NVO3 WG Fangwei Hu
Internet-Draft Ran Chen
Intended status: Standards Track ZTE Corporation
Expires: March 1, 2019 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

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Terminology 2
3. Requirements Language 2
4. YANG Data Model for VxLAN Configuration 2
  4.1. VxLAN Multicast IP Address 2
  4.2. VxLAN Access Type 3
  4.3. Inner VLAN Tag Handling Mode 3
5. Design Tree of VxLAN YANG Data Model 3
6. VxLAN YANG Model 5
7. Security Considerations 17
8. Acknowledgements 18
9. IANA Considerations 18
10. Normative References 19
Authors' Addresses 20

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:

- o vlan-1:1: the vxlan access type is VLAN, and each VxLAN is only mapping to one VLAN.

- o vlan- n:1: the vxlan access type is VLAN, and each VxLAN is mapped to several VLANs.

- o L3-interface: the VxLAN access type is layer 3 interface.

- o mac: the VxLAN access type is MAC address.

- o 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

T. Herbert
Facebook

February 29, 2016

Remote checksum offload for encapsulation
draft-herbert-remotecsumoffload-02

Abstract

This document describes remote checksum offload for encapsulation, which is a mechanism that provides checksum offload of encapsulated packets using rudimentary offload capabilities found in most Network Interface Card (NIC) devices. The outer header checksum e.g. that in UDP or GRE) is enabled in packets and, with some additional meta information, a receiver is able to deduce the checksum to be set for an inner encapsulated packet. Effectively this offloads the computation of the inner checksum. Enabling the outer checksum in encapsulation has the additional advantage that it covers more of the packet than the inner checksum including the encapsulation headers.

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

Copyright and License Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1 Introduction 3
2 Checksum offload background 3
2.1 The Internet checksum 3
2.2 Transmit checksum offload 4
2.2.1 Generic transmit offload 4
2.2.2 Local checksum offload 4
2.2.3 Protocol specific transmit offload 5
2.3 Receive checksum offload 5
2.3.1 CHECKSUM_COMPLETE 6
2.3.2 CHECKSUM_UNNECESSARY 6
3.0 Remote checksum offload 6
3.1 Option format 6
3.2 Transmit operation 7
3.3 Receiver operation 8
3.4 Interaction with TCP segmentation offload 9
4 Security Considerations 9
5 IANA Considerations 9
6 References 9
6.1 Normative References 9
6.2 Informative References 10
Authors' Addresses 10

1 Introduction

Checksum offload is a capability of NICs where the checksum calculation for a transport layer packet (TCP, UDP, etc.) is performed by a device on behalf of the host stack. Checksum offload is applicable to both transmit and receive, where on transmit the device writes the computed checksum into the packet, and on receive the device provides the computed checksum of the packet or an indication that specific transport checksums were validated. This feature saves CPU cycles in the host and has become ubiquitous in modern NICs.

A host may both source transport packets and encapsulate them for transit over an underlying network. In this case checksum offload is still desirable, but now must be done on an encapsulated packet. Many deployed NICs are only capable of providing checksum offload for simple TCP or UDP packets. Such NICs typically use protocol specific mechanisms where they must parse headers in order to perform checksum calculations. Updating these NICs to perform checksum offload for encapsulation requires new parsing logic which is likely infeasible or at cost prohibitive.

In this specification we describe an alternative that uses rudimentary NIC offload features to support offloading checksum calculation of encapsulated packets. In this design, the outer checksum is enabled on transmit, and meta information indicating the location of the checksum field being offloaded and its starting point for computation are sent with a packet. On receipt, after the outer checksum is verified, the receiver sets the offloaded checksum field per the computed packet checksum and the meta data.

2 Checksum offload background

In this section we provide some background into checksum offload operation.

2.1 The Internet checksum

The Internet checksum [RFC0791] is used by several Internet protocols including IP [RFC1122], TCP [RFC0793], UDP [RFC0768] and GRE [RFC2784]. Efficient checksum calculation is critical to good performance [RFC1071], and the mathematical properties are useful in incrementally updating checksums [RFC1624]. An early approach to implementing checksum offload in hardware is described in [RFC1936].

TCP and UDP checksums cover a pseudo header which is composed of the source and destination addresses of the corresponding IP packet, upper layer packet length, and protocol. The checksum pseudo header

is defined in [RFC0768] and [RFC0793] for IPv4, and in [RFC2460] for IPv6.

2.2 Transmit checksum offload

In transmit checksum offload, a host network stack defers the calculation and setting of a transport checksum in the packet to the device. A device may provide checksum offload only for specific protocols, or may provide a generic interface. In either case, support for only one offloaded checksum per packet is typical.

When using transmit checksum offload, a host stack must initialize the checksum field in the packet. This is done by setting to zero (GRE) or to the bitwise not of the pseudo header (UDP or TCP). The device proceeds by computing the packet checksum from the start of the transport header through to the end of the packet. The bitwise not of the resulting value is written in the checksum field of the transport packet.

2.2.1 Generic transmit offload

A device can provide a generic interface for transmit checksum offload. Checksum offload is enabled by setting two fields in the transmit descriptor for a packet: start offset and checksum offset. The start offset indicates the byte in the packet where the checksum calculation should start. The checksum offset indicates the offset in the packet where the checksum value is to be written.

The generic interface is protocol agnostic, however only supports one offloaded checksum per packet. While it is conceivable that a NIC could provide offload for more checksums by defining more than one checksum start/offset pair in the transmit descriptor, a more general and efficient solution is Local Checksum Offload.

2.2.2 Local checksum offload

Local Checksum Offload [LCO] (or LCO) is a technique for efficiently computing the outer checksum of an encapsulated datagram when the inner checksum is due to be offloaded. The ones-complement sum of a correctly checksummed TCP or UDP packet is equal to the sum of the pseudo header, since everything else gets ’cancelled out’ by the checksum field. This property holds since the sum was complemented before being written to the checksum field. More generally, this holds in any case where the Internet one’s complement checksum is used, and thus any checksum that generic transmit offload supports. That is, if we have set up transmit checksum offload with a start/offset pair, we know that after the device has filled in that checksum the one’s complement sum from checksum start to the end of

the packet will be equal to whatever value is set in the checksum field beforehand. This property allows computing the outer checksum without considering at the payload per the algorithm:

1) Compute the checksum from the outer packet's checksum start offset to the inner packet's checksum start offset.

2) Add the bit-wise not of the pseudo header checksum for the inner packet.

3) The result is the checksum from the outer packet's start offset to the end of the packet. Taking into account the pseudo header for the outer checksum allows the outer checksum field to be set without offload processing.

Step 1) requires that some checksum calculation is performed on the host stack, however this is only done over some portion of packet headers which is typically much smaller than the payload of the packet.

LCO can be used for nested encapsulations; in this case, the outer encapsulation layer will sum over both its own header and the 'middle' header. Thus, if the device has the capability to offload an inner checksum in encapsulation, any number of outer checksums can be efficiently calculated using this technique.

2.2.3 Protocol specific transmit offload

Some devices support transmit checksum offload for very specific protocols. For instance, many legacy devices can only perform checksum offload for UDP/IP and TCP/IP packets. These devices parse transmitted packets in order to determine the checksum start and checksum offset. They may also ignore the value in the checksum field by setting it to zero for checksum computation and computing the checksum of the pseudo header themselves.

Protocol specific transmit offload is limited to the protocols a device supports. To support checksum offload of an encapsulated packet, a device must be a able to parse the encapsulation layer in order to locate the inner packet.

2.3 Receive checksum offload

Upon receiving a packet, a device may perform a checksum calculation over the packet or part of the packet depending on the protocol. A result of this calculation is returned in the meta data of the receive descriptor for the packet. The host stack can apply the result in verifying checksums as it processes the packet. The intent

is that the offload will obviate the need for the networking stack to perform its own checksum calculation over the packet.

There are two basic methods of receive checksum offload: CHECKSUM_COMPLETE and CHECKSUM_UNNECESSARY.

2.3.1 CHECKSUM_COMPLETE

A device may calculate the checksum of a whole packet (layer 2 payload) and return the resultant value to the host stack. The host stack can subsequently use this value to validate checksums in the packet. As the packet is parsed through various layers, the calculated checksum is updated to correspond to each layer (subtract out checksum for preceding bytes for a given header).

CHECKSUM_COMPLETE is protocol agnostic and does not require any protocol awareness in the device. It works for any encapsulation and supports an arbitrary number of checksums in the packet.

2.3.2 CHECKSUM_UNNECESSARY

A device may explicitly validate a checksum in a packet and return a flag in the receive descriptor that a transport checksum has been verified (host performing checksum computation is unnecessary). Some devices may be capable of validating more than one checksum in the packet, in which case the device returns a count of the number verified. Typically, only a positive signal is returned, if the device was unable to validate a checksum it does not return any information and the host will generally perform its own checksum computation. If a device returns a count of validations, this must refer to consecutive checksums that are present and validated in a packet (checksums cannot be skipped).

CHECKSUM_UNNECESSARY is protocol specific, for instance in the case of UDP or TCP a device needs to consider the pseudo header in checksum validation. To support checksum offload of an encapsulated packet, a device must be able to parse the encapsulation layer in order to locate the inner packet.

3.0 Remote checksum offload

This section describes the remote checksum offload mechanism. This is primarily useful with UDP based encapsulation where the UDP checksum is enabled (not set to zero on transmit). The same technique could be applied to GRE encapsulation where the GRE checksum is enabled.

3.1 Option format

Remote checksum offload requires the sending of optional data with an encapsulated packet. This data is a pair of checksum start and checksum offset values. More than one offloaded checksum could be supported if multiple pairs are sent.

The logical data format for remote checksum offload is:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Checksum start         |        Checksum offset        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- o Checksum start: starting offset for checksum computation relative to the start of the encapsulated packet. This is typically the offset of a transport header (e.g. UDP or TCP).

- o Checksum offset: Offset relative to the start of the encapsulated packet where the derived checksum value is to be written. This typically is the offset of the checksum field in the transport header (e.g. UDP or TCP).

Support for remote checksum offload with specific encapsulation protocols is outside the scope of this document, however any encapsulation format that supports some reasonable form of optional meta data should be amenable. In Generic UDP Encapsulation [GUE] this would entail defining an optional field, in Geneve [GENEVE] a TLV would be defined, for NSH [NSH] the meta data can either be in a service header or within a TLV. In any scenario, what the offsets in the meta data are relative to must be unambiguous.

3.2 Transmit operation

The typical actions to set remote checksum offload on transmit are:

1) Transport layer creates a packet and indicates in internal packet meta data that checksum is to be offloaded to the NIC (normal transport layer processing for checksum offload). The checksum field is populated with the bitwise not of the checksum of the pseudo header or zero as appropriate.

2) Encapsulation layer adds its headers to the packet including the offload meta data. The start offset and checksum offset are set accordingly.

3) Encapsulation layer arranges for checksum offload of the outer header checksum (e.g. UDP).

4) Packet is sent to the NIC. The NIC will perform transmit checksum offload and set the checksum field in the outer header. The inner header and rest of the packet are transmitted without modification.

3.3 Receiver operation

The typical actions a host receiver does to support remote checksum offload are:

1) Receive packet and validate outer checksum following normal processing (e.g. validate non-zero UDP checksum).

2) Deduce full checksum for the IP packet. This is directly provided if device returns the packet checksum in CHECKSUM_COMPLETE. If the device returned CHECKSUM_UNNECESSARY, then the complete checksum can be trivially derived as either zero (GRE) or the bitwise not of the outer pseudo header (UDP).

3) From the packet checksum, subtract the checksum computed from the start of the packet (outer IP header) to the offset in the packet indicted by checksum start in the meta data. The result is the deduced checksum to set in the checksum field of the encapsulated transport packet.

   In pseudo code:

```
csum: initialized to checksum computed from start (outer IP
      header) to the end of the packet
start_of_packet: address of start of packet
encap_payload_offset: relative to start_of_packet
csum_start: value from meta data
checksum(start, len): function to compute checksum from start
      address for len bytes

csum -= checksum(start_of_packet, encap_payload_offset +
                 csum_start)
```

4) Write the resultant checksum value into the packet at the offset provided by checksum offset in the meta data.

   In pseudo code:

```
csum_offset: offset of checksum field

*(start_of_packet + encap_payload_offset +
  csum_offset) = csum
```

5) Checksum is verified at the transport layer using normal processing. This should not require any checksum computation over the packet since the complete checksum has already been provided.

3.4 Interaction with TCP segmentation offload

Remote checksum offload may be useful with TCP Segmentation Offload (TSO) in order to avoid host checksum calculations at the receiver. This can be implemented on a transmitter as follows:

1) Host stack prepares a large segment for transmission including adding of encapsulation headers and the remote checksum option which refers to the encapsulated transport checksum in the large segment.

2) TSO is performed by the device taking encapsulation into account. The outer checksum is computed and written for each packet. The inner checksum is not computed, and the encapsulation header (including checksum meta data) is replicated for each packet.

3) At the receiver remote checksum offload processing occurs as normal for each packet.

4 Security Considerations

Remote checksum offload should not impact protocol security.

5 IANA Considerations

There are no IANA considerations in this specification. The remote checksum offload meta data may require an option number or type in specific encapsulation formats that support it.

6 References

6.1 Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, March 2000.

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

6.2 Informative References

[RFC1071] Braden, R., Borman, D., and C. Partridge, "Computing the Internet checksum", RFC1071, September 1988.

[RFC1624] Rijsinghani, A., Ed., "Computation of the Internet Checksum via Incremental Update", RFC1624, May 1994.

[RFC1936] Touch, J. and B. Parham, "Implementing the Internet Checksum in Hardware", RFC1936, April 1996.

[GUE] Herbert, T., Yong, L, and Zia, O., "Generic UDP Encapsulation". draft-ietf-nvo3-gue-02

[GENEVE] Gross, J. and Gango, I., "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-01, January 1, 2016

[NSH] Quinn, P. and Elzur, U., "Network Service Header", draft-ietf-sfc-nsh-02.txt, January 19,2016

[LOC] Cree, E. Checksum Offloads in the Linux Networking Stack, Linux documentation: Documentation/networking/checksum-offloads.txt

Authors' Addresses

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA
US

EMail: tom@herbertland.com

T. Herbert
Facebook

February 29, 2016

Remote checksum offload for VXLAN
draft-herbert-vxlan-rco-01

Abstract

This specification describes remote checksum offload for VXLAN. Remote checksum offload is a mechanism that provides checksum offload of transport checksums in encapsulated packets using rudimentary offload capabilities found in most Network Interface Card (NIC) devices. The outer UDP checksum is enabled on transmit and, with some additional meta data, a receiver is able to deduce the checksum to be set in an encapsulated packet. Effectively this offloads the computation of the inner checksum which can be a significant performance optimization. Enabling the UDP checksum has the additional advantage that it covers more of the packet including the IP pseudo header and virtual network identifier.

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

Copyright and License Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1 Introduction 3
2 Remote checksum offload for VXLAN 3
2.1 Header format 3
2.2 Transmitter operation 4
2.3 Receiver operation 4
3 Security Considerations 6
4 IANA Considerations 6
5 References 6
5.1 Normative References 6
5.2 Informative References 6
Authors' Addresses 6

1 Introduction

Remote checksum offload is a mechanism that uses rudimentary NIC offload features to support offloading checksum calculation of encapsulated packets. The background and motivation for remote checksum offload is presented in [RCO].

In this specification we describe remote checksum offload for VXLAN [RFC7348]. In this design the UDP [RFC0768] checksum is enabled on transmit, and optional data conveyed in the VXLAN header specifies the location of the checksum field being offloaded and its starting point for computation. Upon receipt, after the UDP checksum is verified, the receiver sets the offloaded checksum field per the computed packet checksum and the data in the header.

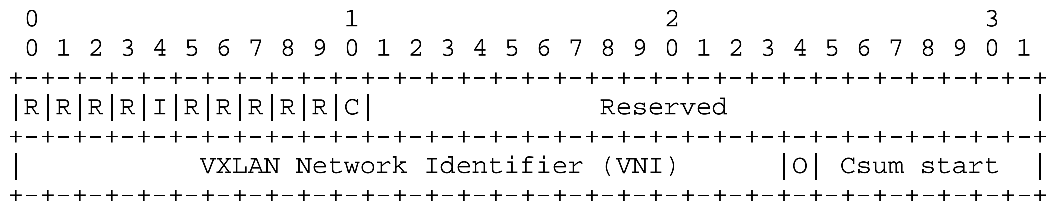
This design should also be compatible with VXLAN-GPE [VXLANGPE].

2 Remote checksum offload for VXLAN

This section describes remote checksum offload for VXLAN.

2.1 Header format

VXLAN header with remote checksum data:



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|R|R|R|I|R|R|R|R|R|C|              Reserved                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           VXLAN Network Identifier (VNI)      |O| Csum start  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- C bit: Remote checksum offload bit. When set indicates that the remote checksum offload data is present.

- O bit: Offset bit. Indicates the checksum offset relative to checksum start. Two offsets are supported corresponding to TCP [RFC0793] and UDP [RFC0768].

  O = 1 indicates checksum offset is checksum start + 6 (UDP)

  O = 0 indicates checksum offset is checksum start + 16 (TCP)

- Csum start: Checksum start divided by two. Checksum start is relative to the the first byte of the encapsulated packet. Note that only even offsets are supported and that the maximum value is 254. This typically refers to the offset of a transport

header.

The remote checksum data is encoded within the eight reserved bits of the VXLAN header that follow the VNI. A flag bit is allocated to indicate the presence of the remote checksum data.

2.2 Transmitter operation

The typical actions to set remote checksum offload on transmit are:

1) Transport layer creates a packet and indicates in internal packet meta data that checksum is to be offloaded to the NIC (normal transport layer processing for checksum offload). The checksum field is populated with the bitwise "not" of the checksum of the pseudo header.

2) VXLAN header is added to the packet to do encapsulation. If the transport checksum is for UDP or TCP, checksum start is even, and checksum start relative to start of the payload is <= 254, then remote checksum offload may be used. To set remote checksum offload the C bit is set, the O bit is set for a UDP offset or cleared for a TCP offset, and checksum start value divided by two is set in the csum start field.

3) Encapsulation layer arranges for NIC checksum offload of the outer UDP header checksum. This supersedes the settings to offload the inner packet’s transport checksum.

4) Packet is sent to the NIC. The NIC will perform transmit checksum offload and set the checksum field in the outer UDP header. The inner header and rest of the packet are transmitted without modification.

2.3 Receiver operation

The typical actions a VXLAN receiver does to support remote checksum offload are:

1) Receive packet and validate outer checksum following normal processing (ie. validate non-zero UDP checksum).

2) Deduce full checksum for the IP packet. This is directly provided if a device returns the packet checksum in checksum-complete or checksum-unnecessary conversion can be done.

3) If the C bit is set, remote checksum offload is enabled. Checksum start is csum start value times two. If O bit is set then checksum offset is checksum start + 6, else it is checksum

start + 16.

4) From the packet checksum, subtract the checksum computed from the start of the packet (outer IP header) to the offset in the packet indicted by checksum start. The result is the deduced checksum to set in the checksum field of the encapsulated transport packet.

5) Write the resultant checksum value into the packet at the offset provided by checksum offset.

6) Adjust the saved packet checksum to account for changing the checksum field within the packet.

7) Checksum is verified at the transport layer using normal processing. This should not require any checksum computation over the packet since the complete checksum has already been provided.

Steps 3,4,5, and 6 in pseudo code:

```
packet_csum: checksum computed by receiver covering the start
  of the packet (outer IP header) to the end of the packet

start_of_packet: memory address of start of packet

offset_encap_payload: offset of encapsulation payload relative
  to start_of_packet

csum_start: value of csum start field

o_bit: value of the O bit

checksum(start, len): function to compute checksum from start
  address for len bytes

// Derive the start and offset values
start = csum_start * 2
if (o_bit)
    offset = start + 6
else
    offset = start + 16

// Compute packet checksum starting from checksum start value
// (1's complement arithmetic)
csum = packet_csum - checksum(start_of_packet,
                             offset_encap_payload + start)
```

```
        // Set derived checksum in the checksum field
        old = *(start_of_packet + offset_encap_payload + offset)
        *(start_of_packet + offset_encap_payload + offset) = csum

        // Adjust packet checksum (1's complement arithmetic)
        packet_csum += (csum - old)
```

3 Security Considerations

Remote checksum offload should not impact protocol security.

4 IANA Considerations

There are no IANA considerations in this specification. Remote checksum offload requires a one VXLAN reserved bit and use of the eight reserved bits after the VNI.

5 References

5.1 Normative References

[RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, August 2014, <http://www.rfc-editor.org/info/rfc7348>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

5.2 Informative References

[RCO] Herbert T., "Remote checksum offload", draft-herbert-remotecsumoffload-02.

[VXLANGPE] Quinn P. and et al., "Generic Protocol Extension for VXLAN", draft-quinn-vxlan-gpe-04.txt

Authors' Addresses

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA

US

EMail: tom@herbertland.com