Remote checksum offload for encapsulation
draft-herbert-remotecsumoffload-01

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) 2014 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.

Abstract

This specification describes remote checksum offload, which 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 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 in an 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.

Table of Contents

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,

layer 4 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 networking 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, 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 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. It is conceivable that a NIC could provide offload for more checksums by defining more than one checksum start, checksum offset pair in the transmit descriptor.

2.2.2 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 pseudo header checksum 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 for 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.

2.3.3 Checksum-unnecessary conversion

If a device returns checksum-unnecessary for a non-zero checksum, the checksum-complete value can easily be derived as the bitwise "not" of the pseudo header checksum. This is useful in the case that the device has verified the outermost checksum of the packet, and there are checksums in an encapsulated packet to be verified.

3 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 Meta data format

Remote checksum offload requires the sending of meta 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.

Remote checksum offload will typically be implemented as a remote checksum option in the encapsulation headers. Any encapsulation format that allows optional data for extensibility should be able to support remote checksum offload. The format of the remote checksum offload option is diagrammed below.

```
 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 payload. This is typically the offset of a transport header (e.g. UDP or TCP).

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

3.2 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 or zero as appropriate.

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

3) Encapsulation layer arranges for checksum offload of the outer header checksum (e.g. UDP). 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 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 or checksum-unnecessary conversion can be done.

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 remote checksum offload option. The result is the deduced checksum to set in the checksum field of the encapsulated transport packet.

4) Write the resultant checksum value into the packet at the offset provided by checksum offset in the remote checksum offload option.

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

6) 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, and 5 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, csum_offset: values from remote checksum offload
  option

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

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

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

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

3.4 Interaction with TCP segmentation offload

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

1) Host stack prepares a large segment for transmission including 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 Remote checksum offload for Generic UDP Encapsulation

Remote checksum offload in Generic UDP Encapsulation [GUE] is supported with the addition of a remote checksum option. The GUE

header format below illustrates remote checksum option as a private field.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Source port            |      Destination port         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Length              |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0x0|C|   Hlen  |  Proto/ctype  |             Flags           |P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      Fields (optional)                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|                  Private flags(optional)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Checksum start         |       Checksum offset         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                  Private fields (optional)                    ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Pertinent fields are described below:

- Hlen: GUE header length. The offset of the encapsulated payload is Hlen * 4 + 4.

- P bit: Set to one to indicate presence of private options

- R bit: Private flag bit that indicates presence of the remote checksum option. Remote checksum offload is four bytes in length

- Checksum start: Offset of start of checksum computation for remote checksum offload. This is relative to the encapsulated payload whose offset is provided by Hlen.

- Checksum offset: Offset to write the checksum which is computed by the receiver. This is relative to the encapsulated payload whose offset is provided by Hlen.

5 Security Considerations

Remote checksum offload should not impact protocol security.

6 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.

7 References

7.1 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.

7.2 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]   Generic UDP Encapsulation draft-herbert-gue-02

Authors' Addresses

Tom Herbert
Google
1600 Amphitheatre Parkway
Mountain View, CA

EMail: therbert@google.com