NVO3 WG
Internet-Draft
Intended status: Standards Track
Expires: March 1, 2019

Fangwei Hu
Ran Chen
ZTE Corporation
Mallik Mahalingam
Springpath
Qiang Zu
Ericsson
S. Davari
yahoo
Xufeng Liu
Volta Networks

August 28, 2018

YANG Data Model for VxLAN Protocol
draft-chen-nvo3-vxlan-yang-07.txt

Abstract

This document defines a YANG data model for VxLAN protocol.

Status of This Memo

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

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

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

This Internet-Draft will expire on March 1, 2019.

Copyright Notice

Table of Contents

1. Introduction

YANG[RFC6020]is a data definition language that was introduced to define the contents of a conceptual data store that allows networked devices to be managed using NETCONF [RFC6241]. This document defines a YANG data model for the configuration of VxLAN protocol [RFC7348].

2. Terminology

3. Requirements Language

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.

4. YANG Data Model for VxLAN Configuration

4.1. VxLAN Multicast IP Address

The vxlan-multicast-ip is used to configure the IP multicast group, which the VxLAN VNI of the VTEP is mapping to. Both the IPv4 and IPv6 address family are supported in this document.

4.2. VxLAN Access Type

There are several access types supported for VxLAN:

- vlan-1:1: the vxlan access type is VLAN, and each VxLAN is only mapping to one VLAN.
- vlan- n:1: the vxlan access type is VLAN, and each VxLAN is mapped to several VLANs.
- L3-interface: the VxLAN access type is layer 3 interface.
- mac: the VxLAN access type is MAC address.
- vlan-l2-interface: the VxLAN access type is VLAN plus Layer 2 interface.

4.3. Inner VLAN Tag Handling Mode

There are two handling modes for the inner VLAN tag: discard-inner-vlan mode and no-discard-inner-vlan mode. If the VTEP interface works in the discard-inner-vlan mode, the VxLAN is only mapped to one VLAN. The inner VLAN tag will be stripped when encapsulating the VxLAN frame. On the decapsulation side, if VTEP receives the VxLAN frame with inner VLAN tag, it will discard the frame in this work mode. If the VTEP receives the VxLAN frame without VLAN tag, it will fill in the VLAN tag based on the VxLAN and VLAN mapping entry.

If the VTEP interface works in the no-discard-inner-vlan mode, the VxLAN could be mapped to several VLANs. The inner VLAN tag will not stripped when encapsulating the VxLAN frame in the VxLAN encapsulation side. On the decapsulation side, if VTEP receives the VxLAN frame, it will strip the VxLAN header, and keep the VLAN frame.

5. Design Tree of VxLAN YANG Data Model

```
module: ietf-vxlan
    +--rw vxlan
    |  +--rw global-enable?                empty
    |  +--rw vxlan-instance* [vxlan-id]
    |     +--rw vxlan-id                   vxlan-id
    |     +--rw description?               string
    |     +--rw unknow-unicast-drop?       enumeration
    |     +--rw filter-vrrp?               enumeration
    |     +--rw (vxlan-access-types)? {vxlan-access-types}?
    |     |  +--:(access-type-vlan)
    |     |  |  +--rw access-type-vlan?          access-type-vlan
    |     |  |  +--rw access-vlan-list* [vlan-id]
```

```
|     |  |     +--rw vlan-id    vlan
|     |  +--:(access-type-mac)
|     |  |  +--rw access-type-mac?          empty
|     |  |  +--rw mac                       yang:mac-address
|     |  +--:(access-type-l2interface)
|     |  |  +--rw access-type-l2interface?  empty
|     |  |  +--rw vlan-id                   vlan
|     |  |  +--rw interface-name            if:interface-ref
|     |  +--:(access-type-l3interface)
|     |     +--rw access-type-l3interface?  empty
|     |     +--rw map-l3interface* [interface-name]
|     |        +--rw interface-name    if:interface-ref
|     +--rw vtep-instances* [vtep-id]
|     |  +--rw vtep-id                     uint32
|     |  +--rw vtep-name?                  string
|     |  +--rw source-interface?           if:interface-ref
|     |  +--rw multicast-ip                inet:ip-address
|     |  +--rw mtu?                        uint32 {mtu}?
|     |  +--rw inner-vlan-handling-mode?   inner-vlan-handling-mode
|     |  +--rw bind-vxlan-id* [vxlan-id]
|     |     +--rw vxlan-id    vxlan-id
|     +--rw static-vxlan-tunnel* [vxlan-tunnel-id]
|     |  +--rw vxlan-tunnel-id      uint32
|     |  +--rw vxlan-tunnel-name?   string
|     |  +--rw address-family* [af]
|     |     +--rw af                       address-family-type
|     |     +--rw tunnel-source-ip?        inet:ip-address
|     |     +--rw tunnel-destination-ip?   inet:ip-address
|     |     +--rw bind-vxlan-id* [vxlan-id]
|     |        +--rw vxlan-id    vxlan-id
|     +--rw redundancy-group-binds
|        +--rw redundancy-group-bind* [vxlan-id redundancy-group]
|           +--rw vxlan-id            uint32
|           +--rw redundancy-group    uint32
|
+--ro vxlan-state
   +--ro vxlan
      +--ro vxlan-tunnels
         +--ro vxlan-tunnel* [local-ip remote-ip]
            +--ro local-ip            inet:ip-address
            +--ro remote-ip           inet:ip-address
            +--ro static-tunnel-id?   uint32
            +--ro evpn-tunnel-id?     uint32
            +--ro statistics
               +--ro tunnel-statistics
               |  +--ro in-bytes?      string
               |  +--ro out-bytes?     string
               |  +--ro in-packets?    string
               |  +--ro out-packets?   string
```

```
                  +--ro tunnel-vni-statistics
                     +--ro tunnel-vni-statistic* [vxlan-id]
                        +--ro vxlan-id       uint32
                        +--ro in-bytes?      string
                        +--ro out-bytes?     string
                        +--ro in-packets?    string
                        +--ro out-packets?   string

  augment /evpn:evpn/evpn:evpn-instances/evpn:evpn-instance/evpn:bgp-parameters/
evpn:common:
    +--rw bgp-parameters
       +--rw common
          +--rw rd-rt* [route-distinguisher]
             +--rw route-distinguisher    string
             +--rw vpn-target* [rt-value]
                +--rw rt-value    string
                +--rw rt-type     bgp-rt-type
```

6. VxLAN YANG Model

```
<CODE BEGINS> file "ietf-vxlan@2018-08-29.yang"
module ietf-vxlan {
    namespace "urn:ietf:params:xml:ns:yang:ietf-vxlan";
    prefix "vxlan";

     import ietf-evpn {
       prefix "evpn";
     }

    import ietf-interfaces {
      prefix "if";
     }

    import ietf-inet-types {
       prefix "inet";
     }

    import ietf-yang-types {
       prefix yang;
     }

    organization
      "IETF NVO3(Network Virtualization Overlays) Working Group";

      contact
       "
```

```
         WG List:  <mailto:nvo3@ietf.org>

         WG Chair: Matthew Bocci
                   <mailto:matthew.bocci@alcatel-lucent.com>

         WG Chair: Benson Schliesser
                   <mailto:bensons@queuefull.net>

         Editor:   Fangwei Hu
                   <mailto:hu.fangwei@zte.com.cn>

         Editor:   Ran Chen
                   <mailto:chen.ran@zte.com.cn>

         Editor:   Mallik Mahalingam
                   <mailto:mallik_mahalingam@yahoo.com>

         Editor:   Zu Qiang
                   <mailto:Zu.Qiang@Ericsson.com>
        ";

    description
       "The YANG module defines a generic configuration
        model for VxLAN protocol";

    revision 2018-08-29 {
      description "Fixs some type error.";
      reference
         "draft-chen-nvo3-vxlan-yang-07";
    }

    revision 2018-01-03 {
      description "Changes the yang data model according to the NMDA style.";
      reference
         "draft-chen-nvo3-vxlan-yang-06";
    }

    revision 2017-06-29 {
      description "no changes.";
      reference
         "draft-chen-nvo3-vxlan-yang-05";
    }

    revision 2016-12-08 {
      description "updated the vxlan yang model based on the comments from IETF
97th meeting,"
       +"augmenting EVPN data model, adding access type configuration and MTU co
nfiguration.";
      reference
         "draft-chen-nvo3-vxlan-yang-04";
```

```
    }

    revision 2016-06-02 {
       description
         "03 revision. Update the YANG data model based on thec comments of IETF
 95th meeting.";
       reference
         "draft-chen-nvo3-vxlan-yang-03";
     }

    revision 2015-12-01 {
       description
         "02 revision.";
       reference
         "draft-chen-nvo3-vxlan-yang-02";
     }

    revision 2015-10-12 {
       description
         "01 revision.";
       reference
         "draft-chen-nvo3-vxlan-yang-01";
     }

    revision 2015-05-05 {
      description "Initial revision";
      reference
         "draft-chen-nvo3-vxlan-yang-00";
    }

    /* Feature  */

    feature vxlan-access-types {
       description
         "Support configuration vxlan access types.";
     }

    feature mtu {
        description
         "Support configuration vxlan MTU value.";
    }

    feature evpn-bgp-params {
       description "Support EVPN BGP parameter.";
      }

    /* Typedefs  */

    typedef vlan {
```

```
      type uint16 {
        range 1..4094;
      }
      description
      "Typedef for VLAN";
    }

    typedef vxlan-id {
      type uint32;
      description
        "Typedef for VxLAN ID.";
    }
      typedef access-type-vlan {
      type enumeration {
        enum access-type-vlan1to1 {
          description
            "Access type is VLAN 1:1.";
        }
        enum access-type-vlan1ton {
          description
            "Access type is VLAN 1:n.";
        }

      }
      default access-type-vlan1to1 ;
      description
        "VxLAN access type is VLAN.";
    }

    typedef access-type-mac {
       type empty ;
       description
         "VxLAN access type is MAC.";
    }


    typedef inner-vlan-handling-mode {
       type enumeration {
         enum discard-inner-vlan {
              description
                "Discard inner-VLAN.";
         }
         enum no-discard-inner-vlan {
              description
                "No discard inner-VLAN.";
         }
       }
       default discard-inner-vlan ;
```

```
       description
        "Typedef for inner-vlan-handling-mode";
    }

    typedef address-family-type {
       type enumeration {
         enum ipv4 {
           description
             "IPv4";
         }
         enum ipv6 {
           description
             "IPv6";
         }
       }
       description
         "Typedef for address family type.";
     }

    /* Configuration Data */

      container vxlan{
        leaf global-enable {
          type empty ;
          description 'VXLAN global enble.';
        }

        list vxlan-instance {
           key vxlan-id ;
           leaf vxlan-id {
             type vxlan-id;
             description "VxLAN ID.";
           }

           leaf description {
            type string {
              length 0..64 {
                description 'VXLAN instance description information.';
              }
            }
            description 'The description information of VXLAN instance.';
           }

           leaf unknow-unicast-drop {
            type enumeration {
              enum enable {
                value 1 ;
                description 'Unknown unicast drop enable.';
```

```
            }
            enum disable {
              value 2 ;
              description 'Unknown unicast drop disable.';
            }
          }
          default enable ;
          description 'Unknow unicast drop configuration of VXLAN instance.';
        }

        leaf filter-vrrp {
          type enumeration {
            enum enable {
              value 1 ;
              description 'VRRP packets filter.';
            }
            enum disable {
              value 2 ;
              description 'VRRP packets not filter.';
            }
          }
          default enable ;
          description 'VRRP packets filter configuration of VXLAN instance.';
        }

        choice vxlan-access-types {
          if-feature  vxlan-access-types;
           case access-type-vlan {

             leaf access-type-vlan {
               type access-type-vlan;

               description
                  "Access type is VLAN.";
             }

             list access-vlan-list {
               key vlan-id ;
               leaf vlan-id {
                 type vlan;
                 description
                   "VLAN ID.";
               }
               description
                 "VLAN ID list." ;
             }
               description
                 "VxLAN access type choice is VLAN.";
```

```
            }

            case access-type-mac {
              leaf access-type-mac {
                type empty ;
                description
                  "Access type is MAC.";
              }

              leaf mac {
                  type yang:mac-address ;
                  mandatory true ;
                  description
                    "MAC Address.";
                }
              description
                "VxLAN access type choice is MAC Address.";
            }

            case access-type-l2interface {
              leaf access-type-l2interface {
                type empty ;
                description
                  "VXLAN map layer two interface.";
              }

              leaf vlan-id {
                  type vlan;
                  mandatory true ;
                  description
                    "VLAN ID.";
               }

              leaf interface-name {
                  type if:interface-ref;
                  mandatory true ;
                  description
                    "Layer two interface name.";
              }
              description
                "VxLAN access type choice is layer two interface.";
            }

            case access-type-l3interface {
              leaf access-type-l3interface {
                type empty ;
                description
                  "Access type of VxLAN is layer three interface.";
```

```
                }

                list map-l3interface {
                  key interface-name ;
                  leaf interface-name {
                    type if:interface-ref;
                    description
                      "Layer three interface name.";
                  }
                  description
                     "Layer three interface list.";
                }
                description
                  "VxLAN access type choice is layer three interface.";
              }
              description
                 "VxLAN access type choice.";
            }

        list vtep-instances {
            key vtep-id ;
            leaf vtep-id {
               type uint32;
               description
               "VTEP ID.";
            }

            leaf vtep-name{
               type string;
               description
                "VTEP instance name.";
            }

            leaf source-interface {
               type if:interface-ref;
               description
                     "Source interface name.";
            }

            leaf multicast-ip {
                type inet:ip-address;
                mandatory true ;
                description
                  "VxLAN multicast IP address.";
            }

            leaf mtu {
              if-feature mtu;
```

```
              type uint32;
              description "vxlan mtu";
            }

            leaf inner-vlan-handling-mode {
                type inner-vlan-handling-mode;
                description
                   "The inner vlan tag handling mode.";
            }

            list bind-vxlan-id {
                key vxlan-id;
                leaf vxlan-id {
                  type vxlan-id;
                  description
                       "VxLAN ID.";
                }
                description
                  "VxLAN ID list for the VTEP.";
            }
            description
                "VTEP instance.";
        }

        list static-vxlan-tunnel{
            key vxlan-tunnel-id;
            leaf vxlan-tunnel-id {
               type uint32;
               description
                     "Static VxLAN tunnel ID.";
            }

            leaf vxlan-tunnel-name {
               type string;
               description
               "Name of the static VxLAN tunnel.";
            }

            list address-family {
               key "af";
               leaf af {
                 type address-family-type;
                 description
                 "Address family type value.";
               }

               leaf tunnel-source-ip {
                 type inet:ip-address;
```

```
                description
                "Source IP address for the static VxLAN tunnel";
              }

              leaf tunnel-destination-ip {
                 type inet:ip-address;
                 description
                  "Destination IP address for the static VxLAN tunnel";
              }

              list bind-vxlan-id {
                 key vxlan-id;
                 leaf vxlan-id {
                   type vxlan-id;
                   description
                      "VxLAN ID.";
                 }
                 description
                    "VxLAN ID list for the VTEP.";
              }

               description
                  "Per-af params.";
            }
           description
             "Configure the static VxLAN tunnel";
       }

       container redundancy-group-binds {
          list redundancy-group-bind {
             key 'vxlan-id redundancy-group';
             leaf vxlan-id {
               type uint32 {
                 range 1..16777215 {
                   description 'The value of VXLAN,it must between 1 to 1677721
5.';
                 }
               }
               description 'VXLAN ID binding by redundancy group.';
             }

             leaf redundancy-group {
               type uint32 {
                  range 1..4294967293 {
                     description 'The value of redundancy group,it must between
 1 to'
                          + ' 4294967293.';
                  }
               }
               description 'Redundancy group ID.';
```

```
            }
            description ’Redundancy group bind table.’;
          }
          description ’Redundancy group bind table.’;
        }
        description "vxlan instance list";
      }
      description
        "VxLAN configure model.";
    }


    augment "/evpn:evpn/evpn:evpn-instances/evpn:evpn-instance"
         +"/evpn:bgp-parameters/evpn:common" {

      uses evpn:bgp-parameters-grp {
        if-feature evpn-bgp-params;
       }
       description "EVPN configuration";
      }

       /* Operational  data */
    container vxlan-state{
      config false;
      container vxlan {
        container vxlan-tunnels {
          list vxlan-tunnel {
             key ’local-ip remote-ip’;
             leaf local-ip {
               type inet:ip-address;
               description ’Local IP of tunnel.’;
             }

             leaf remote-ip {
               type inet:ip-address;
               description ’Remote IP of tunnel.’;
             }

             leaf static-tunnel-id {
               type uint32 ;
               description ’Static tunnel ID.’;
             }

             leaf evpn-tunnel-id {
               type uint32 ;
               description ’EVPN tunnel ID.’;
             }
```

```
            container statistics {
               container tunnel-statistics {
                 leaf in-bytes {
                    type string {
                      length 0..24 ;
                    }
                    description 'Total bytes received.';
                 }

                 leaf out-bytes {
                   type string {
                     length 0..24 ;
                   }
                   description 'Total bytes sent.';
                 }

                 leaf in-packets {
                   type string {
                      length 0..24;
                   }
                   description 'Total packets received.';
                 }

                 leaf out-packets {
                   type string {
                     length 0..24 ;
                   }
                   description 'Total packets sent.';
                 }
                 description 'Total tunnel statistics.';
               }

               container tunnel-vni-statistics {
                list tunnel-vni-statistic {
                 key vxlan-id ;
                 leaf vxlan-id {
                   type uint32 ;
                   description 'The VXLAN in tunnel.';
                 }

                 leaf in-bytes {
                   type string {
                     length 1..24 ;
                   }
                   description 'Total bytes received.';
                 }

                 leaf out-bytes {
```

```
                  type string {
                    length 1..24 ;
                  }
                  description 'Total bytes sent.';
                }

                leaf in-packets {
                  type string {
                    length 1..24 ;
                  }
                  description 'Total packets received.';
                }

                leaf out-packets {

                  type string {
                    length 1..24 ;
                  }
                  description 'Total packets sent.';
                }
                description 'Statistics in VXLAN tunnel.';
              }
              description 'Statistics in VXLAN tunnel.';
            }
             description 'Tunnel statistics.' ;
          }
            description 'VXLAN tunnel info.';
        }
          description 'VXLAN tunnel Info.';
      }
      description 'Information of VXLAN state.';
    }
    description 'Information of VXLAN state.';
  }
 }
<CODE ENDS>
```

7. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH)[RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations.

The vulnerable "config true" parameters and subtree are the following:

ietf-vxlan/global-enable: this subtree specifies VxLAN enable switch. Modify the configuration can cause the VxLAN disable.

ietf-vxlan/vxlan-instance/static-vxlan-tunnel: this subtree specifies static VxLAN tunnel configuration. Modify the configuration can cause static VxLAN tunnel disconnection.

Unauthorized access to any of these lists can adversely affect the security of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations, and other problems.

8. Acknowledgements

9. IANA Considerations

This document registers three URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested to be made.

urn:ietf:params:xml:ns:yang:ietf-vxlan.

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers three YANG modules in the YANG Module Names registry [RFC6020].

```
name:        ietf-vxlan
namespace:   urn:ietf:params:xml:ns:yang:ietf-vxlan
prefix:      vxlan
reference:   RFC XXXX
```

10. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

[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>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Authors' Addresses

Fangwei Hu
ZTE Corporation
No.889 Bibo Rd
Shanghai 201203
China

Phone: +86 21 68896273
Email: hu.fangwei@zte.com.cn

Ran Chen
ZTE Corporation
No.50 Software Avenue,Yuhuatai District
Nanjing, Jiangsu Province 210012
China

Phone: +86 025 88014636
Email: chen.ran@zte.com.cn

Mallik Mahalingam
Springpath
640 W. California Ave, Suite #110
Sunnyvale, CA 94086
USA

Email: mallik_mahalingam@yahoo.com

Zu Qiang
Ericsson
8400, boul. Decarie
Ville Mont-Royal, QC
Canada

Email: Zu.Qiang@Ericsson.com

Davari Shahram
yahoo

Email: davarish@yahoo.com

Xufeng Liu
Volta Networks
USA

Email: xufeng.liu.ietf@gmail.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.

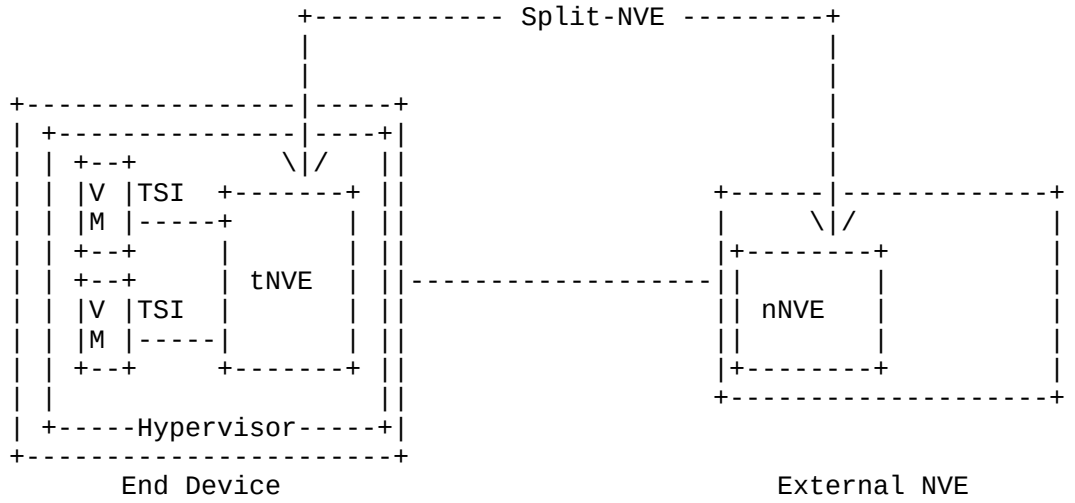

Figure 1 Split-NVE structure

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.

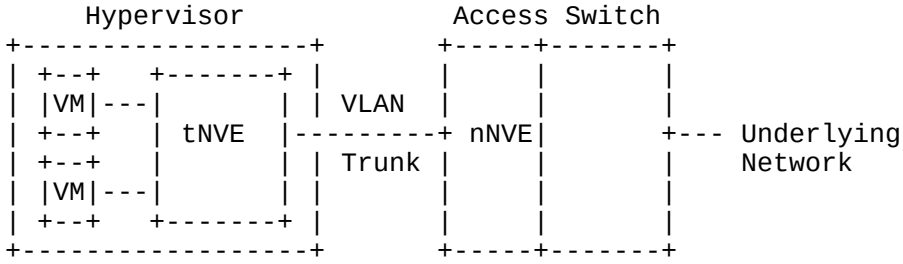

Figure 2 Hypervisor with an External NVE

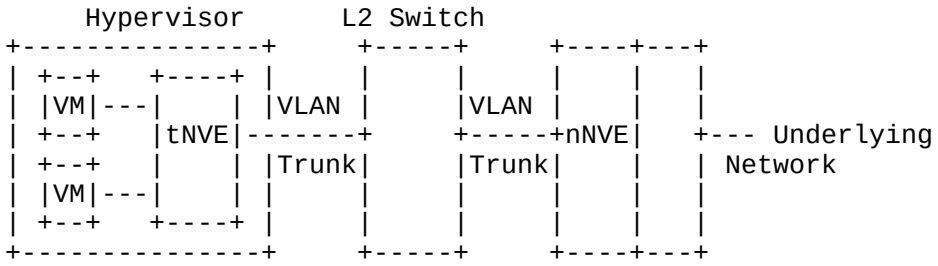

Figure 3 Hypervisor with an External NVE
connected through an Ethernet Access Switch

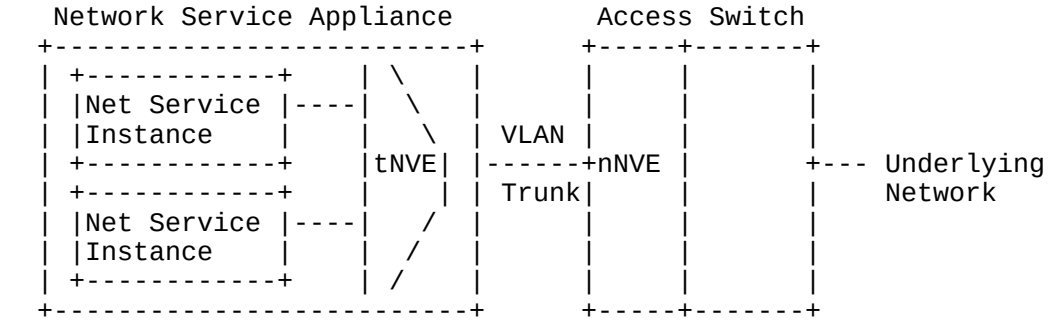

Figure 4 Physical Network Service Appliance with an External NVE

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.

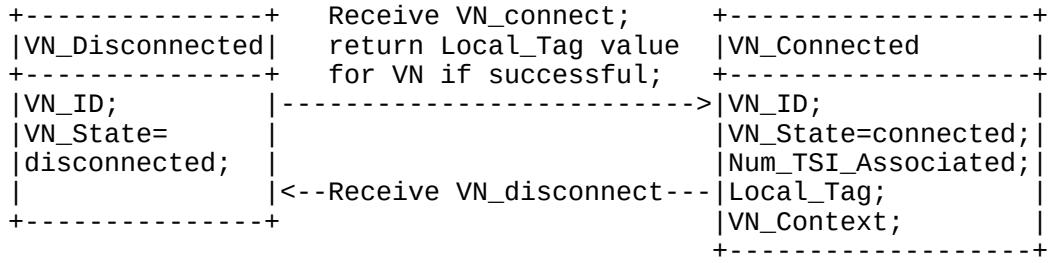

Figure 5. State Transition Example of a VAP Instance on an External NVE

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.

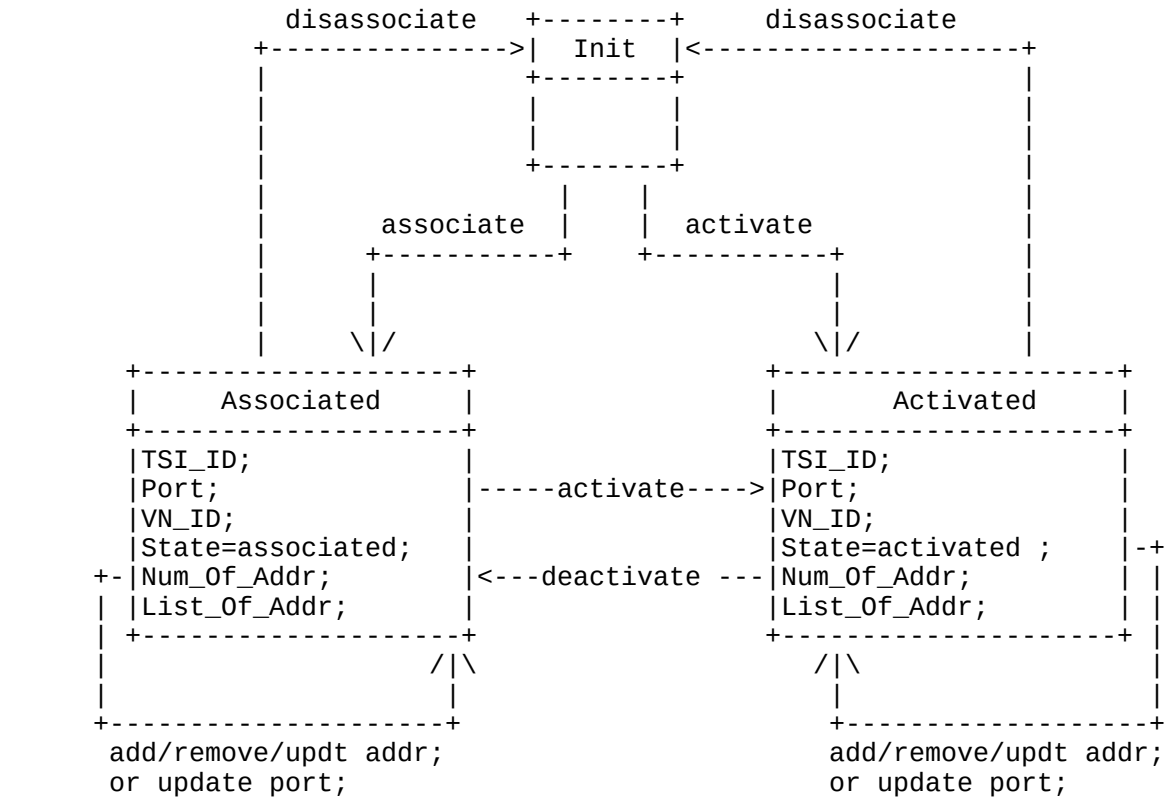

Figure 6 State Transition Example of a TSI Instance on an External NVE

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.

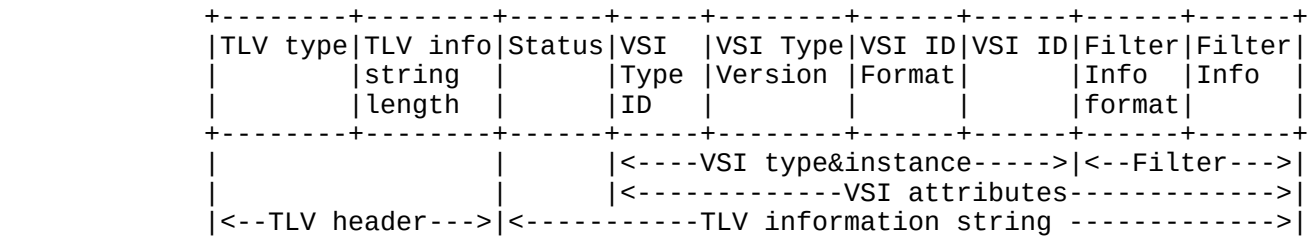

Figure A.1: VDP association TLV

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

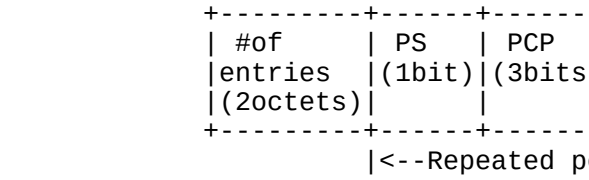

Figure A.2 VID Filter Info format

2. MAC/VID Filter Info format

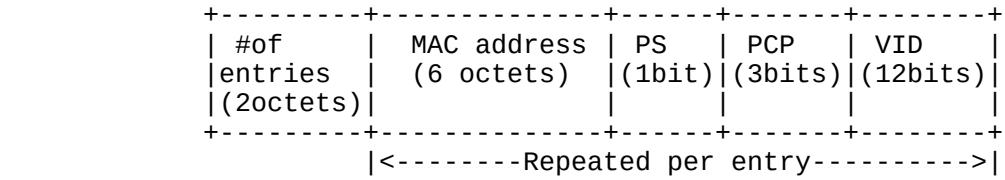

Figure A.3 MAC/VID filter format

3. GroupID/VID Filter Info format

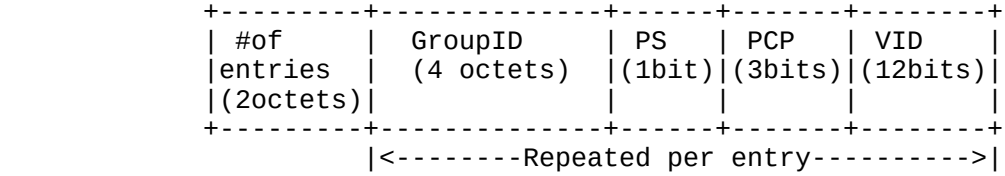

Figure A.4 GroupID/VID filter format

4. GroupID/MAC/VID Filter Info format

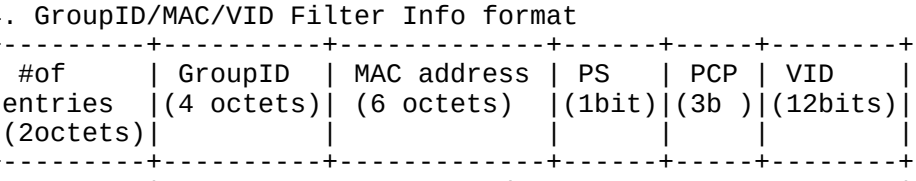

Figure A.5 GroupID/MAC/VID filter format

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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 11, 2017

G. Mirsky
ZTE Corp.
N. Kumar
D. Kumar
Cisco Systems, Inc.
M. Chen
Y. Li
Huawei Technologies
D. Dolson
Sandvine

March 10, 2017

Echo Request and Echo Reply for Overlay Networks
draft-ooamdt-rtgwg-demand-cc-cv-03

Abstract

This document defines Overlay Echo Request and Echo Reply that enable on-demand Continuity Check, Connectivity Verification among other operations in overlay networks.

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 September 11, 2017.

Copyright Notice

Table of Contents

1. Introduction

Operations, Administration, and Maintenance (OAM) toolset provides methods for fault management and performance monitoring in each layer of the network, in order to improve their ability to support services with guaranteed and strict Service Level Agreements (SLAs) while reducing operational costs.

1.1. Conventions used in this document

1.1.1. Terminology

Term "Overlay OAM" used in this document interchangeably with longer version "set of OAM protocols, methods and tools for Overlay networks". And "Overlay ping" is used interchangeably with longer version Overlay Echo Request/Reply.

CC Continuity Check

CV Connectivity Verification

ECMP Equal Cost Multipath

FM Fault Management

Geneve Generic Network Virtualization Encapsulation

GUE Generic UDP Encapsulation

MPLS Multiprotocol Label Switching

NVO3 Network Virtualization Overlays

OAM Operations, Administration, and Maintenance

SFC Service Function Chaining

SFP Service Function Path

VXLAN Virtual eXtensible Local Area Network

VXLAN-GPE Generic Protocol Extension for VXLAN

1.1.2. Requirements Language

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 [RFC2119].

2. On-demand Continuity Check and Connectivity Verification

2.1. Requirements Towards On-demand CC/CV OAM

Availability, not as performance metric, is understood as ability to reach the node, i.e. the fact that path between ingress and egress does exist. Such OAM mechanism also referred as Continuity Check (CC). Connectivity Verification (CV) extends Continuity Check functionality in order to provide confirmation that the desired source is connected to the desired sink.

Echo Request/Reply OAM mechanism enables detection of the loss of continuity defect, its localization and collection information in order to discover root cause. These are requirements considered:

REQ#1: MUST support fault localization of Loss of Continuity check at Overlay layer.

REQ#2: MAY support fault localization of Loss of Continuity check at transport layer.

REQ#3: MUST support tracing path in overlay network through the overlay nodes.

REQ#4: MAY support tracing path in underlay network connecting overlay border nodes.

REQ#5: MAY support verification of the mapping between its data plane state and client layer services.

REQ#6: MUST have the ability to discover and exercise equal cost multipath (ECMP) paths in its underlay network.

REQ#7: MUST be able to trigger on-demand FM with responses being directed towards initiator of such proxy request.

2.2. Proposed Solution

The format of the Echo Request/Echo Reply control packet is to support ping and traceroute functionality in overlay networks Figure 1 resembles the format of MPLS LSP Ping [RFC4379] with some exceptions.

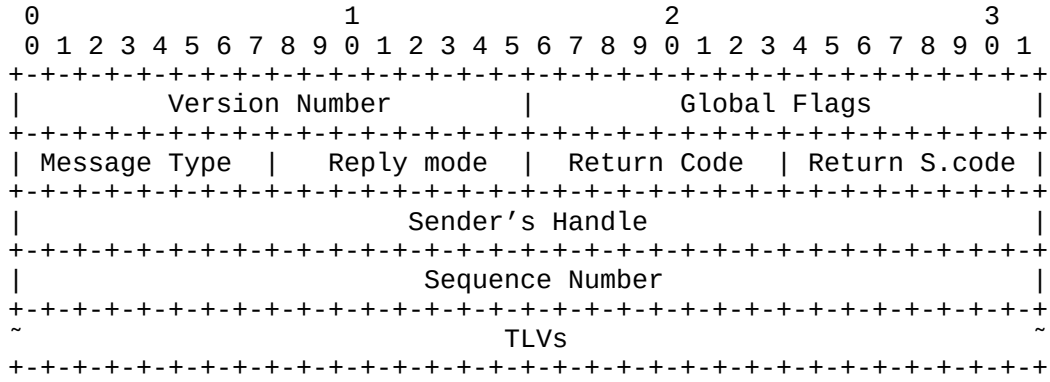

Figure 1: Overlay OAM Ping format

The interpretation of the fields is as following:

The Version reflects the current version. The version number is to be incremented whenever a change is made that affects the

ability of an implementation to correctly parse or process control packet.

The Global Flags is a bit vector field

The Message Type filed reflects the type of the packet. Value TBA2 identifies Echo Request and TBA3 - Echo Reply

The Reply Mode defines the type of the return path requested by the sender of the Echo Request.

Return Codes and Subcodes can be used to inform the sender about result of processing its request.

The Sender's Handle is filled in by the sender, and returned unchanged by the receiver in the Echo Reply.

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

TLVs (Type-Length-Value tuples) have the two octets long Type field, two octets long Length field that is length of the Value field in octets.

2.3. Overlay Echo Request Transmission

Overlay Echo Request control packet MUST use the appropriate encapsulation of the monitored overlay network. Overlay network encapsulation MUST identify Echo Request as OAM packet. Overlay encapsulation uses different methods to identify OAM payload [I-D.ietf-nvo3-vxlan-gpe], [I-D.ietf-nvo3-gue], [I-D.ietf-nvo3-geneve], [I-D.ietf-sfc-nsh],[I-D.ietf-bier-mpls-encapsulation]. Overlay network's header MUST be immediately followed by the Overlay OAM Header [I-D.ooamdt-rtgwg-ooam-header]. Message Type field in the Overlay OAM Header MUST be set to Overlay Echo Request value (TBA2).

Value of the Reply Mode field MAY be set to:

- Do Not Reply (TBA4) if one-way monitoring is desired. If Echo Request is used to measure synthetic packet loss, the receiver MAY report loss measurement results to a remote node.
- Reply via an IPv4/IPv6 UDP Packet (TBA5) value likely will be the most used.
- Reply via Application Level Control Channel (TBA6) value if the overlay network MAY have bi-directional paths.

- Reply via Specified Path (TBA7) value in order to enforce use of the particular return path specified in the included TLV to verify bi-directional continuity and also increase robustness of the monitoring by selecting more stable path.

2.4. Overlay Echo Request Reception

2.5. Overlay Echo Reply Transmission

The Reply Mode field directs whether and how the Echo Reply message should be sent. The sender of the Echo Request MAY use TLVs to request that corresponding Echo Reply be sent using the specified path. Value TBA3 is referred as "Do not reply" mode and suppresses transmission of Echo Reply packet. Default value (TBA5) for the Reply mode field requests the responder to send the Echo Reply packet out-of-band as IPv4 or IPv6 UDP packet. [Selection of destination and source IP addresses and UDP port numbers to be provided in the next update.]

2.6. Overlay Echo Reply Reception

3. IANA Considerations

3.1. Overlay Echo Request/Echo Reply Type

IANA is requested to assign new type from the Overlay OAM Protocol Types registry as follows:

Value	Description	Reference
TBA1	Overlay Echo Request/Echo Reply	This document

Table 1: Overlay Echo Request/Echo Reply Type

3.2. Overlay Ping Parameters

IANA is requested to create new Overlay Echo Request/Echo Reply Parameters registry.

3.3. Overlay Echo Request/Echo Reply Message Types

IANA is requested to create in the Overlay Echo Request/Echo Reply Parameters registry the new sub-registry Message Types. All code points in the range 1 through 191 in this registry shall be allocated according to the "IETF Review" procedure as specified in [RFC5226] and assign values as follows:

Value	Description	Reference
0	Reserved	
TBA2	Overlay Echo Request	This document
TBA3	Overlay Echo Reply	This document
TBA3+1-191	Unassigned	IETF Review
192-251	Unassigned	First Come First Served
252-254	Unassigned	Private Use
255	Reserved	

Table 2: Overlay Echo Request/Echo Reply Message Types

3.4. Overlay Echo Reply Modes

IANA is requested to create in the Overlay Echo Request/Echo Reply Parameters registry the new sub-registry Reply Modes All code points in the range 1 through 191 in this registry shall be allocated according to the "IETF Review" procedure as specified in [RFC5226] and assign values as follows:

Value	Description	Reference
0	Reserved	
TBA4	Do Not Reply	This document
TBA5	Reply via an IPv4/IPv6 UDP Packet	This document
TBA6	Reply via Application Level Control Channel	This document
TBA7	Reply via Specified Path	This document
TBA7+1-191	Unassigned	IETF Review
192-251	Unassigned	First Come First Served
252-254	Unassigned	Private Use
255	Reserved	

Table 3: Overlay Echo Reply Modes

4. Security Considerations

Overlay Echo Request/Reply operates within the domain of the overlay network and thus inherits any security considerations that apply to the use of that overlay technology and, consequently, underlay data plane. Also, the security needs for Overlay Echo Request/Reply are

similar to those of ICMP ping [RFC0792], [RFC4443] and MPLS LSP ping [I-D.ietf-mpls-rfc4379bis].

There are at least three approaches of attacking a node in the overlay network using the mechanisms defined in the document. One is a Denial-of-Service attack, by sending Overlay ping to overload a node in the overlay network. The second may use spoofing, hijacking, replying, or otherwise tampering with Overlay Echo Requests and/or Replies to misrepresent, alter operator's view of the state of the overlay network. The third is an unauthorized source using an Overlay Echo Request/Reply to obtain information about the overlay and/or underlay network.

To mitigate potential Denial-of-Service attacks, it is RECOMMENDED that implementations throttle the Overlay ping traffic going to the control plane.

Reply and spoofing attacks involving faking or replying Overlay Echo Reply messages would have to match the Sender's Handle and Sequence Number of an outstanding Overlay Echo Request message which is highly unlikely. Thus the non-matching reply would be discarded. But since "even a broken clock is right twice a day" implementations MAY use Timestamp control block [I-D.ooamdt-rtgwg-ooam-header] to validate the TimeStamp Sent by requiring an exact match on this field.

To protect against unauthorized sources trying to obtain information about the overlay and/or underlay an implementation MAY check that the source of the Echo Request is indeed part of the overlay domain.

5. Contributors

Work on this documented started by Overlay OAM Design Team with contributions from:

Carlos Pignataro

Cisco Systems, Inc.

cpignata@cisco.com

Santosh Pallagatti

santosh.pallagatti@gmail.com

Erik Nordmark

Arista Networks

nordmark@acm.org

Ignas Bagdonas

ibagdona@gmail.com

David Mozes

Mellanox Technologies Ltd.

davidm@mellanox.com

6. Acknowledgment

TBD

7. References

7.1. Normative References

[I-D.ietf-bier-mpls-encapsulation]
Wijnands, I., Rosen, E., Dolganow, A., Tantsura, J., Aldrin, S., and I. Meilik, "Encapsulation for Bit Index Explicit Replication in MPLS and non-MPLS Networks", draft-ietf-bier-mpls-encapsulation-06 (work in progress), December 2016.

[I-D.ietf-nvo3-geneve]
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-03 (work in progress), September 2016.

[I-D.ietf-nvo3-gue]
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-ietf-nvo3-gue-05 (work in progress), October 2016.

[I-D.ietf-nvo3-vxlan-gpe]
Maino, F., Kreeger, L., and U. Elzur, "Generic Protocol Extension for VXLAN", draft-ietf-nvo3-vxlan-gpe-03 (work in progress), October 2016.

[I-D.ietf-sfc-nsh]
Quinn, P. and U. Elzur, "Network Service Header", draft-ietf-sfc-nsh-12 (work in progress), February 2017.

[I-D.ooamdt-rtgwg-ooam-header]
Mirsky, G., Kumar, N., Kumar, D., Chen, M., Yizhou, L., Mozes, D., and D. Dolson, "OAM Header for use in Overlay Networks", draft-ooamdt-rtgwg-ooam-header-02 (work in progress), February 2017.

[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>.

7.2. Informative References

[I-D.ietf-mpls-rfc4379bis]
Kompella, K., Swallow, G., Pignataro, C., Kumar, N., Aldrin, S., and M. Chen, "Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures", draft-ietf-mpls-rfc4379bis-09 (work in progress), October 2016.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <http://www.rfc-editor.org/info/rfc792>.

[RFC4379] Kompella, K. and G. Swallow, "Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures", RFC 4379, DOI 10.17487/RFC4379, February 2006, <http://www.rfc-editor.org/info/rfc4379>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, DOI 10.17487/RFC4443, March 2006, <http://www.rfc-editor.org/info/rfc4443>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

Authors' Addresses

Greg Mirsky
ZTE Corp.

Email: gregimirsky@gmail.com

Nagendra Kumar
Cisco Systems, Inc.

Email: naikumar@cisco.com

Deepak Kumar
Cisco Systems, Inc.

Email: dekumar@cisco.com

Mach Chen
Huawei Technologies

Email: mach.chen@huawei.com

Yizhou Li
Huawei Technologies

Email: liyizhou@huawei.com

David Dolson
Sandvine

Email: ddolson@sandvine.com

NVO3 Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 19, 2018

G. Mirsky
ZTE Corp.
N. Kumar
D. Kumar
Cisco Systems, Inc.
M. Chen
Y. Li
Huawei Technologies
D. Dolson
Sandvine

March 18, 2018

OAM Header for use in Overlay Networks
draft-ooamdt-rtgwg-ooam-header-04

Abstract

This document introduces Overlay Operations, Administration, and Maintenance (OOAM) Header to be used in overlay networks to create Overlay Associated Channel (OAC) to ensure that OOAM control packets are in-band with user traffic and de-multiplex OOAM protocols.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 19, 2018.

Copyright Notice

Table of Contents

1. Introduction

New protocols that support overlay networks like VxLAN-GPE [I-D.ietf-nvo3-vxlan-gpe], GUE [I-D.ietf-nvo3-gue], Geneve [I-D.ietf-nvo3-geneve], BIER [RFC8296], and NSH [RFC8300] support multi-protocol payload, e.g. Ethernet, IPv4/IPv6, and recognize Operations, Administration, and Maintenance (OAM) as one of distinct types. That ensures that Overlay OAM (OOAM)packets are sharing fate with Overlay data packet traversing the underlay.

This document introduces generic requirements to OAM protocols used in overlay networks and defines OOAM Header to be used in overlay networks to de-multiplex OOAM protocols.

1.1. Conventions used in this document

1.1.1. Terminology

Term "Overlay OAM" used in this document interchangeably with longer version "set of OAM protocols, methods and tools for Overlay networks".

NTP Network Time Protocol

OAC Overlay Associated Channel

OAM Operations, Administration, and Maintenance

OOAM Overlay OAM

PTP Precision Time Protocol

1.1.2. Requirements Language

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 [RFC2119].

2. General Requirements to OAM Protocols in Overlay Networks

OAM protocols, whether it is part of fault management or performance monitoring, intended to provide reliable information that can be used to identify defect, localize it and apply corrective actions. One of the main challenges that network operators may encounter is interpretations of reports of the defect or service degradation and correlation to affected services. In order to improve reliability of the correlation process we set forth the following requirements:

REQ#1: Overlay OAM packets SHOULD be fate sharing with data traffic, i.e. in-band with the monitored traffic, i.e. follow exactly the same overlay and transport path as data plane traffic, in forward direction, i.e. from ingress toward egress end point(s) of the OAM test.

REQ#2: Encapsulation of OAM control message and data packets in underlay network MUST be indistinguishable from underlay network forwarding point of view.

REQ#3: Presence of OAM control message in overlay packet MUST be unambiguously identifiable.

REQ#4: It MUST be possible to express entropy for underlay Equal Cost Multipath in overlay encapsulation in order to avoid using data packet content by underlay transient nodes.

3. Associated Channel in Overlay Networks

Associated channel in the overlay network is the channel that, by using the same encapsulation as user traffic, follows the same path through the underlay network as user traffic. In other words, the associated channel is in-band with user traffic. Creating notion of the overlay associated channel (OAC) in the overlay network ensures that control packets of active OAM protocols carried in the OAC are in-band with user traffic. Additionally, OAC allows development of OAM tools that, from operational point of view, function in essentially the same manner in any type of overlay.

4. Overlay OAM Header

OOAM Header immediately follows the header of the overlay and identifies OAC. The format of the OOAM Header is:

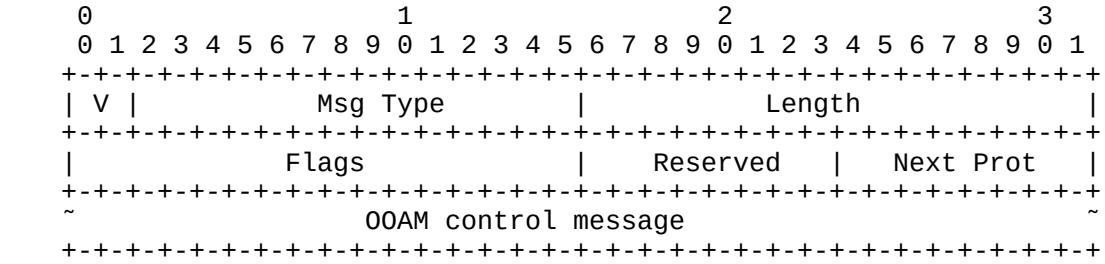

Figure 1: Overlay OAM Header format

The OAM Header consists of the following fields:

- V - two bits long field indicates the current version of the Overlay OAM Header. The current value is 0;
- Msg Type - 14 bits long field identifies OAM protocol, e.g. Echo Request/Reply, BFD, Performance Measurement;
- Length - two octets long field that is length of the OOAM control packet in octets;
- Flags -two octets long field carries bit flags that define optional capability and thus processing of the OOAM control packet;

- Reserved - one octet field that MUST be zeroed on transmit and ignored on receipt;

- Next Prot - one octet long field that defines optional payload that is present after the OOAM Control Packet.

The format of the Flags field is:

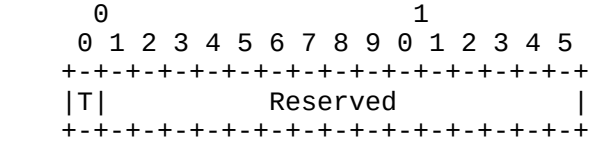

Figure 2: Flags field format

where:

- T - Timestap block flag.

- Reserved - must be set to all zeroes on transmission and ignored on receipt.

The OOAM header may be followed by the Timestamp control block Figure 3 and then by OOAM Control Packet identified by the Msg Type field.

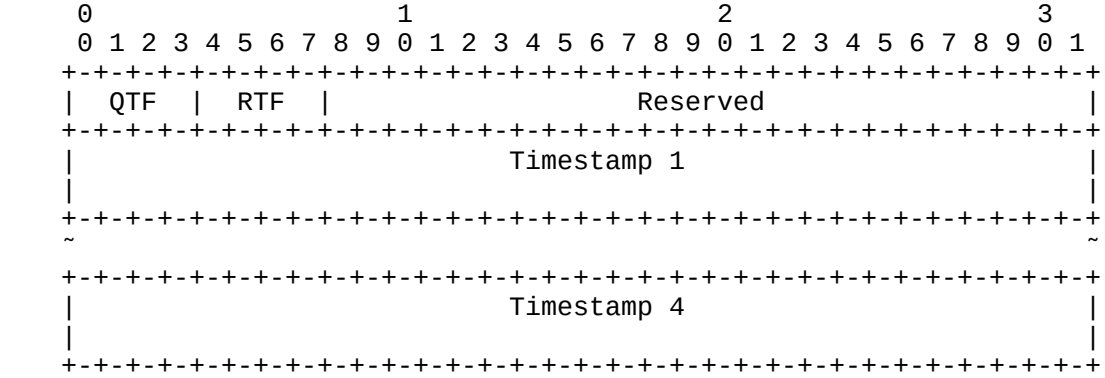

Figure 3: Timestamp block format

where:

QTF - Querier timestamp format

RTF - Responder timestamp format

Timestamp 1-4 - 64-bit timestamp values

Network Time Protocol (NTP), described in [RFC5905], is widely used and has long history of deployment. But it is the IEEE 1588 Precision Time Protocol (PTP) [IEEE.1588.2008] that is being broadly used to achieve high-quality clock synchronization. Converging between NTP and PTP time formats is possible but is not trivial and does come with cost, particularly when it is required to be performed in real time without loss of accuracy. And recently protocols that supported only NTP time format, like One-Way Active Measurement Protocol [RFC4656] and Two-Way Active Measurement Protocol [RFC5357], have been enhanced to support the PTP time format as well [RFC8186]. This document proposes to select PTP time format as default time format for Overlay OAM performance measurement. Hence QTF, RTF fields MUST be set to 0 if querier or responder use PTP time format respectively. If the querier or responder use the NTP time format, then QTF and/or RTF MUST be set to 1. Use of other values MUST be considered as error and MAY be reported.

4.1. Use of OOAM Header in Active OAM

Active OAM methods, whether used for fault management or performance monitoring, generate dedicated test packets [RFC7799]. Format of an OAM test packet in overlay network presented in Figure 4.

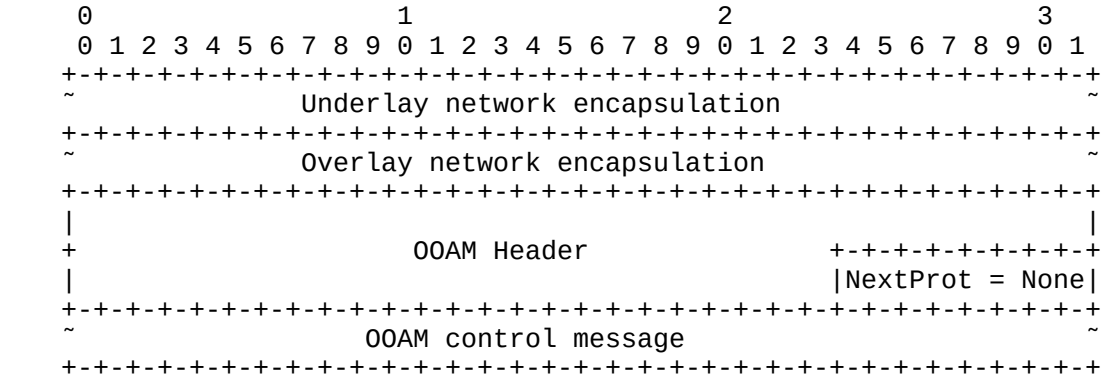

Figure 4: Overlay OAM Header in Active OAM Control Packet

Because active OAM method uses only OAM protocol value of Next Prot field in the OOAM header is set to None indicating that there’s no content from other protocol immediately after OOAM control message in the packet.

4.2. Use of OOAM Header in Hybrid OAM

Hybrid OAM Type I methods, whether used for fault management or performance monitoring, modify user data packets [RFC7799]. Format of such modified packet in overlay network presented in Figure 5.

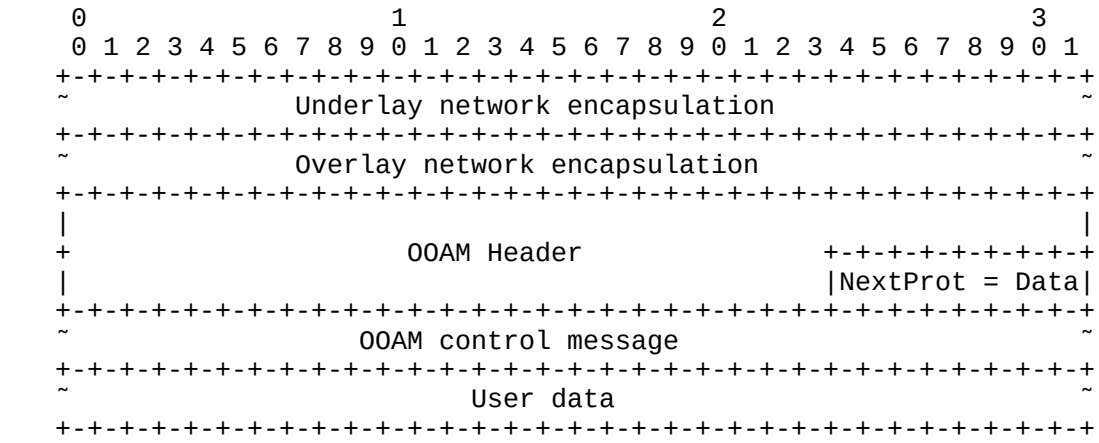

Figure 5: Overlay OAM Header in Hybrid OAM Control Packet

In case when OOAM header used for Hybrid Type I OAM method value of the Next Prot field is set to the value associated with the protocol of the user data.

5. IANA Considerations

IANA is requested to create new registry called "Overlay OAM".

5.1. OOAM Message Types

IANA is requested to create new sub-registry called "Overlay OAM Protocol Types" in the "Overlay OAM" registry. All code points in the range 1 through 15615 in this registry shall be allocated according to the "IETF Review" procedure as specified in [RFC8126] . Remaining code points are allocated according to the Table 1:

Value	Description	Reference
0	Reserved	
1 - 15615	Unassigned	IETF Review
15616 - 16127	Unassigned	First Come First Served
16128 - 16143	Experimental	This document
16144 - 16382	Private Use	This document
16383	Reserved	This document

Table 1: Overlay OAM Protocol type

5.2. OOAM Header Flags

IANA is requested to create sub-registry "Overlay OAM Header Flags" in "Overlay OAM" registry. Two flags are defined in this document. New values are assigned via Standards Action [RFC8126].

Flags bit	Description	Reference
Bit 0	Timestamp field	This document
Bit 1-15	Unassigned	

Table 2: Overlay OAM Flags

6. Security Considerations

TBD

7. Contributors

Work on this documented started by Overlay OAM Design Team with contributions from:

Carlos Pignataro

Cisco Systems, Inc.

cpignata@cisco.com

Erik Nordmark

Arista Networks

nordmark@acm.org

Ignas Bagdonas

ibagdona@gmail.com

David Mozes

Mellanox Technologies Ltd.

davidm@mellanox.com

8. Acknowledgement

TBD

9. References

9.1. Normative References

[IEEE.1588.2008]
"Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE Standard 1588, July 2008.

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

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

9.2. Informative References

[I-D.ietf-nvo3-geneve]
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-06 (work in progress), March 2018.

[I-D.ietf-nvo3-gue]
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-ietf-nvo3-gue-05 (work in progress), October 2016.

[I-D.ietf-nvo3-vxlan-gpe]
Maino, F., Kreeger, L., and U. Elzur, "Generic Protocol Extension for VXLAN", draft-ietf-nvo3-vxlan-gpe-05 (work in progress), October 2017.

[RFC4656] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)", RFC 4656, DOI 10.17487/RFC4656, September 2006, <https://www.rfc-editor.org/info/rfc4656>.

[RFC5357] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, DOI 10.17487/RFC5357, October 2008, <https://www.rfc-editor.org/info/rfc5357>.

[RFC7799] Morton, A., "Active and Passive Metrics and Methods (with Hybrid Types In-Between)", RFC 7799, DOI 10.17487/RFC7799, May 2016, <https://www.rfc-editor.org/info/rfc7799>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8186] Mirsky, G. and I. Meilik, "Support of the IEEE 1588 Timestamp Format in a Two-Way Active Measurement Protocol (TWAMP)", RFC 8186, DOI 10.17487/RFC8186, June 2017, <https://www.rfc-editor.org/info/rfc8186>.

[RFC8296] Wijnands, IJ., Ed., Rosen, E., Ed., Dolganow, A., Tantsura, J., Aldrin, S., and I. Meilik, "Encapsulation for Bit Index Explicit Replication (BIER) in MPLS and Non-MPLS Networks", RFC 8296, DOI 10.17487/RFC8296, January 2018, <https://www.rfc-editor.org/info/rfc8296>.

[RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <https://www.rfc-editor.org/info/rfc8300>.

Authors' Addresses

Greg Mirsky
ZTE Corp.

Email: gregimirsky@gmail.com

Nagendra Kumar
Cisco Systems, Inc.

Email: naikumar@cisco.com

Deepak Kumar
Cisco Systems, Inc.

Email: dekumar@cisco.com

Mach Chen
Huawei Technologies

Email: mach.chen@huawei.com

Yizhou Li
Huawei Technologies

Email: liyizhou@huawei.com

David Dolson
Sandvine

Email: ddolson@sandvine.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2018

S. Pallagatti, Ed.
Independent Contributor
S. Paragiri
Juniper Networks
V. Govindan
M. Mudigonda
Cisco
G. Mirsky
ZTE Corp.
October 13, 2017

BFD for VXLAN
draft-spallagatti-bfd-vxlan-06

Abstract

This document describes use of Bidirectional Forwarding Detection (BFD) protocol in Virtual eXtensible Local Area Network (VXLAN) overlay network.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 16, 2018.

Copyright Notice

Table of Contents

1. Introduction

"Virtual eXtensible Local Area Network (VXLAN)" has been described in [RFC7348]. VXLAN provides an encapsulation scheme that allows virtual machines (VMs) to communicate in a data center network.

VXLAN is typically deployed in data centers interconnecting virtualized hosts, which may be spread across multiple racks. The individual racks may be part of a different Layer 3 network or they could be in a single Layer 2 network. The VXLAN segments/overlay networks are overlaid on top of these Layer 2 or Layer 3 networks.

A VM can communicate with another VM only if they are on the same VXLAN. VMs are unaware of VXLAN tunnels as VXLAN tunnel is terminated on VXLAN Tunnel End Point (VTEP) (hypervisor/TOR). VTEPs (hypervisor/TOR) are responsible for encapsulating and decapsulating frames exchanged among VMs.

Since underlay is a L3 network, ability to monitor path continuity, i.e. perform proactive continuity check (CC) for these tunnels is important. Asynchronous mode of BFD, as defined in [RFC5880], can be

used to monitor a VXLAN tunnel. Use of [I-D.ietf-bfd-multipoint] is for future study.

Also BFD in VXLAN can be used to monitor special service nodes that are designated to properly handle Layer 2 broadcast, unknown unicast, and multicast traffic. Such nodes, often referred "replicators", are usually virtual VTEPs can be monitored by physical VTEPs in order to minimize BUM traffic directed to unavialable replicator.

This document describes use of Bidirectional Forwarding Detection (BFD) protocol VXLAN to enable continuity monitoring between Network Virtualization Edges (NVEs) and/or availability of a replicator service node using BFD.

2. Conventions used in this document

2.1. Terminology

BFD - Bidirectional Forwarding Detection

CC - Continuity Check

NVE - Network Virtualization Edge

TOR - Top of Rack

VM - Virtual Machine

VTEP - VXLAN Tunnel End Point

VXLAN - Virtual eXtensible Local Area Network

2.2. Requirements Language

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.

3. Use cases

Main use case of BFD for VXLAN is for continuity check of a tunnel. By exchanging BFD control packets between VTEPs an operator exercises the VXLAN path in both in underlay and overlay thus ensuring the VXLAN path availability and VTEPs reachability. BFD failure detection can be used for maintenance. There are other use cases such as

Layer 2 VMs:

Most deployments will have VMs with only L2 capabilities that may not support L3. BFD being a L3 protocol can be used as tunnel CC mechanism, where BFD will start and terminate at the NVEs, e.g. VTEPs.

It is possible to aggregate the CC sessions for multiple tenants by running a BFD session between the VTEPs over VxLAN tunnel. In rest of this document terms NVE and VTEP are used interchangeably.

Fault localization:

It is also possible that VMs are L3 aware and can possibly host a BFD session. In these cases BFD sessions can be established among VMs for CC. In addition, BFD sessions can be established among VTEPs for tunnel CC. Having a hierarchical OAM model helps localize faults though requires additional consideration.

Service node reachability:

Service node is responsible for sending BUM traffic. In case of service node tunnel terminates at VTEP and it might not even host VM. BFD session between TOR/hypervisor and service node can be used to monitor service node reachability.

4. Deployment

Figure 1 illustrates the scenario with two servers, each of them hosting two VMs. These servers host VTEPs that terminate two VXLAN tunnels with VNI number 100 and 200. Separate BFD sessions can be established between the VTEPs (IP1 and IP2) for monitoring each of the VXLAN tunnels (VNI 100 and 200). No BFD packets, intended to Hypervisor VTEP, should be forwarded to a VM as VM may drop BFD packets leading to false negative. This method is applicable whether VTEP is a virtual or physical device.

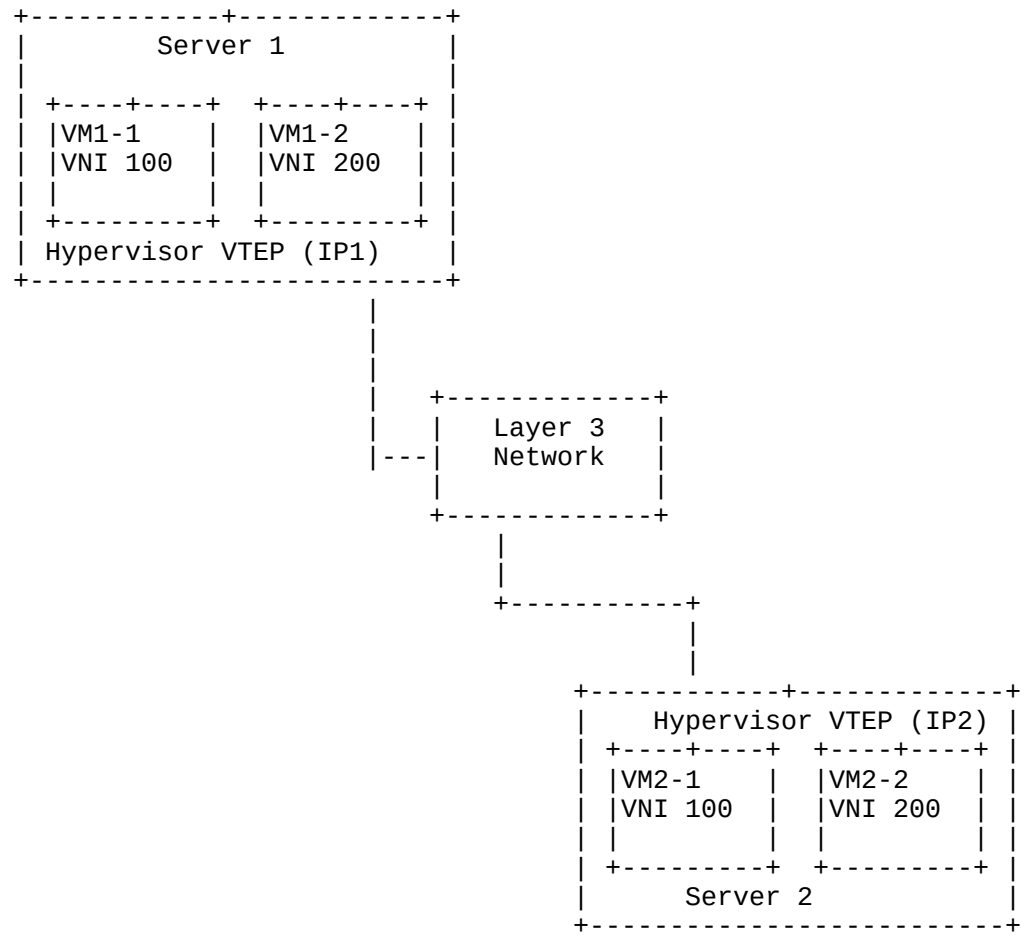

Figure 1: Reference VXLAN domain

5. BFD Packet Transmission over VXLAN Tunnel

BFD packet MUST be encapsulated and sent to a remote VTEP as explained in Section 5.1. Implementations SHOULD ensure that the BFD packets follow the same lookup path of VXLAN packets within the sender system.

5.1. BFD Packet Encapsulation in VXLAN

VXLAN packet format has been described in Section 5 of [RFC7348]. The Outer IP/UDP and VXLAN headers MUST be encoded by the sender as per [RFC7348].

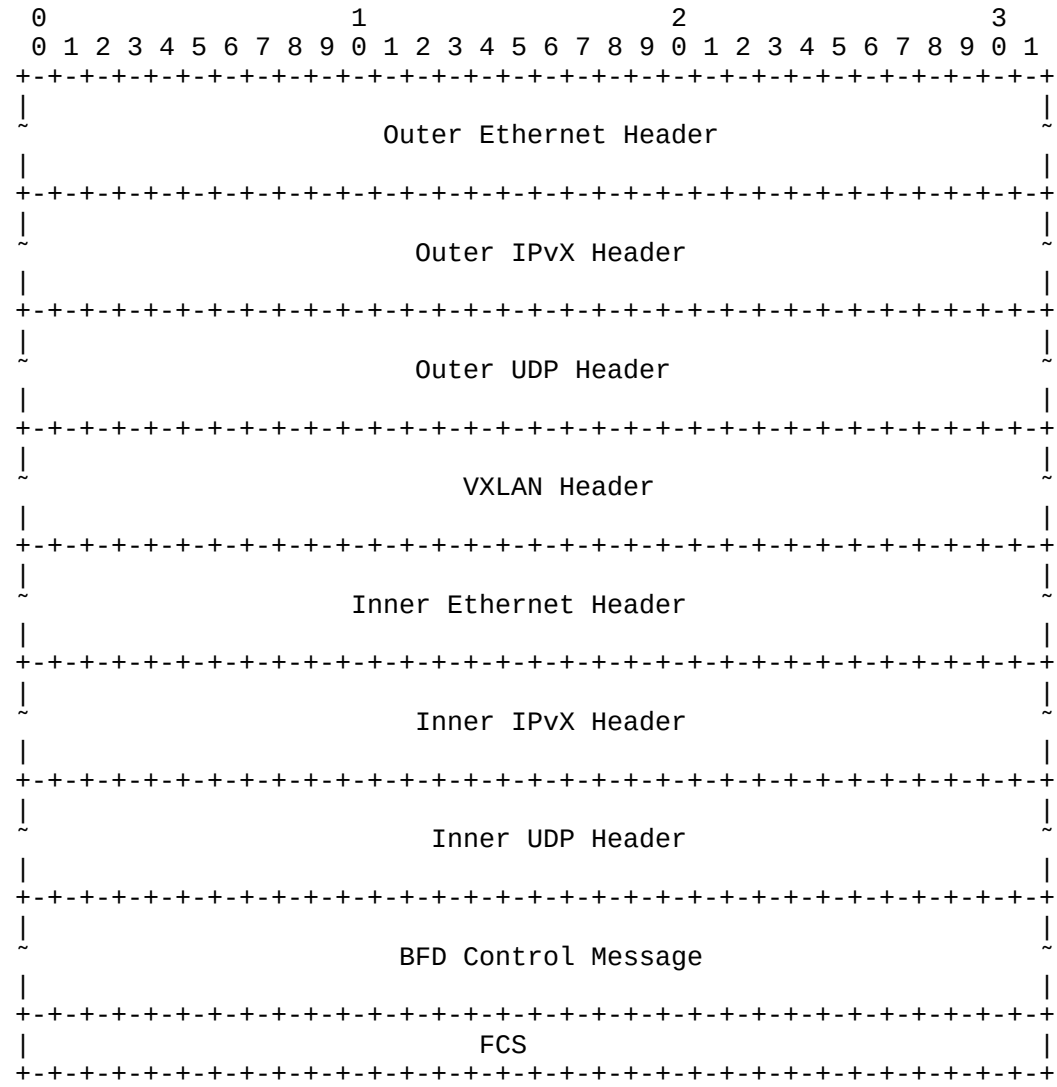

Figure 2: VXLAN Encapsulaion of BFD Control Message

The BFD packet MUST be carried inside the inner MAC frame of the VXLAN packet. The inner MAC frame carrying the BFD payload has the following format:

Ethernet Header:

Destination MAC: This MUST be a dedicated MAC (TBA) Section 9 or the MAC address of the destination VTEP. The details of how the MAC address of the destination VTEP is obtained are outside the scope of this document.

Source MAC: MAC address of the originating VTEP

IP header:

Source IP: IP address of the originating VTEP.

Destination IP: IP address of the terminating VTEP.

TTL: This MUST be set to 1. This is to ensure that the BFD packet is not routed within the L3 underlay network.

[Ed.Note]:Use of inner source and destination IP addresses needs more discussion by the WG.

The fields of the UDP header and the BFD control packet are encoded as specified in [RFC5881] for p2p VXLAN tunnels.

6. Reception of BFD packet from VXLAN Tunnel

Once a packet is received, VTEP MUST validate the packet as described in Section 4.1 of [RFC7348]. If the Destination MAC of the inner MAC frame matches the dedicated MAC or the MAC address of the VTEP the packet MUST be processed further.

The UDP destination port and the TTL of the inner Ethernet frame MUST be validated to determine if the received packet can be processed by BFD. BFD packet with inner MAC set to VTEP or dedicated MAC address MUST NOT be forwarded to VMs.

To ensure BFD detects the proper configuration of VXLAN Network Identifier (VNI) in a remote VTEP, a lookup SHOULD be performed with the MAC-DA and VNI as key in the Virtual Forwarding Instance (VFI) table of the originating/ terminating VTEP in order to exercise the VFI associated with the VNI.

6.1. Demultiplexing of the BFD packet

Demultiplexing of IP BFD packet has been defined in Section 3 of [RFC5881]. Since multiple BFD sessions may be running between two VTEPs, there needs to be a mechanism for demultiplexing received BFD packets to the proper session. The procedure for demultiplexing packets with Your Discriminator equal to 0 is different from [RFC5880]. For such packets, the BFD session MUST be identified using the inner headers, i.e. the source IP and the destination IP present in the IP header carried by the payload of the VXLAN encapsulated packet. The VNI of the packet SHOULD be used to derive interface related information for demultiplexing the packet. If BFD packet is received with non-zero Your Discriminator then BFD session MUST be demultiplexed only with Your Discriminator as the key.

7. Use of reserved VNI

BFD session MAY be established for the reserved VNI 0. One way to aggregate BFD sessions between VTEP’s is to establish a BFD session with VNI 0. A VTEP MAY also use VNI 0 to establish a BFD session with a service node.

8. Echo BFD

Support for echo BFD is outside the scope of this document.

9. IANA Considerations

IANA is requested to assign a dedicated MAC address to be used as the Destination MAC address of the inner Ethernet which carries BFD control packet in IP/UDP encapsulation.

10. Security Considerations

Document recommends setting of inner IP TTL to 1 which could lead to DDoS attack, implementation MUST have throttling in place. Throttling MAY be relaxed for BFD packets based on port number.

Other than inner IP TTL set to 1 this specification does not raise any additional security issues beyond those of the specifications referred to in the list of normative references.

11. Contributors

Reshad Rahman
rrahman@cisco.com
Cisco

12. Acknowledgments

Authors would like to thank Jeff Hass of Juniper Networks for his reviews and feedback on this material.

Authors would also like to thank Nobo Akiya, Marc Binderberger and Shahram Davari for the extensive review.

13. Normative References

[I-D.ietf-bfd-multipoint] Katz, D., Ward, D., and J. Networks, "BFD for Multipoint Networks", draft-ietf-bfd-multipoint-10 (work in progress), April 2017.

[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>.

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <https://www.rfc-editor.org/info/rfc5880>.

[RFC5881] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop)", RFC 5881, DOI 10.17487/RFC5881, June 2010, <https://www.rfc-editor.org/info/rfc5881>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Authors' Addresses

Santosh Pallagatti (editor)
Independent Contributor

Email: santosh.pallagatti@gmail.com

Sudarsan Paragiri
Juniper Networks
1194 N. Mathilda Ave.
Sunnyvale, California 94089-1206
USA

Email: sparagiri@juniper.net

Vengada Prasad Govindan
Cisco

Email: venggovi@cisco.com

Mallik Mudigonda
Cisco

Email: mmudigon@cisco.com

Greg Mirsky
ZTE Corp.

Email: gregimirsky@gmail.com