modulate the value to perform a type of multipath source routing. Assuming that networking switches perform ECMP based on the flow hash, a sender can affect the path by altering the flow entropy. For instance, a host may store a flow hash in its PCB for an inner flow, and may alter the value upon detecting that packets are traversing a lossy path. Changing the flow entropy for a flow should be subject to hysteresis (at most once every thirty seconds) to limit the number of out of order packets.

B.3. Hardware protocol implementation considerations

A low level protocol, such is GUE, is likely interesting to being supported by high speed network devices. Variable length header (VLH) protocols like GUE are often considered difficult to efficiently implement in hardware. In order to retain the important characteristics of an extensible and robust protocol, hardware vendors may practice "constrained flexibility". In this model, only

certain combinations or protocol header parameterizations are implemented in hardware fast path. Each such parameterization is fixed length so that the particular instance can be optimized as a fixed length protocol. In the case of GUE this constitutes specific combinations of GUE flags, fields, and next protocol. The selected combinations would naturally be the most common cases which form the "fast path", and other combinations are assumed to take the "slow path".

In time, needs and requirements of the protocol may change which may manifest themselves as new parameterizations to be supported in the fast path. To allow allow this extensibility, a device practicing constrained flexibility should allow the fast path parameterizations to be programmable.

Authors' Addresses

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA 94052
US

Email: tom@herbertland.com

Lucy Yong
Huawei USA
5340 Legacy Dr.
Plano, TX 75024
US

Email: lucy.yong@huawei.com

Osama Zia
Microsoft
1 Microsoft Way
Redmond, WA 98029
US

Email: osamaz@microsoft.com

NVO3 Working Group
INTERNET-DRAFT
Intended Status: Informational

Y. Li
D. Eastlake
Huawei Technologies
L. Kreeger
Arrcus, Inc
T. Narten
IBM
D. Black
Dell EMC

Expires: September 14, 2018
March 13, 2018

Split Network Virtualization Edge (Split-NVE) Control Plane Requirements
draft-ietf-nvo3-hpvr2nve-cp-req-17

Abstract

In a Split Network Virtualization Edge (Split-NVE) architecture, the functions of the NVE (Network Virtualization Edge) are split across a server and an external network equipment which is called an external NVE. The server-resident control plane functionality resides in control software, which may be part of hypervisor or container management software; for simplicity, this document refers to the hypervisor as the location of this software.

Control plane protocol(s) between a hypervisor and its associated external NVE(s) are used by 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 clarify the protocol requirements.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

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 (Network Virtualization Edge) 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 end device is called the tNVE (terminal-side NVE) and the portion located on the external NVE is called the nNVE (network-side NVE) 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 and/or virtual switch in an virtualized end device. 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 (Virtual Machine) connected to a VN (Virutal Network). 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 or Linux Container.

The NVO3 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, control plane protocol(s) between a hypervisor and its associated external NVE(s) are required for the hypervisor to distribute the virtual machines networking states to the NVE(s) for further handling. The protocol is an NVE-internal protocol and runs between tNVE and nNVE logical entities. This protocol is mentioned in the NVO3 problem statement [RFC7364] and appears as the third work item.

Virtual machine states and state transitioning are summarized in this document showing events where the NVE needs to take specific actions. Such events might correspond to actions the control plane signaling protocol(s) need to take between tNVE and nNVE in the Split-NVE scenario. The high level requirements to be fulfilled are stated.

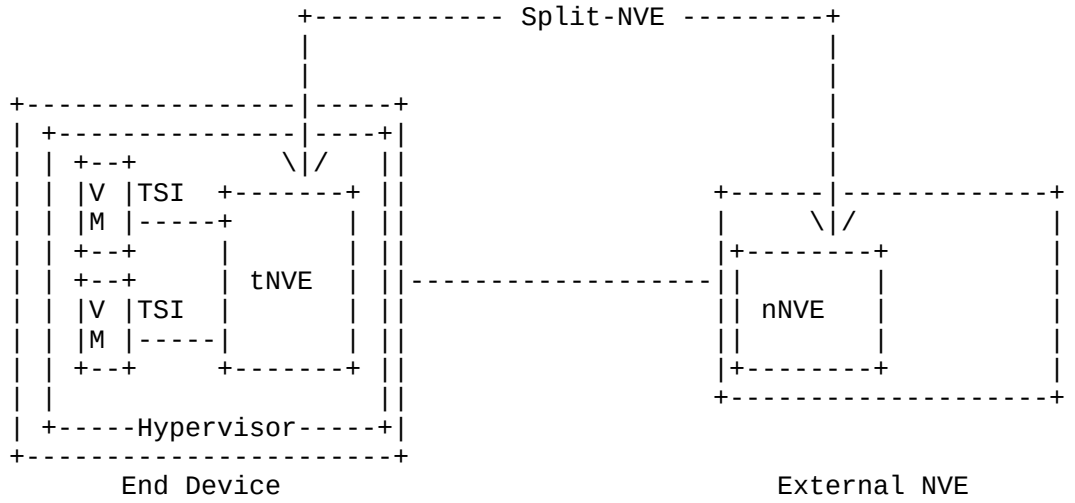


Figure 1 Split-NVE structure

This document uses VMs as an example of Tenant Systems (TSs) in order to describe the requirements, 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 is another type of TS, as are systems running on an OS-level virtualization technologies like containers. The fact that VMs have lifecycles (e.g., can be created and destroyed, can be moved, and can be started or stopped) results in a general set of protocol requirements, most of which are applicable to other forms of TSs although not all of the requirements are applicable to all forms of TSs.

Section 2 describes VM states and state transitioning in the VM's 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the same terminology as found in [RFC7365]. This section defines additional terminology used by this document.

Split-NVE: a type of NVE (Network Virtualization Edge) where the functionalities 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. It interacts with a tenant system through an internal interface in the end device.

nNVE: the portion of Split-NVE functionalities located on the network device that is directly or indirectly connected to the end device holding the corresponding tNVE. nNVE normally performs encapsulation to and decapsulation from the overlay network.

External NVE: the physical network device holding the nNVE

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

Container: Please refer to Hypervisor. For simplicity this document use the term hypervisor to represent both hypervisor and container.

VN Profile: Meta data associated with a VN (Virtual Network) that is applied to any attachment point to the VN. That is, VAP (Virtual Access Point) properties that are applied 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 (Network Virtualization Authority) could use the VN ID or VN Name to obtain the VN Profile.

VSI: Virtual Station Interface. [IEEE 802.1Q]

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

1.2 Target Scenarios

In the Split-NVE architecture, an external NVE can provide an offload of the encapsulation / decapsulation functions and network policy enforcement as well as the VN Overlay protocol overhead. This

offloading may improve performance and/or save resources in the End Device (e.g. hypervisor) using the external NVE.

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

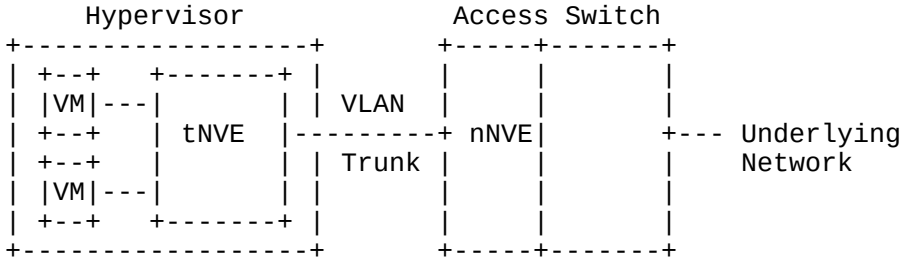


Figure 2 Hypervisor with an External NVE

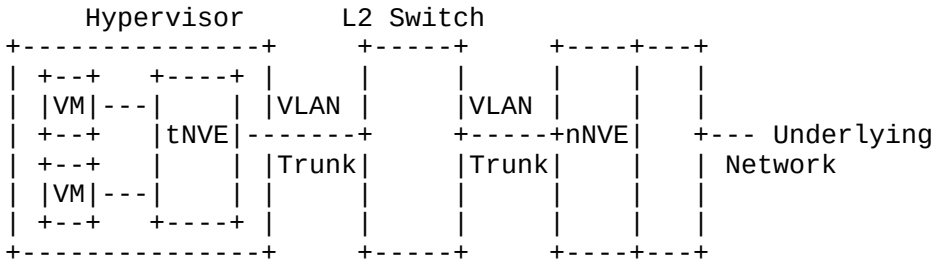


Figure 3 Hypervisor with an External NVE
connected through an Ethernet Access Switch

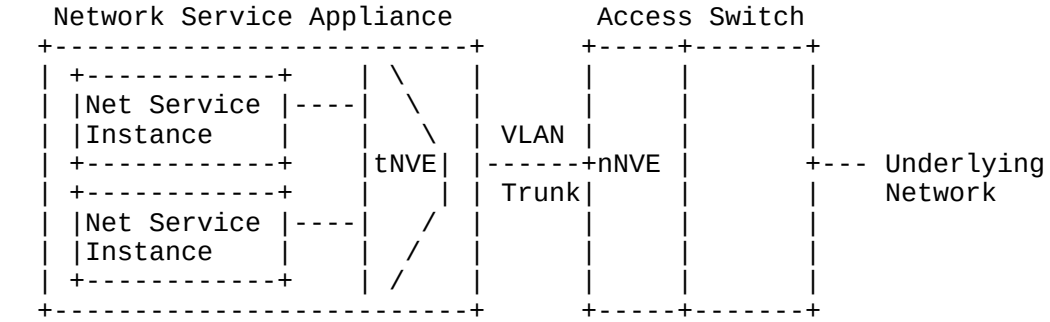


Figure 4 Physical Network Service Appliance with an External NVE

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) [RFC8014]. 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. An IP address can be in either IPv4 or IPv6 format. For example, Tenant Systems that are providing network services (such as transparent firewall, load balancer, or VPN gateway) are likely to have a 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.

Figures 2 through 4 show the use of VLANs to separate traffic for multiple VNs between the tNVE and nNVE; VLANs are not strictly necessary if only one VN is involved, but multiple VNs are expected in most cases. Hence this draft assumes the presence of VLANs.

2. VM Lifecycle

Figure 2 of [RFC7666] 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. Note 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

The VM creation event causes the VM state transition from Preparing to Shutdown and then to Running [RFC7666]. The end device allocates and initializes local virtual resources like storage in the VM Preparing state. In the 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. The transition from the Shutdown state to the Running state normally requires human action or a 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. Initially, when Running, 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 before it signals successful association with the TSI. Such preparation 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 the Running state, preferably in the Shutdown state. If association with an external NVE fails, the VM should not go into the 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 shutting down VM execution on one server and restarting it on another. For simplicity, the following abstract summary of 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 a migration event involves state transitions on both source hypervisor 1 and destination hypervisor 2. The VM state on source hypervisor 1 transits from Running to Migrating and then to Shutdown [RFC7666]. The 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 VLAN ID 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 until all the network provisioning and binding has been done.

The states of VM on the source and destination hypervisors both are Migrating during transfer of migration execution. 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 the VM on the source hypervisor stays in Migrating state for a while after the VM on the destination hypervisor enters Running state.

2.3 VM Termination Event

A VM termination event is also referred to as "powering off" a VM. A VM termination event leads to its state becoming Shutdown. There are two possible causes of VM termination [RFC7666]. One is the normal "power off" of a running VM; the other is that the VM has been migrated to another hypervisor and the VM image on the source hypervisor has to stop executing and 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

A 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 [RFC7666]. The VM can be easily re-activated from Paused state to Running state.

A VM suspension event leads to the VM transiting from Running state to Suspended state. A 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 [RFC7666].

In the Split-NVE architecture, the external NVE should not

disassociate the paused or suspended VM as the VM can return to Running state at any time.

3. Hypervisor-to-NVE Control Plane Protocol Functionality

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

3.1 VN Connect and Disconnect

In the Split-NVE scenario, a protocol is needed between the End Device (e.g. Hypervisor) and the external NVE it is using 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.

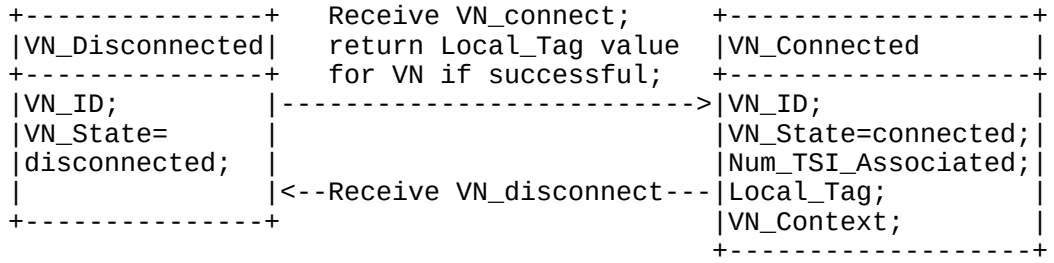


Figure 5. State Transition Example of a VAP Instance on an External NVE

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 to better explain the whole procedure; while others of them may not be shown.

The external NVE must be notified when an End Device requires connection to a particular VN and when it no longer requires connection. Connection clean up for the failed devices should be employed which is out of the scope of the protocol specified in this document.

In addition, the external NVE should provide a local tag value for each connected VN to the End Device to use for exchanging packets between the End Device and the external NVE (e.g. a locally significant [IEEE 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 their VN coming from a given TSI, strips the tag, 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]. The external NVE needs to synchronize the mapping information of the local tag and VN Name or VN ID with NVA.

The VN_connect message can be explicit or implicit. Explicit means the hypervisor sends a request message explicitly 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, that implicitly indicate its interest in connecting to a VN.

A VN_disconnect message indicates 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 for use by another 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 the corresponding VN to the NVA.

The NVE has two ways it can discover the tenant addresses for which frames are to 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. An 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 use the second approach above, the "hypervisor-to-NVE" protocol must support End Devices communicating 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.

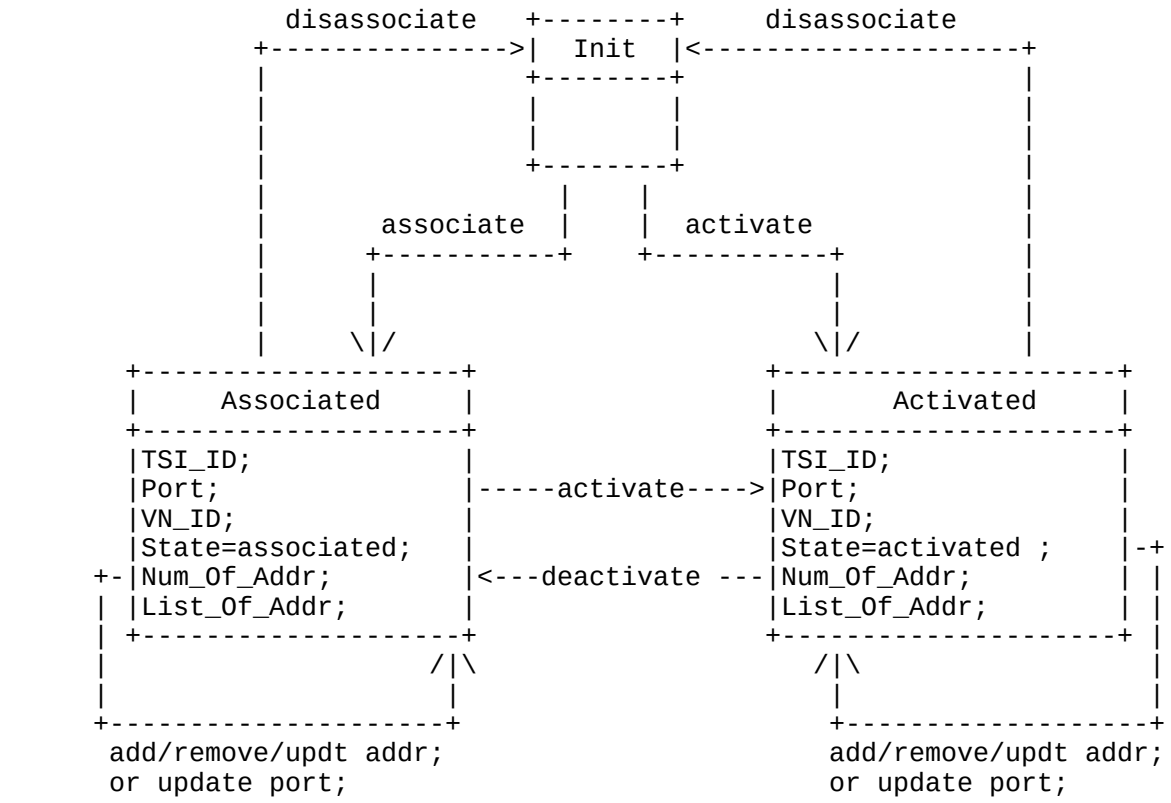


Figure 6 State Transition Example of a TSI Instance on an External NVE

The 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 a given port e.g. 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 their mapping tables but not their forwarding tables. An NVE may create ACL or filter rules based on the associated TSI addresses on that attached port p but not enable them yet. The local tag for the VN corresponding to the TSI instance should be provisioned on port p to receive packets.

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

the same practice.

The Activated state of a TSI instance on an external NVE indicates that all the addresses for that TSI are functioning correctly on a given port e.g. 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 the NVE are enabled. The 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 sending the Activate message from the hypervisor to the external NVE.

TSI information may get updated in either the Associated or Activated state. The following are considered updates to the TSI information: add or remove the associated addresses, update the current associated addresses (for example updating IP for a given MAC), and update the NVE port information based on where the NVE receives messages. Such updates do not change the state of TSI. When any address associated with a given TSI changes, the NVE should inform the NVA to update the mapping information for 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 updates will cause the provisioning of the local tag for the VN corresponding to the TSI instance on new port and removal from the old port.

3.3 TSI Disassociate and Deactivate

Disassociate and deactivate behaviors are conceptually the reverse 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. No traffic to or from the TSI is expected or allowed to pass through. For example, the NVE needs to tell the NVA to remove the relevant addresses mapping information from forwarding and routing tables. ACL and filtering rules regarding the relevant addresses should be disabled.

From Associated or Activated state to the Init state, the NVE releases 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.

A VM shutdown event will normally cause the relevant TSI instance(s) on the NVE to transition to Init state from Activated state. All resources should be released.

A VM migration will cause 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 instances.

Although the triggering messages for the state transition shown in Figure 6 does not indicate the difference between a VM creation/shutdown event and a VM migration arrival/departure event, the external NVE can make optimizations if it is given 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 an IGMP query [RFC2236] can be triggered by the destination external NVE to the migrated VM so that VM is forced to send an IGMP report to the multicast router. Then a multicast router can correctly route the multicast traffic to the new external NVE for those multicast groups the VM 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 the 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 to initiate 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 from 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 one or more locally-significant tag(s) for carrying traffic associated with a specific VN (e.g., [IEEE 802.1Q] tags).

Req-8: The protocol MUST allow an End Device initiating a request to associate/disassociate and/or activate/deactive some or all 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 some or all 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 and the connected End Device to authenticate each other.

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 is the result of a VM hot migration event.

5. VDP Applicability and Enhancement Needs

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

VDP facilitates the automatic discovery and configuration of Edge

Virtual Bridging (EVB) stations and Edge Virtual Bridging (EVB) bridges. An EVB station is normally an end station running multiple VMs. It is conceptually equivalent to a hypervisor in this document. An EVB bridge is conceptually equivalent to the external NVE.

VDP is able to pre-associate/associate/de-associate a VSI on an EVB station with a port on the EVB bridge. A VSI is approximately the concept of a virtual port by which a VM connects to the hypervisor in this document's context. The EVB station and the EVB bridge can reach 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 the Edge Control Protocol(ECP) [IEEE 802.1Q] 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 the NVO3 context.

<table>
<tr><td>Req</td><td>Supported by VDP?</td><td>remarks</td></tr>
<tr><td>Req-1</td><td rowspan="4">Partially</td><td rowspan="4">Needs extension. Must be able to send to a specific unicast MAC and should be able to send to a non-reserved well known multicast address other than the nearest customer bridge address.</td></tr>
<tr><td>Req-2</td></tr>
<tr><td>Req-3</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 or specify GroupID in request and get VID assigned in returning response. Multiple VLANs per group are allowed.</td></tr>
<tr><td rowspan="2">Req-8</td><td rowspan="2">Partially</td><td>requirements: associate/disassociate activate/deactivate</td><td>VDP equivalence: pre-asso/de-associate associate/de-associate</td></tr>
<tr><td colspan="2">Needs extension to allow associate->pre-assoc</td></tr>
</table>

Req-9	Yes	VDP bridge initiates de-associate
Req-10	Partially	Needs extension for IPv4/IPv6 address. Add a new "filter info format" type.
Req-11	No	Out-of-band mechanism is preferred, e.g. MACSec or 802.1X. Implicit authentication based on control of physical connectivity exists in VDP when the External NVE connects to the End Device directly and is reachable with the nearest customer bridge address.
Req-12	Yes	L2 protocol naturally
Req-13	Partially	M bit for migrated VM on destination hypervisor and S bit for that on source hypervisor. It is indistinguishable when M/S is 0 between no guidance and events not caused by migration where NVE may act differently. Needs new New bits for migration indication in new "filter info format" type.

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 an NVO3 network.

6. Security Considerations

External NVEs must ensure that only properly authorized Tenant Systems are allowed to join and become a part of any particular Virtual Network. In some cases, tNVE may want to connect to the nNVE for provisioning purposes. This may require that the tNVE authenticate the nNVE in addition to the nNVE authenticating the tNVE. If a secure channel is required between tNVE and nNVE to carry encrypted split-NVE control plane protocol, then existing mechanisms such as MACsec [IEEE 802.1AE] can be used. In some deployments, authentication may be implicit based on control of physical connectivity, e.g., if the nNVE is located in the bridge that is directly connected to the server that contains the tNVE. Use of "nearest customer bridge address" in VDP [IEEE 802.1Q] is an example where this sort of implicit authentication is possible, although explicit authentication also applies in that case.

As the control plane protocol results in configuration changes for

both the tNVE and nNVE, tNVE and nNVE implementations should log all state changes, including those described in Section 3. Implementations should also log significant protocol events, such as establishment or loss of control plane protocol connectivity between the tNVE and nNVE and authentication results.

In addition, external NVEs will need appropriate mechanisms to ensure that any hypervisor wishing to use the services of an NVE is properly authorized to do so. One design point is whether the hypervisor should supply the external NVE with necessary information (e.g., VM addresses, VN information, or other parameters) that the external 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 external NVE using that information to obtain the information needed to validate the hypervisor-provided parameters or obtain related parameters in a secure manner. The former approach can be used in a trusted environment so that the external NVE can directly use all the information retrieved from the hypervisor for local configuration. It saves the effort on the external NVE side from information retrieval and/or validation. The latter approach gives more reliable information as the external NVE needs to retrieve them from some management system database. Especially some network related parameters like VLAN IDs can be passed back to hypervisor to be used as a more authoritative provisioning. However in certain cases, it is difficult or inefficient for an external NVE to have access or query on some information to those management systems. Then the external NVE has to obtain those information from hypervisor.

7. IANA Considerations

No IANA action is required.

8. Acknowledgements

This document was initiated based on the merger of 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 Lucy Yong and Jon Hudson for their generous help in improving 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.

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

[RFC7666] Asai H., MacFaden M., Schoenwaelder J., Shima K., Tsou T., "Management Information Base for Virtual Machines Controlled by a Hypervisor", October 2015.

[RFC8014] Black, D., Hudson, J., Kreeger, L., Lasserre, M., Narten, T., "An Architecture for Data-Center Network Virtualization over Layer 3 (NVO3)", December 2016.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words ", BCP 14, RFC 8174, May 2017.

[IEEE 802.1Q] IEEE, "Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks", IEEE Std 802.1Q-2014, November 2014.

8.2 Informative References

[RFC2236] Fenner, W., "Internet Group Management Protocol, Version 2", RFC 2236, November 1997.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

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

[IEEE 802.1AE] IEEE, "MAC Security (MACsec)", IEEE Std 802.1AE-2006, August 2006.

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

The VDP (VSI Discovery and Discovery and Configuration Protocol, clause 41 of [IEEE 802.1Q]) can be considered as a controlling protocol running between the hypervisor and the external bridge. VDP association TLV structure are formatted as shown in Figure A.1.

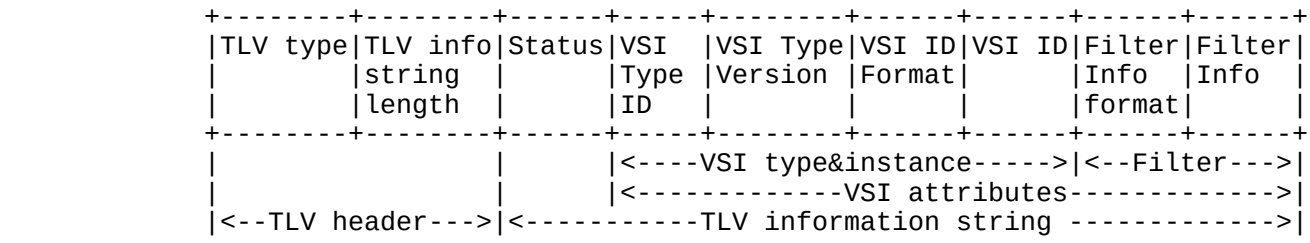


Figure A.1: VDP association TLV

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-associate 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 success it also reserves resources in the bridge to prepare for a subsequent Associate request.

3. Associate: Associate creates and activates an association between a VSI instance and a bridge port. An 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. De-associate: 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.

De-associate can be initiated by either side and the other types 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 defines 4 types. Each of the filter information is shown in details as follows.

1. VID Filter Info format

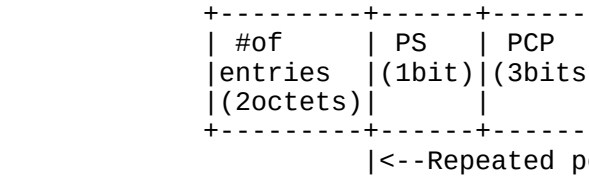


Figure A.2 VID Filter Info format

2. MAC/VID Filter Info format

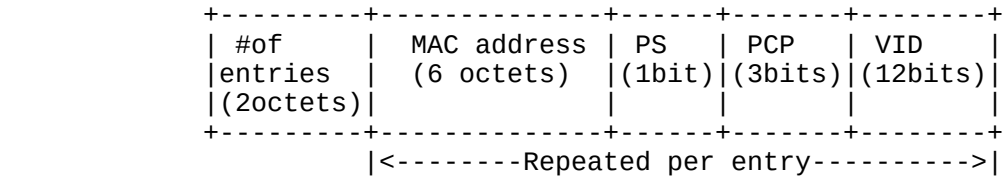


Figure A.3 MAC/VID filter format

3. GroupID/VID Filter Info format

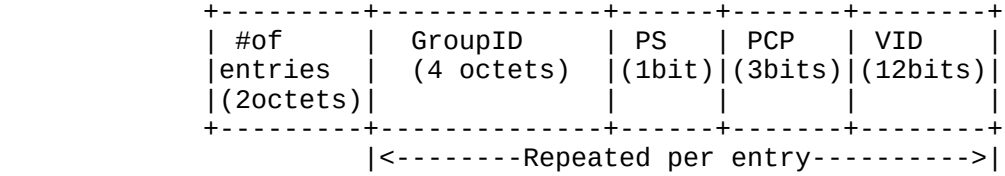


Figure A.4 GroupID/VID filter format

4. GroupID/MAC/VID Filter Info format

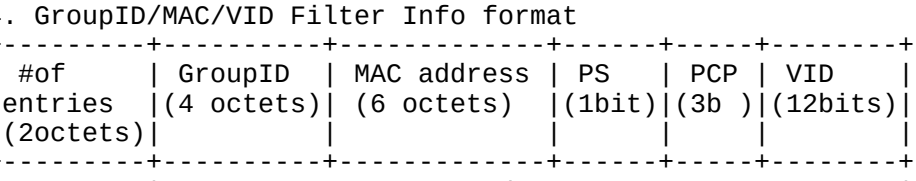


Figure A.5 GroupID/MAC/VID filter format

The null VID can be used in the VDP Request sent from the station 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 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 station to the bridge. The bridge maps GroupID to a locally significant VLAN ID.

The VSI ID in VDP association TLV that identify a VM can be one of the following format: IPV4 address, IPV6 address, MAC address, UUID [RFC4122], or locally defined.

Authors' Addresses

Yizhou Li
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

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

Donald Eastlake
Huawei R&D USA
155 Beaver Street
Milford, MA 01757 USA

Phone: +1-508-333-2270
EMail: d3e3e3@gmail.com

Lawrence Kreeger
Arrcus, Inc

Email: lkreeger@gmail.com

Thomas Narten
IBM

Email: narten@us.ibm.com

David Black
Dell EMC
176 South Street,
Hopkinton, MA 01748 USA

Email: david.black@dell.com

NVO3
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2017

S. Dikshit
A. Sujeet Nayak
Cisco Systems
June 15, 2017

PMTUD Over Vxlan
draft-saum-nvo3-pmtud-over-vxlan-05

Abstract

Path MTU Discovery between hosts/VM/servers/end-points connected over a Data-Center/Service-Provider Overlay Network, is still an unattended problem. It needs a converged solution to ensure optimal usage of network and computational resources for all hooked end-point devices.

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/.

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

This Internet-Draft will expire on December 17, 2017.

Table of Contents

1. Introduction

There is an operational disconnect between underlay network provisioned as the core network, and the overlay network which intends to connect islands of customer deployments. The deployments can range from cloud based services to storage applications or web(over the top) servers hosted over virtual machines or any other end devices like blade servers. Overlay network are provisioned as tunnels leveraging Vxlan (and associated ones like gpe, geneve, gue), NVGRE, MPLS and other overlay encapsulations.

The end hosts in a typical datacenter deployment are connected to devices termed as ToR (top of rack devices). These are the networking devices which encapsulate the packet in an Overlay construct and relays it over Data center core network. Although a ToR device MAY NOT always be a gateway for an overlay.

IPv6/IPv4 enabled hosts/end-points, triggering PMTUD, may not get the right (or any) information from (over) the core network. This document validates the solution for Vxlan core network (overlay) in a data center deployment. This solution is equally applicable to any other tunnel specific core network deployments.

The proposal in this document, formulates an integrated approach which falls inline with OAM modelling discussed in NVO3./>.

2. Requirements

2.1. Requirements Language

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].

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

2.2. Solution Requirements

This section describes the advantages of the proposed solution, considering deployment in a typical data center core network:

(a) Optimal use of bandwidth in core and client side network of typical data center deployment.

(b) In case Vxlan Gateway nodes complies to this solution, it MAY avoid black holing.

(c) All end host applications (like web servers) can tailor the MSS accordingly against their respective transports.

(d) Facilitates seamless integration of IPv6 or dual stack applications over IPv4 based overlays and vice versa.

(e) The proposed solution is applicable to all encapsulations [RFC7348], [I-D.draft-ietf-nvo3-vxlan-gpe], [I-D.draft-ietf-nvo3-gue], [I-D.draft-gross-geneve] and [RFC7637]. Although the problem and solution refers to VXLAN [RFC7348] as a use-case in this document.

3. Problem Description

In current vendor implementation(s) of Vxlan-Gateway/ToR-device or other network devices, which form part of core data center network and is configured with an overlay(tunnel) mechanism to transport packets from one customer end point to another, are incapable of relaying the errors encountered in routing/switching path in their networks (underlay network) to the customer end points (hosts/vm/ blade-servers). This deems right, as core-network should be transparent and water-tight with respect to leaking any public (core)

network information to customer devices (and vice versa), thus ensuring seclusion between different customers provisioning tunneled over the same core network.

For example, the information carried in the IP header of a Vxlan encapsulated packet is transparent to the payload (end-point generated packet). Hence, any network-specific information related to IPv6/IPv4 native functionality is carried to the end-point devices, as is the case with an end-to-end private network. The information generated in the core network devices while processing packets destined-to/sourced-from end-point devices, need to be percolated from underlay encapsulation to end customer specific payload. This is something which is NOT directed by any standards, and also NOT implemented by current deployment(s) of routers and switches.

Considering the fact that future beholds IPv6-only datacenter deployments, IPv6 PMTUD is one of the major casualties which can linger on forever, in case not dealt with as of now. Although this document intends to resolve PMTUD problem as a generic one across all underlay encapsulations.

Note that terms "ICMP(V6)" or "icmp(v4)" are used in the document with an intention to refer to both icmp and icmpv6, in case same context applies to both.

3.1. IPv6 PMTUD Issues

As mentioned in the [RFC1981], IPV6 PMTUD is based on the "Packet too big" icmpv6 error code, generated by the networking device which is capable of generating such messages on encountering packet paths which go over link with MTU size smaller than packet size.

There are problems getting this working when end-point device initiates a "Path MTU Discovery" to remote end-point device. It may lead to black-holing as per the current implementations.

The following bullets provides pointers to potential black holing of PMTUD packets,

(1) Vxlan Gateway(or ToR) MAY not set the DF bit in the outer IP header encapsulation.

(2) Vxlan Gateway(or TOR) is incapable of relaying icmp error "Fragmentation Needed and Don’t Fragment was Set", generated by IPv4 enabled core network device (underlay network), to IPv6 enabled end-point host/vm/server(source of the original packet).

The problems are discussed in detail in the following sub-sections.

3.1.1. Inaccurate MTU relayed to end hosts

Figure 1 depicts the topology referenced in the document for explaining the problem statement and the solution.

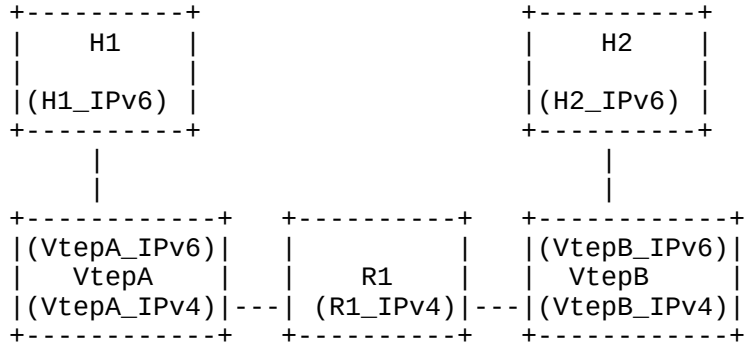


Figure 1. L3 Overlay

```
LEGEND:
MAC address : <Node_name>_MAC
IP address  : <Node_name>_IPv4
IPv6 address: <Node_name>_IPv6
<Node_name> : node names in the above topology are
              H1, VtepA, R1, VtepB, H2.
VtepA, VtepB: Vxlan gateways to core network
          R1: Intermediate router in underlay network
       H1,H2: End-point devices communicating withe each other
```

H1 and H2 are the end point hosts in different subnet connected over Vxlan Overlays in core network. The Vtep tunnel end-points MAY be ToR devices are christened as VtepA and VtepB, reachabile over an underlay IPv4 network. VtepA and VtepB are dual stack enabled and act as Vxlan gateways to connected hosts in this specific example. Link mtu between VtepA, R1 and VtepB is 1300 bytes, where as for the link between H1 and VtepA, H2 and VtepB, is 1500 bytes.

H1 sends out a packet obliging to 1500 bytes MTU packet size containment over the H1 and VtepA link. VtepA encapsulates the packet with (Vxlan + UDP) header and outer IP header corresponding to underlay reachability to destination tunnel end-point, that is VtepB, to reach out to H2.

If size of encapsulated packet to be send over the link VtepA-R1 exceeds the MTU (1300 bytes). IPv4 packet with (IP header + UDP

header + Vxlan header + Original L2 Packet from H1 containing the IPv6 Payload) SHOULD be fragmented. In case Vxlan gateway, VtepA, does not sets the DF-bit in the outer IP header, the packet gets fragmented, with the reassembly done at the egress gateway (VtepB).

The re-assembled packet is routed by VtepB to H2. This can potentially lead to inaccurate Path MTU calculation at H1. H1 assumes it to be 1500 bytes as no icmp error is received. This opens the door for fragment/reassembly and more cpu cycles on networking devices in core network.

3.1.2. Packet_Too_Big not-relayed to host

In figure 1, assume that link between VtepA and R1 is 1500 as the only change from the figure 1 topology. Hence the packet send by H1, leads to VtepA setting the DF-bit in the outer IP header(as part of Vxlan Encapsulation). When R1 receives the packet and the routing table lookup points to the outgoing link with mtu size R1_VtepB_MTU bytes, less than the packet size (1500 bytes). As DF-bit is set, R1 generates ICMPv4 error directed towards the src-ip (VtepA_IPv4). It encapsulates the inner PDU of the original packet. However, VtepA drops the icmp error packet and fails to relay it to H1. This leads to black-holing.

The above two sub-sections lay down potential problems for IPv6 Path MTU Discovery mechanism in an Overlay network. Although these problem are generic to any combination of underlay and overlay network types (IPv4 or IPv6), the use-case topology in this document is specific to IPv6 end-point devices connected over Vxlan network, wherein, the underlay is connected over IPv4 network, unless mentioned specifically.

4. Solution(s)

4.1. Discovery of end-to-end Path MTU

Since Vxlan Gateway (can be a ToR device) is the one, which encapsulates the Vxlan (or any other overlay) header onto the packet traversing through the overlay network and also decapsulates the overlay header for packets egressing out of same and heading towards the end devices, the solution becomes more apt to be installed on devices playing such role.

Firstly, It is a MUST that Vxlan gateways (VtepA, VtepB or ToR device) SHOULD set the DF-bit in Outer header encapsulation for client packets that are wrapped with vxlan, related encapsulation, for Path MTU Discovery. Thus ensuring that ICMP error packet is generated for packet size exceeding the link MTU in underlay network.

Secondly, it is MUST that Vxlan gateway devices translates the ICMP error "Destination Unreachable" with code 'Fragmentation Needed and Don't Fragment was Set', into a ICMPv6 error 'Packet too big' packet. This mandates that original packet carried in the icmp error message MUST carry information about the inner payload(original packet), and it is an IPv6 Packet, originated from the end-point device (H1 for VtepA in figure 1), connected to the Vxlan gateway over L3/L2 network.

Thirdly, it is MUST that Vxlan gateway devices translates the ICMPv6 error 'Packet too big' into a ICMP error 'Destination Unreachable' with code 'Fragmentation Needed and Don't Fragment was Set' packet. Successfully translation mandates that, original packet carried in the icmp error message gives information about the inner payload (original packet), and it is an IPv4 packet, which originated from the end-point device connected to gateway over L3/L2 network.

Fourthly, incase both, the client side network connected to Vxlan Gateway and the underlay network are same, that is, either both are ipv4 or both are ipv6, then icmp error code error translation is NOT required. Rest of the process to retrieve original packet is identical.

4.1.1. ICMP extensions, PMTUD on Vxlan

This solution leverages extensions in ICMP and ICMPv6 standards, [RFC4884], for the maximum size of the original packet that can be encapsulated in ICMP error message with code as "Fragmentation Required(icmp)" or "Packet too big(icmpv6)" respectively. As the host info is encapsulated in the inner payload, this requires additional bytes of data in icmp packet: (Outer IP Header + UDP Header + Vxlan + Inner L2 Header + Inner IPv6 SRC/DST IPs).

In case Vxlan core network is provisioned over IPv6 underlay, then similar extensions are applicable to icmpv6.

The processing of ICMP(V6) packet is extended from the current standards of 'non-delivery of ICMP(v6) packets to upper-layers on Vxlan gateways' to 'relaying it to the end-point devices'.

4.1.2. Packet Path Processing

Packet Path handling and processing is explained in this section. The assumptions are made with respect to network topology mentioned in Section 3.1.1. The packet format in each flow captures packet fields which are significant with respect to this solution. To understand the solution, the packet flow is explained which leads to

generation of ICMP or ICMPv6 error by intermediate node in underlay network.

IPv6 packet is sent by host H1 destined to host H2, both are in different IPv6 subnets.This packet is referred to as P1 in the document.

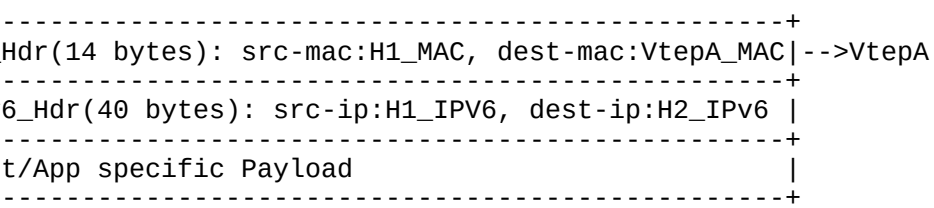


Figure 2a. Packet P1 sent by host H1 to host H2

VtepA re-writes the mac addresses in ’P1’ as part of Vxlan encapsulation. This encapsulation is referred as ’P2’ in the document.

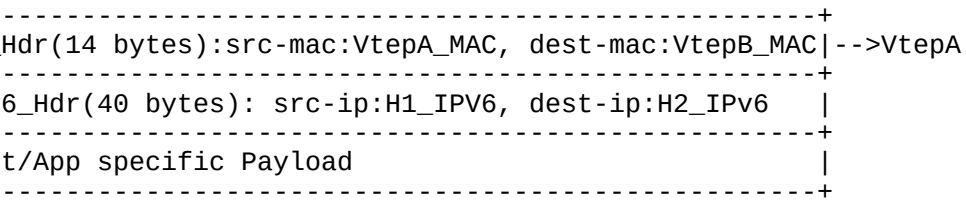


Figure 2b. Packet P1 re-written by VtepA

4.1.2.1. Packet Processing at Vxlan Gateway

Processing at VtepA, in packet path from H1 to H2.

(1) VtepA(Vxlan gateway) performs the Vxlan encapsulation over the packet received from H1, based on route lookup. The detail for encap are mentioned in [RFC7348].

(2) VtepA MUST set the DF-bit in the Outer IP header.

(3) Since the MTU of outgoing link is more than the packet, packet is sent out towards the underlay next hop, R1.

(4) P3 packets encapsulation is shown in figure 3. P3 may find a reference without outer header encapsulation [RFC7348] provides details of the vxlan encapsulation.

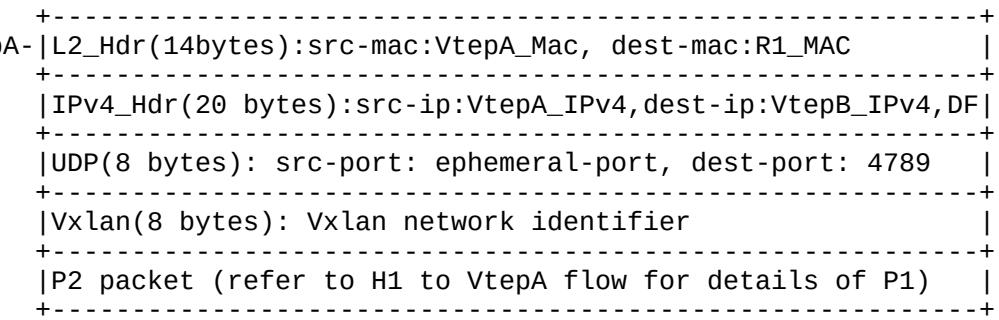


Figure 3. Vxlan Encap packet sent by Vxlan Gateway to core

4.1.2.2. Underlay Generates ICMP error

In case the underlay is ipv6 and not ipv4, icmpv6 error is generated.

Processing at R1:

(1) Packet Size (1500 bytes) is more than the outgoing link's mtu (1300 bytes) and DF-bit is set in the Outer IPv4 header added as part of Vxlan encapsulation at VtepA.

(2) R1 MUST generate icmp error message (Destination Unreachable) with error code (Fragmentation Needed and Don't Fragment was Set). For ease of solution description, mtu is assumed to be symmetric over the reverse path, hence reverse path mtu from R1 to VtepA is 1500 bytes. ICMP(v6) error message MUST include MTU of link between R1 and VtepB.

(3) In a nut shell, the ICMP PDU encapsulation SHOULD be performed as mentioned in [RFC4884] , [RFC4443]. These standards atleast ensure, that original packet carried in icmp error PDU captures enough bytes to include the inner packets IPv6 header atleast. The capture of application specific details depends on the size of the Optional header in the original packet (generated by H1 as in Figure 2b) and subsequent transport header. This helps Vxlan Gateway to trace(L3 reachability) the original packet generator (end-point device) atleast and translate icmp error generted by underlay into icmpv6 one and relay it to end-point device. The length field in ICMP PDU, include the maximum possible length permissible in reverse path MTU

For simplicity, not including the original packet header in the flow diagram in figure 4. ICMP PDU details are depicted in the follow up figure 5.

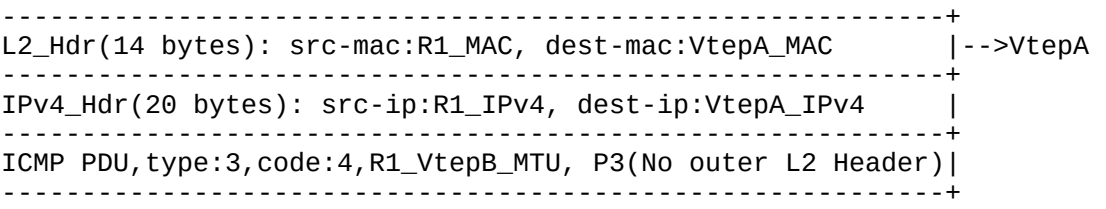
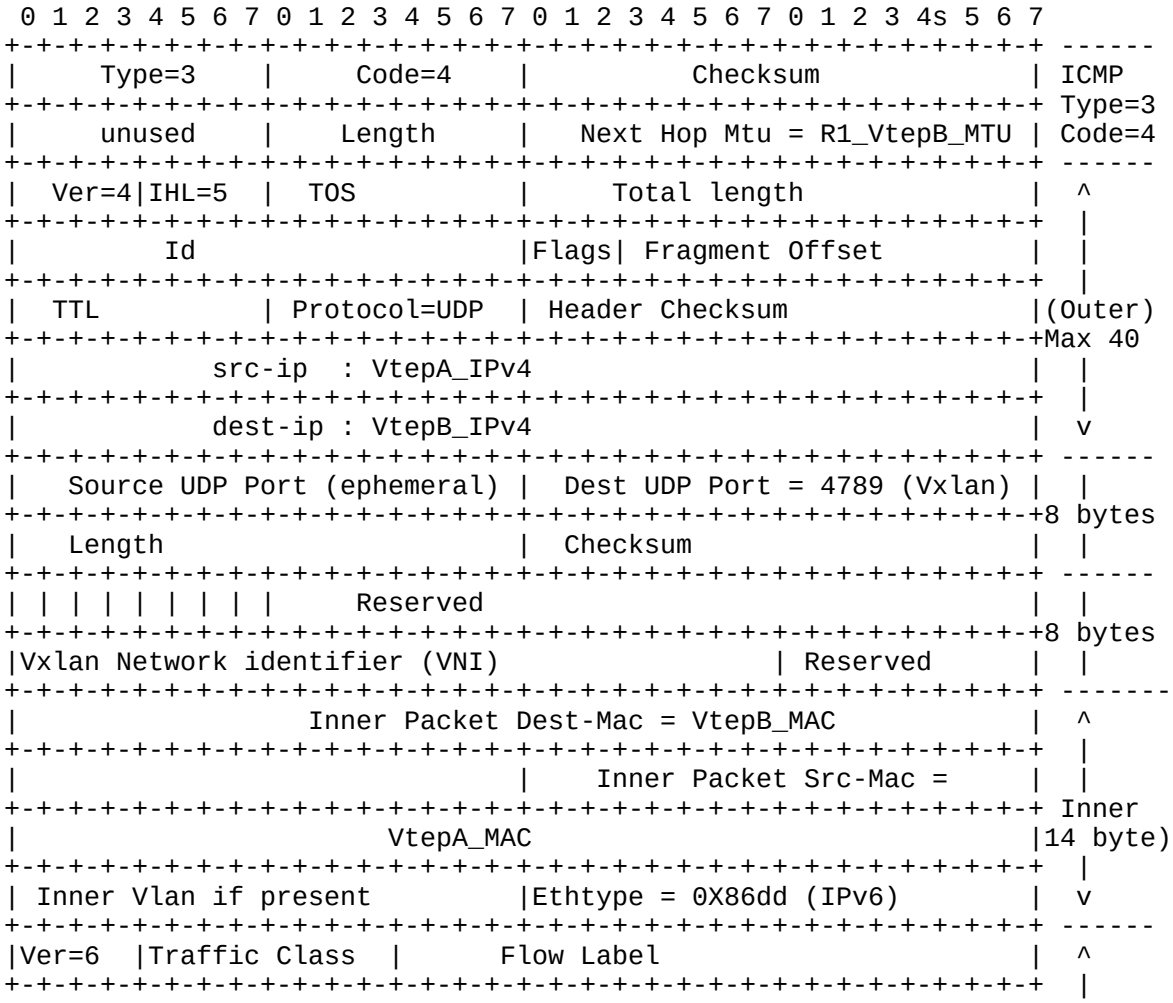


Figure 4. Flow diagram from R1 to VtepA

The details of ICMP PDU are in the following figure. Type ’3’ is "Destination Unreachable". Code ’4’ is "Fragmentation Needed and Don’t Fragment bit is set".



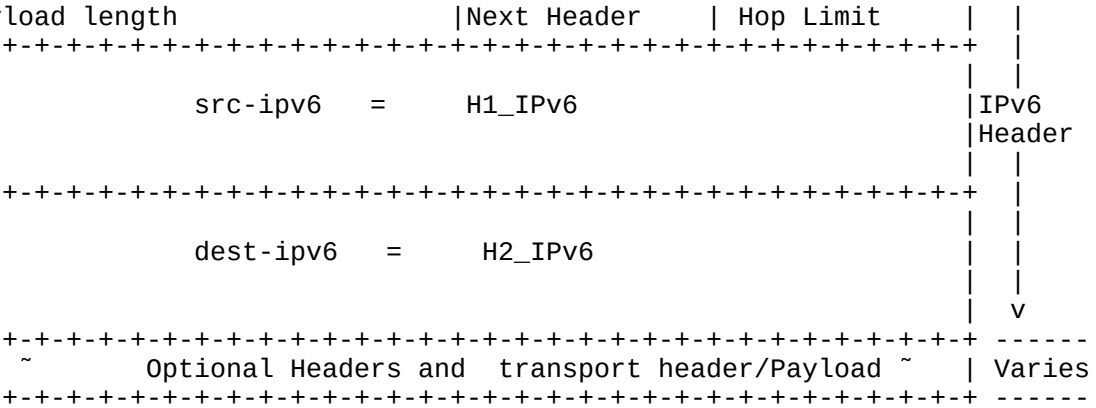


Figure 5. ICMP PDU Original Packet Capture in Detail

4.1.2.3. Relay ICMP(v6) Error to End Devices

This sub-section can also be generalized as: "handling of icmp errors, which are generated by underlay network in response to end-device packets, by Vxlan Gateway".

Processing at VtepA: Processing of icmp error message with code (Fragmentation Needed and Don't Fragment was Set):

(1) The icmp error is processed by Vxlan gateways as per the standards defined in [RFC1981] , [RFC4884] and [RFC4443] .

(2) If error code is (Fragmentation Needed and Don't Fragment was Set), it SHOULD perform further inspection of the original packet, P3(ethernet payload without its header) carried as data in ICMP PDU in extension to standards referred in previous bullet. The extension processing MUST be done prior to taking a decision to either drop the packet or deliver to upper-layer protocols.

(3) In extension to above, Vxlan gateway device SHOULD perform the vxlan decap as defined in [RFC7348], to arrive at the inner packet (P2, original packet with VtepA rewrite). The underlay encap is not carrying the layer-2 header in the icmp error packet. Once this processing is done, P2 is the packet which needs attention now, as it carries the credentials of actual host which should receive the relayed icmp packet.

(4) The layer-3 payload type SHOULD be verified using ethernet type field in ethernet header. In case it point to IPv6, src-ipv6 field should be picked up to check for reahability, as the icmp packet MUST be sent to original sender, that is, H1. In case H1

is reachable, ICMP packet SHOULD be constructed as mentioned in the following bullet.

(5) Now that P2 is out in the open, it's L2 header is decapsulated, and the leftover, in the figure 6, is run through the icmpv6 processing as mentioned in [RFC4443].

(6) It SHOULD generate ICMPv6 error message with type (Packet too big) destined to H1_IPv6, that is inner ipv6 packet's source ipv6 address. The mtu 'R1_VtepB_MTU' is copied from icmp error packet recieved from the underlay.

(7) The IPv6 header is constructed from original payload as shown in figure 5. The source ipv6 address is picked as local ipv6 address "VtepA_IPv6". The destination ipv6 address is set as the "src-ipv6" in original payload, H1_IPv6. The Next Header is set as "58" which denote ICMPv6. The derivation of ethernet header is based on next hop to mac address mapping as is performed in any L3 lookup. The follow up figure 9, shows the icmpv6 error packet sent out to node H1. H1 is the original IPv6 packet generator as mentioned in Figure 2b.

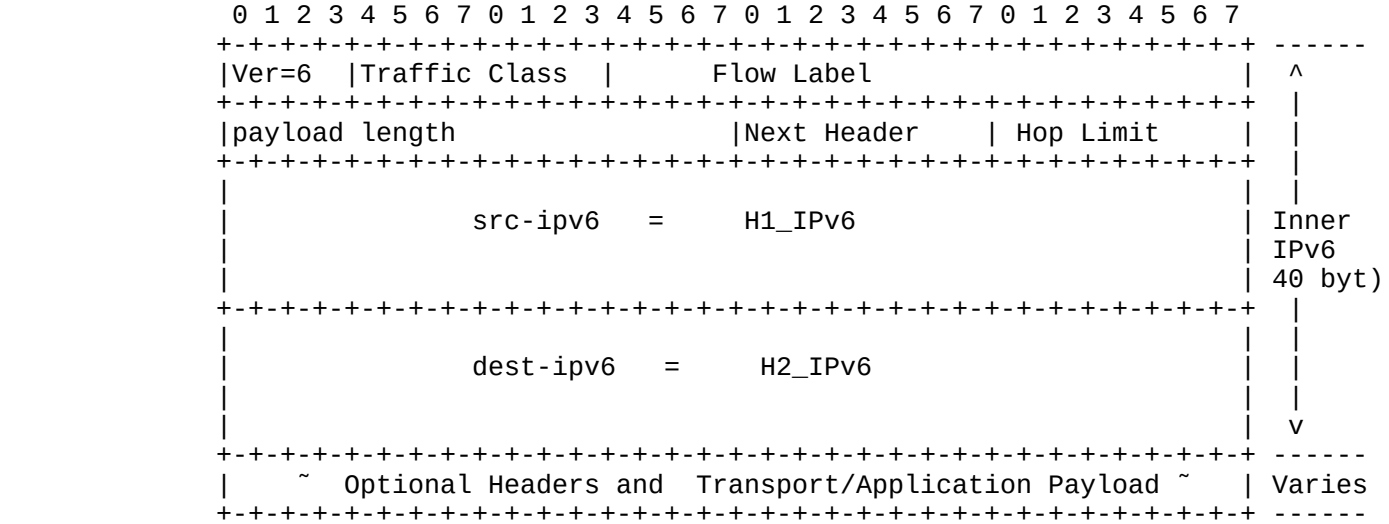


Figure 6. Original IPv6 Packet sent from H1 directed to H2

Figure 6 gives a typical IPv6 format sent by end-host, H1 towards H2 and encapsulated by Vxlan gateway, to translate the icmp error generated by underlay hop, R1, to the one understood in right context by H1.

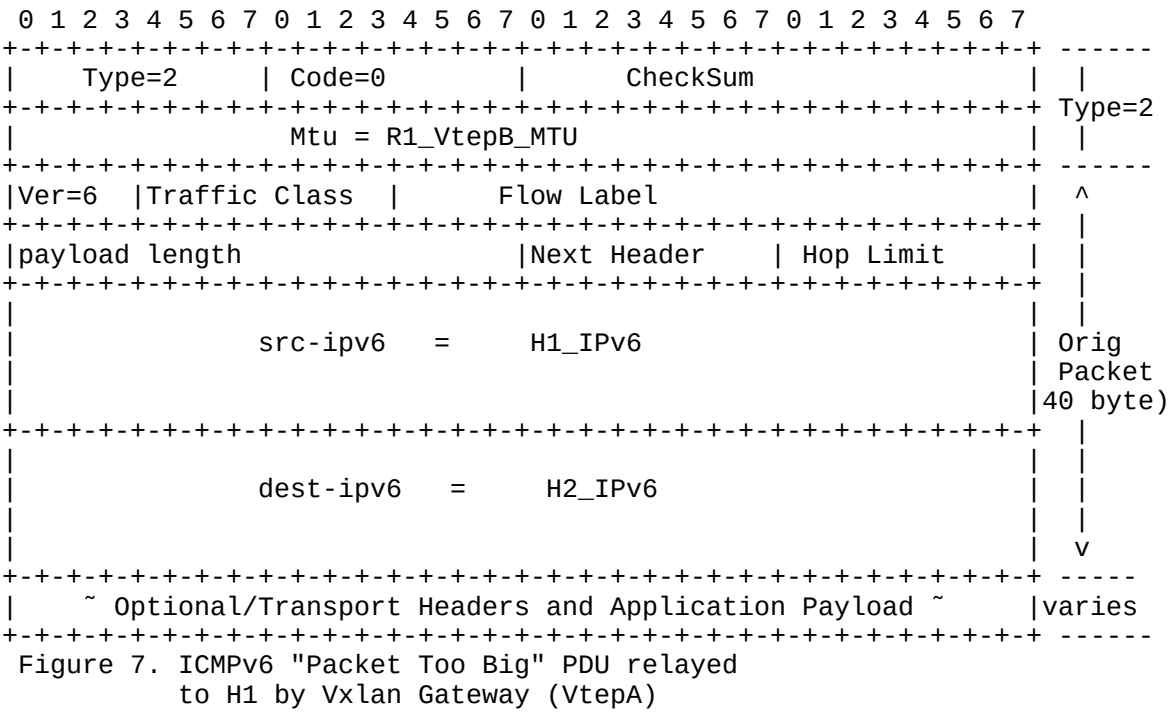


Figure 7. ICMPv6 "Packet Too Big" PDU relayed to H1 by Vxlan Gateway (VtepA)

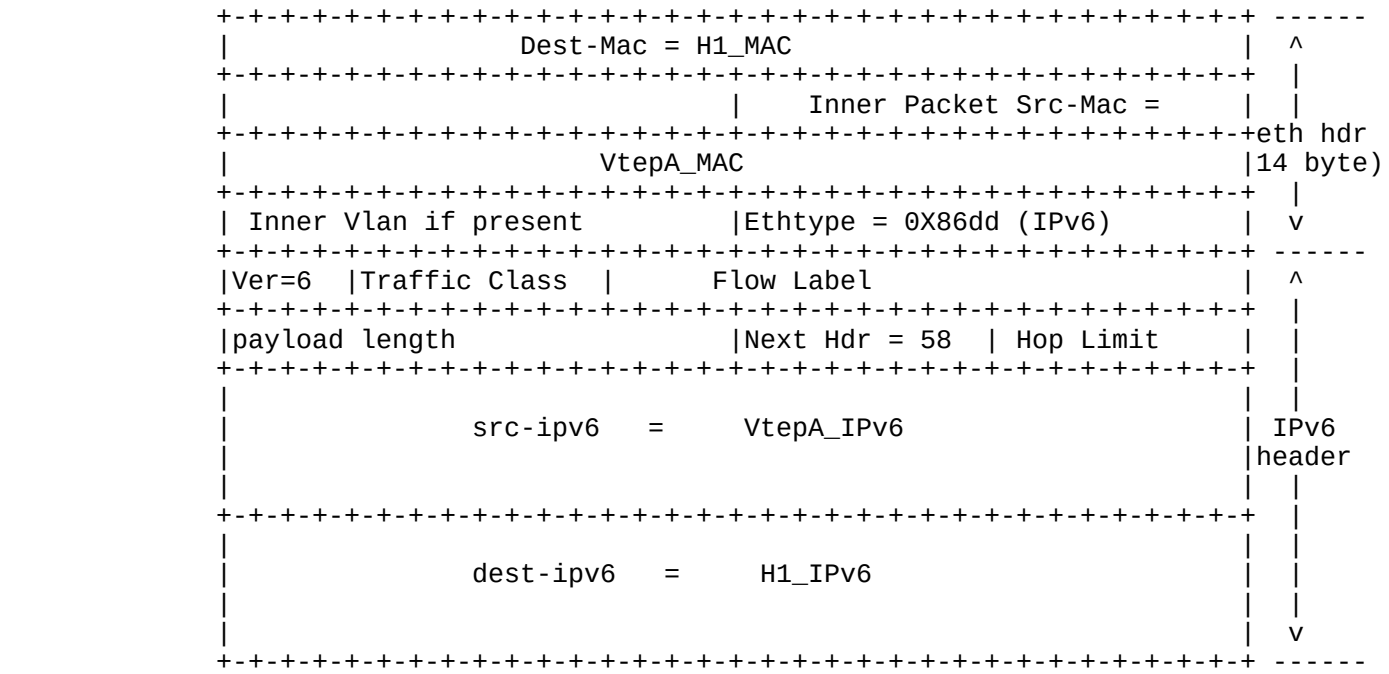


Figure 8. Ethernet and IPv6 encap for ICMPv6 PDU mentioned in figure 7

The translated icmp packet encapsulation looks similar to, figure 7 and figure 8 put together in reverse order. The flow diagram in figure 9 gives a concise form of "packet too big" icmpv6 error relayed by VtepA (Vxlan Gateway) towards H1 (end point device).

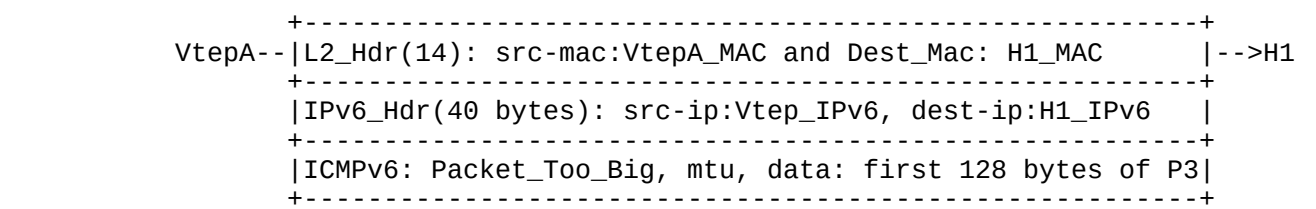


Figure 9. Flow diagram: VtepA to H1

There are few more potential flows worth mentioning in this section. These cases are related to, icmp error getting generated from, ingress Vxlan gateway (VtepA) and egress Vxlan gateway (VtepB) with respect to packet sent from H1 to H2. For ingress Vxlan gateway (VtepA) case, the legacy IPv6 PMTUD rules from [RFC4443] SHOULD be applied as no Vxlan encap is involved.

Where as, egress Vxlan gateway (VtepB) SHOULD send packet P3 (without L2 header) in the icmp data, even though mtu calculation MAY be done post vxlan decapsulation. That is when the outgoing link is identified as the one from VtepB to H2. It MAY buffer packet P3 prior to lookup based on inner packet (P2) credentials, so that P3 can be encapsulated in the icmp packet. This also ensures the packet format consistency, when accessed at the VtepA for translation before relaying it to H1.

4.1.3. ICMP(v6) Error Translation

This section specifically mentions about ICMP and ICMPv6 packet translation, generated in an underlay network to the one which is, understood by the end point device, with encapsulation aligning with the network-type(IPv4 and IPv6), end-point device and underlay is provisioned with. The last leg processing mentioned in previous sub-section is specific to the topology mentioned in Section 3.1.1. However, this subsection elaborates on all possible topology combination of underlay and end-device networks with respect to IPv4 or IPv6. The explanation provided in form of figures for error generated by underlay and the translated one relayed to the end-point device by Vxlan gateway.

(a) End-Point is IPv6 connected and Underlay is IPv4 provisioned.

(b) End-Point is IPv4 connected and Underlay is IPv6 provisioned.

(c) Both End-Point and Underlay are provisioned with IPv6.

(d) Both End-Point and Underlay are provisioned with IPv4.

4.1.3.1. End-Point is IPv6 connected and Underlay is IPv4 provisioned

This case is similar to the last leg processing described in Section 4.1.2 and does not needs any more description.

4.1.3.2. End-Point is IPv4 connected and Underlay is IPv6 provisioned

Topology drawn in figure 10, provides for the icmpv6 PDU encap generated by R1. H1_IPv4 and H2_IPv4 are in distinct ipv4 subnets. R1_IPv6 represents IPv6 addresses falling in both subnets connecting to VtepA and VtepB.

Another difference between an IPv4 and IPv6 underlay is that for IPv6 underlay there is no concept of DF-bit. The fragmentation can only be done at ingress. At all other underlay nodes "Packet too big" icmpv6 error is generated. Vxlan Gateway SHOULD ensure that fragmentation is avoided at Vxlan Gateway and icmp error is sent back

to H1. This procedure is applicable if and only if, original packet contains DF-bit set in it's IP header.

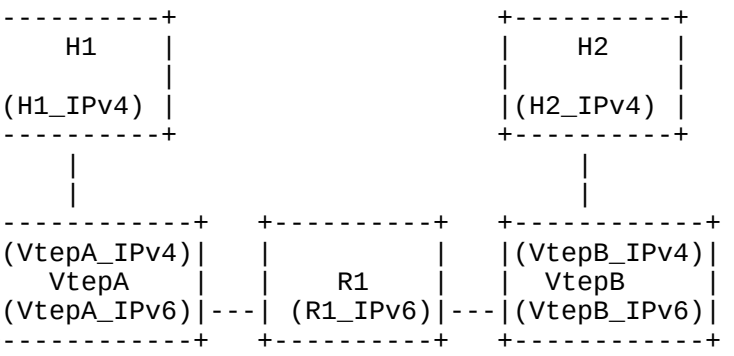
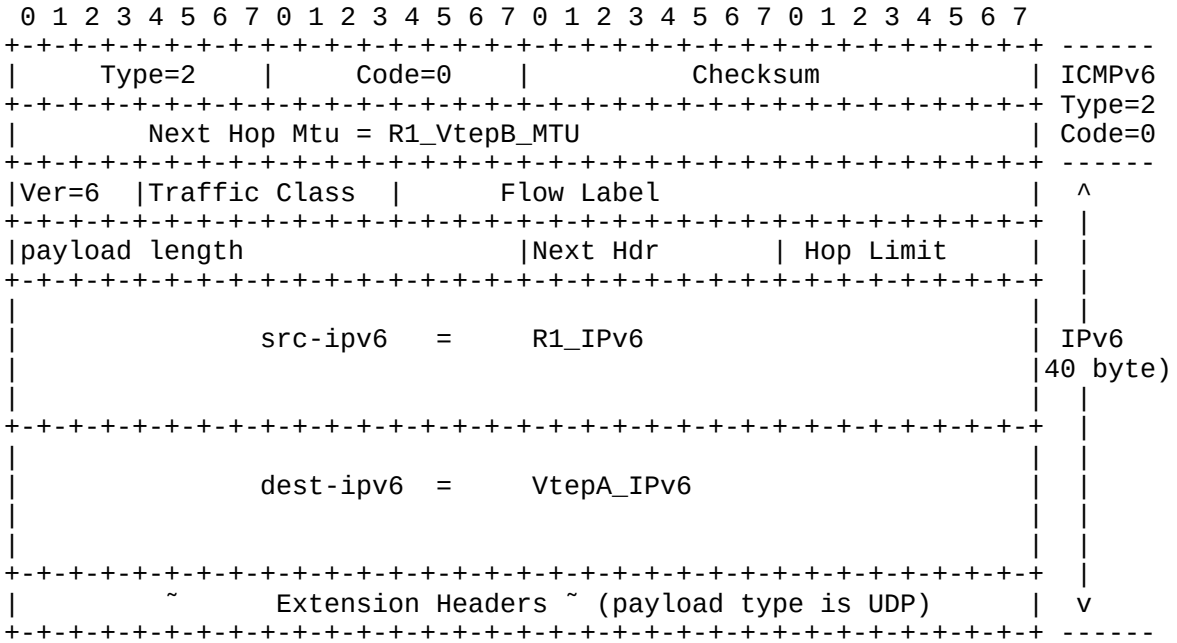


Figure 10. L3 Overlay

LEGEND:
MAC address : <Node_name>_MAC
IPv4 address: <Node_name>_IPv4
IPv6 address: <Node_name>_IPv6
<Node_name> : node names in the above topology are
H1, VtepA, R1, VtepB, H2.
VtepA, VtepB: Vxlan gateways to core network



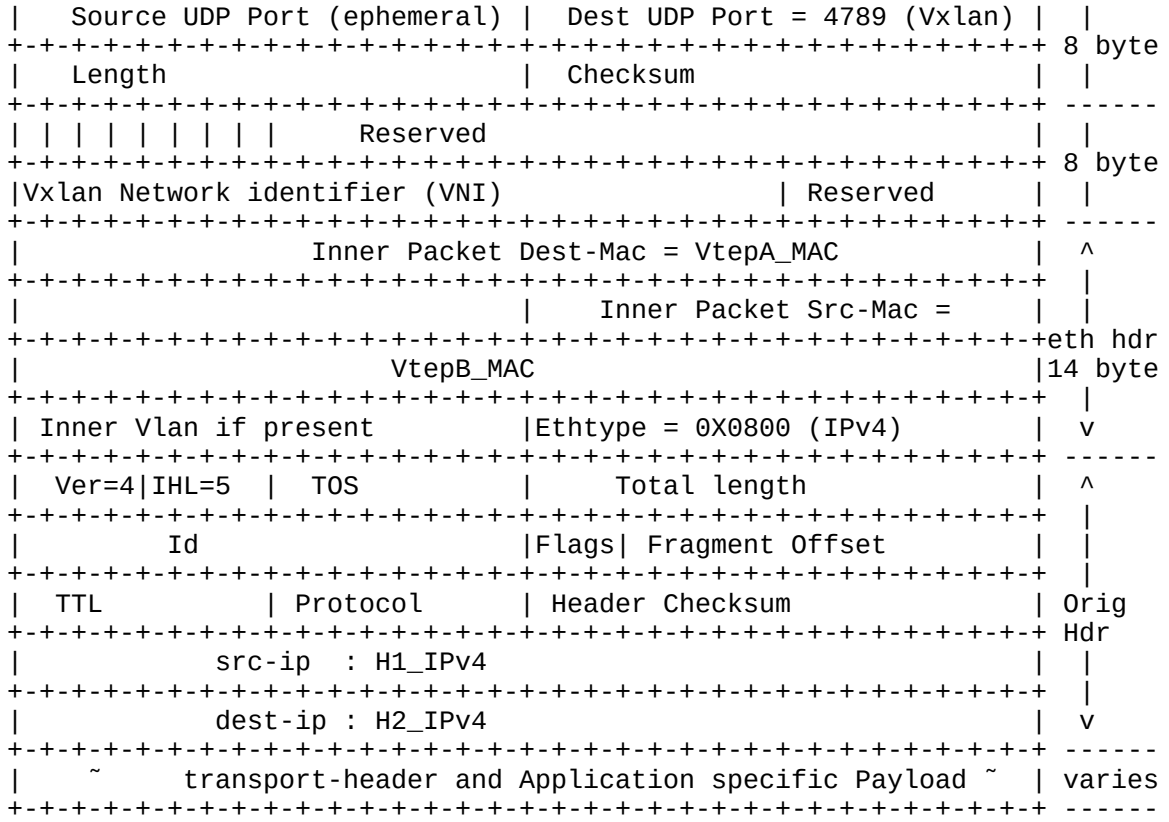


Figure 11. ICMPV6 PDU Sent by R1 to VtepA

R1 sends an icmpv6 error "Packet Too Big" directed towards VtepA. The icmpv6 PDU is shown in Figure 11. VtepA receives the packet with this icmpv6 PDU and translates it to icmp PDU with type "Destination Unreachable" and code "Fragmentation Needed" before relaying it to H1 over ipv4 network. Figure 12, reflects the relayed packet sent by VtepA to H1. All other references SHOULD be taken as it is from Section 4.1.2.

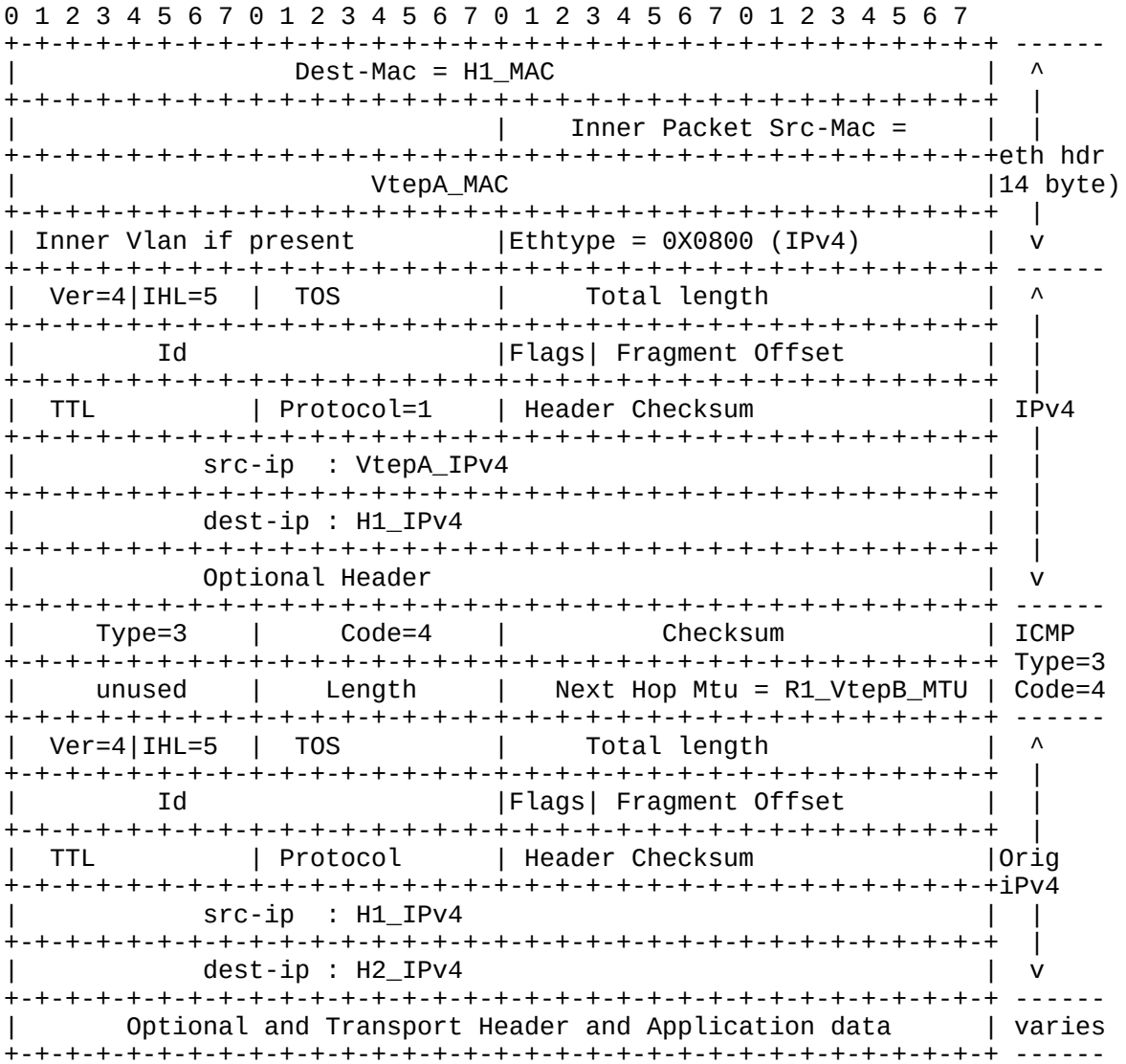


Figure 12. ICMPv4 error Packet relayed to end point Host, H1

4.1.3.3. Both End-Point and Underlay are provisioned with IPv6

Topology is mentioned in Figure 13 with minor changes along with the legend. Figure 14, outlines the icmpv6 PDU, encapsulation generated by R1. H1_IPv6 and H2_IPv6 in different ipv6 subnets. R1_IPv6 reflects both subnets connecting to VtepA and VtepB.

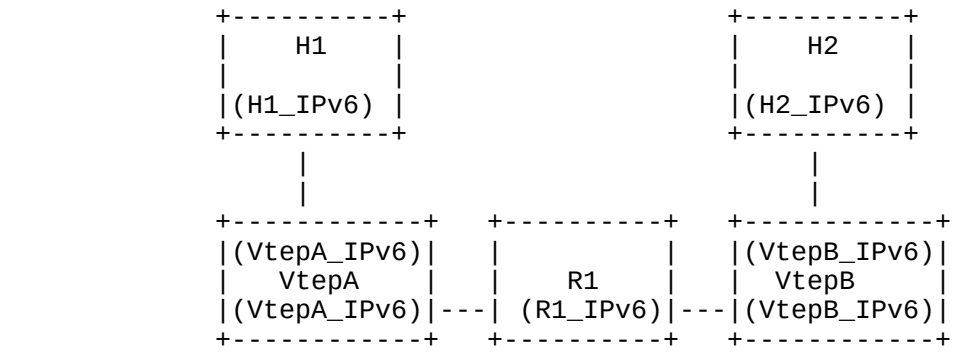
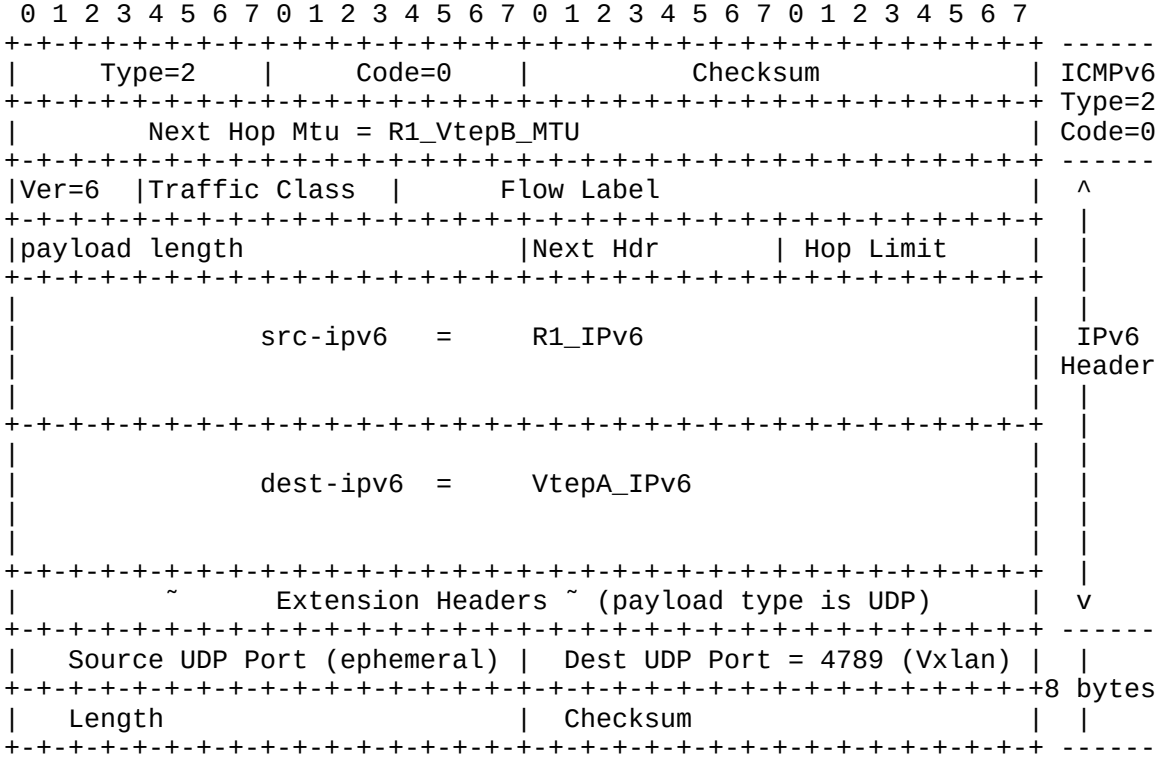


Figure 13. L3 Overlay

LEGEND:
MAC address : <Node_name>_MAC
IPv6 address: <Node_name>_IPv6
<Node_name> : node names in the above topology are
H1, VtepA, R1, VtepB, H2.
VtepA, VtepB: Vxlan gateways to core network



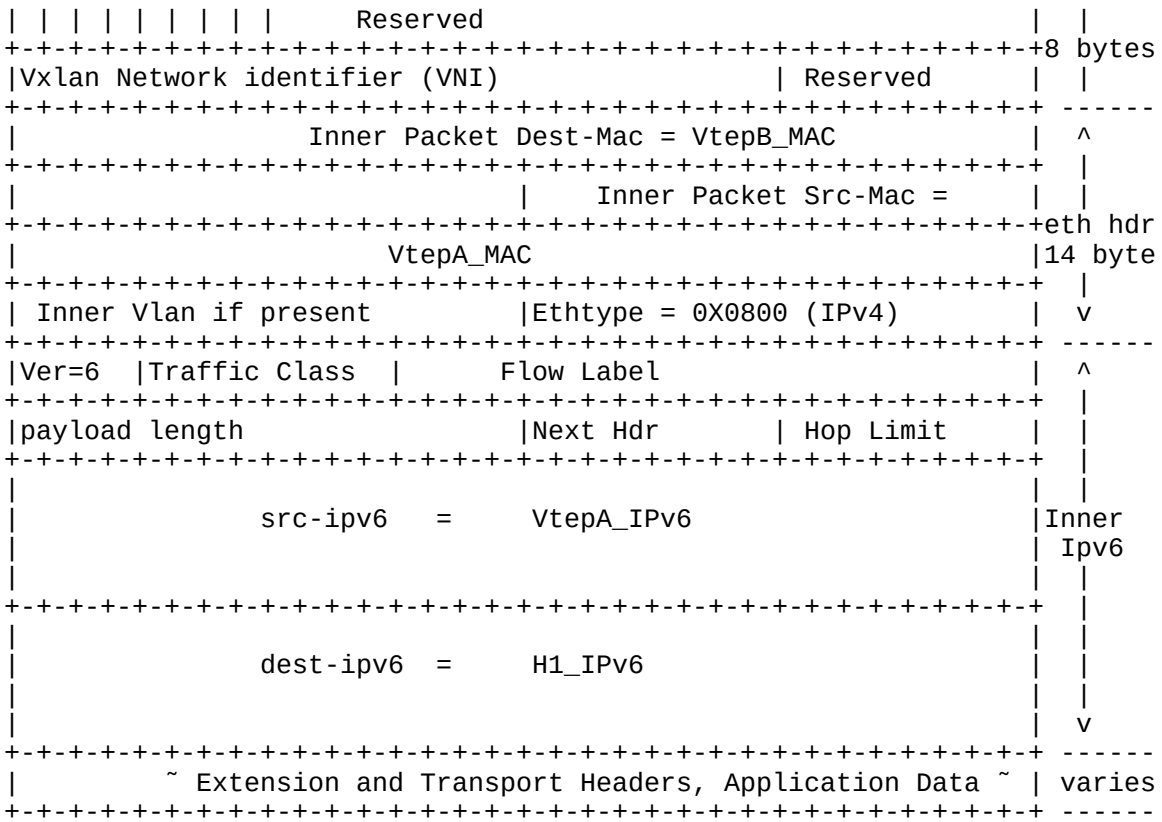


Figure 14. ICMPv6 PDU generated by Intermediate Hop, R1 in Vxlan Network

R1 sends an icmpv6 error "Packet Too Big" directed towards VtepA. The icmpv6 PDU is shown in Figure 14. VtepA receives the packet with this icmpv6 PDU and relays it to H1 without any translation as H1 is connected to VtepA over ipv6 network. All other references about original packet to be include in the icmpv6 PDU can be taken as it is from Section 4.1.2.

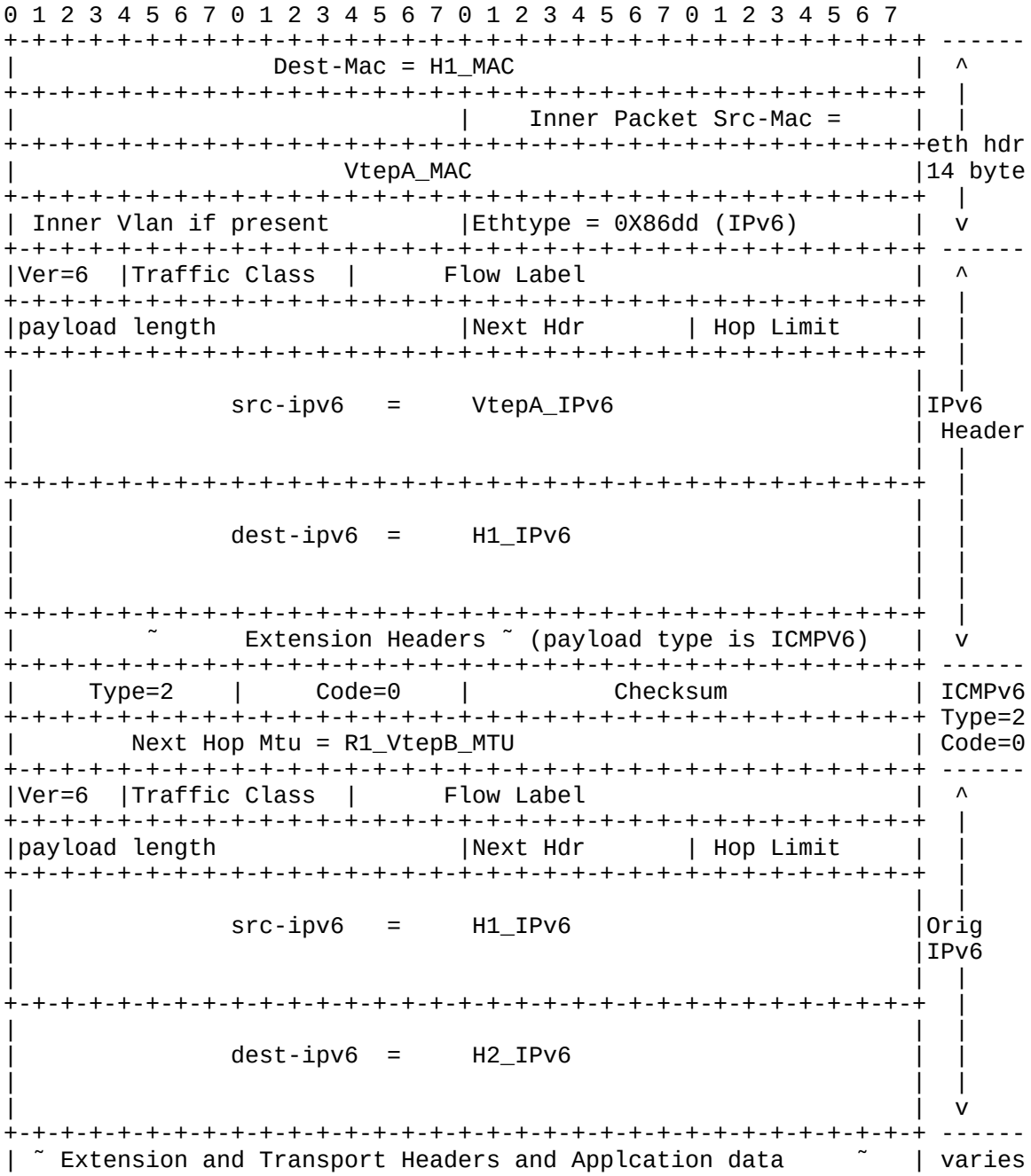


Figure 15. ICMPv6 error Complete Packet sent to H1 by VtepA

4.1.3.4. Both End-Point and Underlay are provisioned with IPv4

Topology is mentioned in figure 16, with minor changes along with the legend, figure 17, provides the icmp PDU encap generated by R1. H1_IPv4 and H2_IPv4 are in different ipv4 subnets.

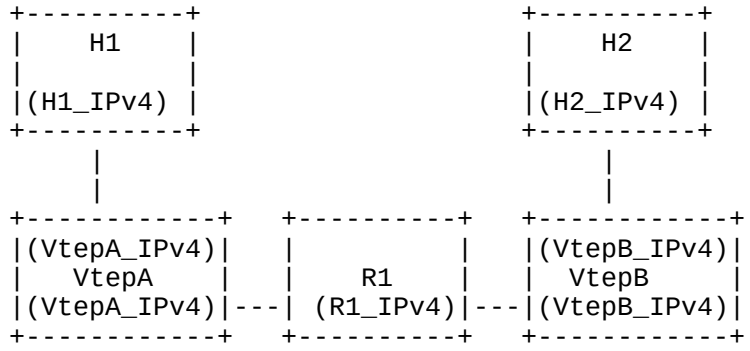


Figure 16. L3 Overlay

LEGEND:
MAC address : <Node_name>_MAC
IPv4 address: <Node_name>_IPv4
<Node_name> : node names in the above topology are H1, VtepA, R1, VtepB, H2.
VtepA, VtepB: Vxlan gateways to core network

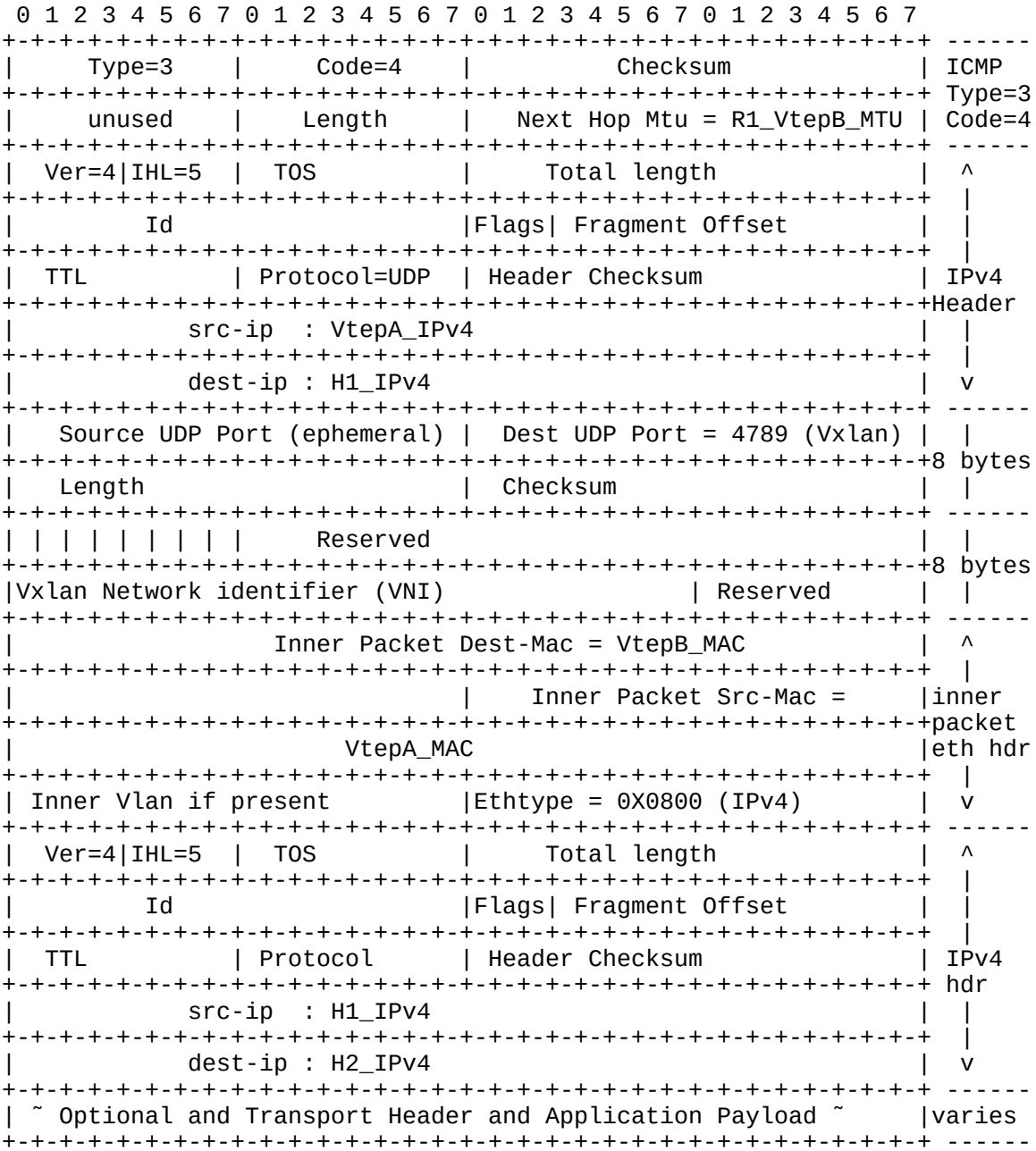


Figure 17. ICMP PDU generated by R1 towards VtepA

R1 sends an icmp error directed towards VtepA. The icmp PDU is shown in figure 17. VtepA receives the packet with this icmp PDU and relays it to H1 over ipv4 network. Figure 16, displays the packet sent by VtepA to H1. All other references can be taken as it is from Section 4.1.2.

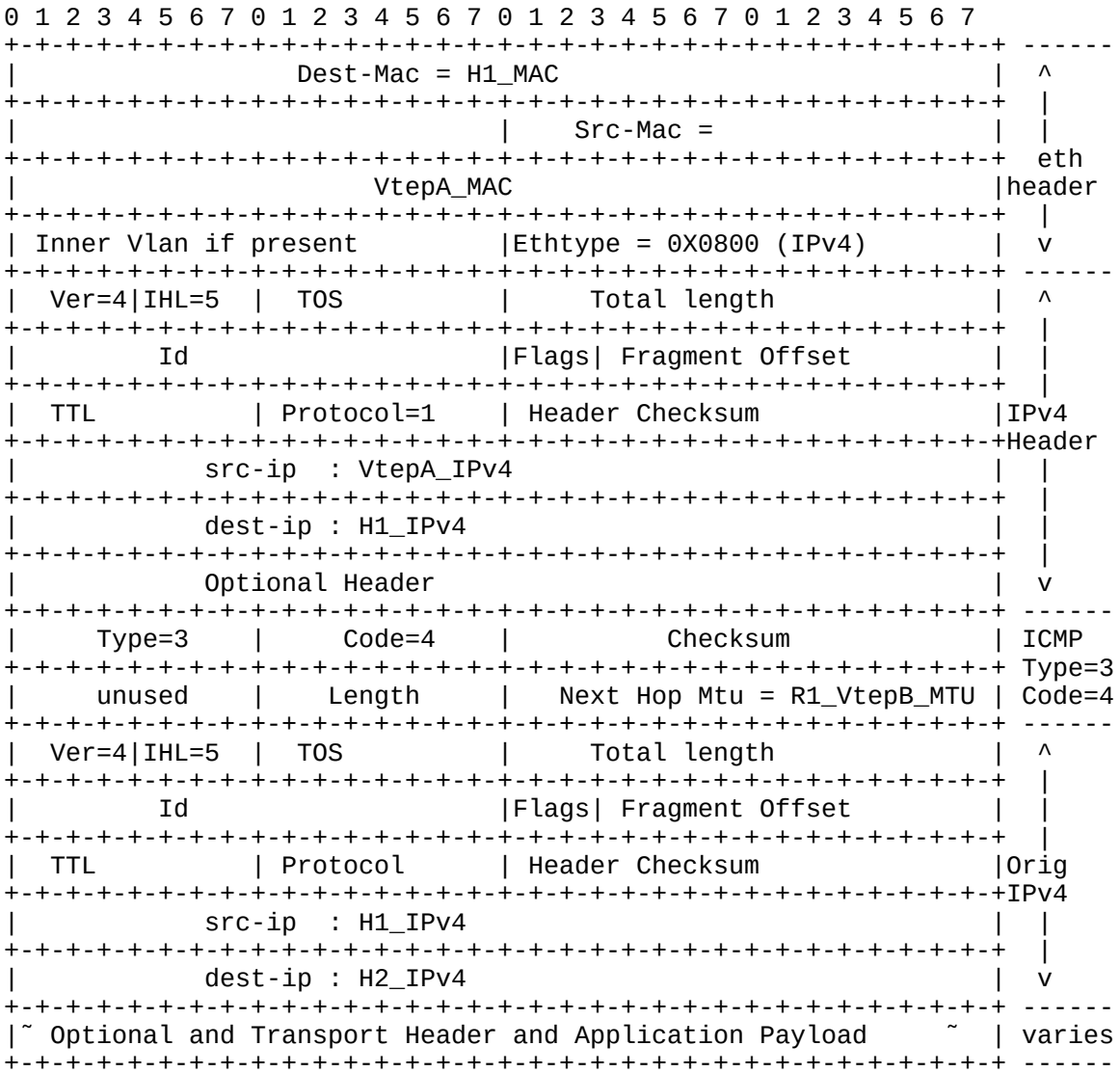


Figure 18. Complete ICMP error Packet sent to H1 by VtepA

5. Multicast and Anycast Considerations

Multicast solution is similar to one proposed in [RFC1981]. This SHOULD be applied at Vtep for cases of unknown unicast destinations.

There are no anycast considerations in this document, as the solution is based upon nodes deriving mtu values from the underlay network which should either have unicast or multicast reachability between them.

6. Ecmp Considerations

Ecmp considerations are driven by the packet sent by the end host application and the way it’s leveraged.

To ensure PMTUD is agnostic to ecmp paths in a Vxlan network, there are few more consideration. In Vxlan Gateway (can be ToR device), the route look-up is done based on attributes carried in packet generated by end point host. The packet generated can potentially be from a tcp based end host application (although should not be generalized).

Where as, for an intermediate node, (lets say, Spine node in Clos topology) in core network the look ups are based on Outer Encap (Vtep ip addresses and and UDP Header).

On another note, for an L2 gateway case, wherein Vxlan gateway (Vtep Node) bridges (and not routes) host packets destined to same subnet destination, MTU calculation SHOULD come into play only in the Spine devices.

7. Security Considerations

This document inherits all the security considerations discussed in [RFC1981] and [RFC1191].

8. IANA Considerations

TBD

9. Acknowledgements

Thanks to Vengada Prasad Govindan, Deepak Kumar, Matthew Bocci and Rohit Mendiratta for providing the inputs.

10. References

10.1. Normative References

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

10.2. Informative References

[I-D.draft-gross-geneve]
Gross, J., Sridhar, T., Garg, P., Wright, C., Ganga, G., Agarwal, P., Duda, C., Dutt, D., and J. Hudson, "Geneve: Generic Network Virtualization Encapsulation", draft-gross-geneve-02 (work in progress), Oct 2015.

[I-D.draft-ietf-nvo3-gue]
Herbert, T., Yong, L., and O. Zia, "Generic Protocol Extension for VXLAN", draft-ietf-nvo3-gue-03 (work in progress), Mar 2015.

[I-D.draft-ietf-nvo3-vxlan-gpe]
Quinn, P., Manur, R., Kreeger, L., Lewis, D., Maino, F., Smith, M., Agarwal, P., Yong, L., Xu, X., Elzur, U., and D. Melman, "Generic Protocol Extension for VXLAN", draft-ietf-nvo3-vxlan-gpe-02 (work in progress), May 2015.

[I-D.nordmark-nvo3-transcending-traceroute]
Nordmark, E., Appanna, C., and A. Lo, "Layer-Transcending Traceroute for Overlay Networks like VXLAN", draft-nordmark-nvo3-transcending-traceroute-02 (work in progress), March 2015.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", RFC 4884, April 2007.

[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.

[RFC7637] Yang, S. and M. Garg, "Network Virtualization Using Generic Routing Encapsulation", RFC 7637, Sep 2015.

Authors' Addresses

Saumya Dikshit
Cisco Systems
Cessna Business Park
Bangalore, Karnataka 560 087
India

Email: sadikshi@cisco.com

A Sujeet Nayak
Cisco Systems
Cessna Business Park
Bangalore, Karnataka 560 087
India

Email: sua@cisco.com