Internet Draft   T. Herbert
<draft-herbert-gue-00.txt>   Google
Category: Experimental
Expires June 2014   December 20, 2013

Generic UDP Encapsulation
<draft-herbert-gue-00.txt>

Status of this Memo

Copyright Notice

Abstract

This specification describes Generic UDP Encapsulation (GUE), which is a scheme for using UDP to encapsulate packets of arbitrary IP protocols for transport across layer 3 networks. By encapsulating packets in UDP, specialized capabilities in networking hardware for efficient handling of UDP packets can be leveraged. GUE specifies basic encapsulation methods upon which higher level constructs, such tunnels and overlay networks, can be constructed.

Table of Contents

1. Introduction

This specification describes a general method for encapsulating packets of arbitrary IP protocols within User Datagram Protocol (UDP) [RFC0768] packets. Encapsulating packets in UDP facilitates efficient

transport across networks. Networking devices widely provide protocol specific processing and optimizations for UDP (as well as TCP) packets. Packets for atypical IP protocols (those not usually parsed by networking hardware) can be encapsulated in UDP packets to maximize deliverability and to leverage flow specific mechanisms for routing and packet steering.

Hardware devices commonly perform hash computations on packet headers to classify packets into flows or flow buckets. Flow classification is done to support load balancing (statistical multiplexing) of flows across a set of networking resources. Examples of such load balancing techniques are Equal Cost Multipath routing (ECMP), port selection in Link Aggregation, and NIC device Receive Side Scaling (RSS). Hashes are usually either a three-tuple hash of IP protocol, source address, and destination address; or a five-tuple hash consisting of IP protocol, source address, destination address, source port, and destination port. Typically, networking hardware will compute five-tuple hashes for TCP and UDP, but only three-tuple hashes for other IP protocols. Since the five-tuple hash provides more granularity, load balancing can be finer grained with better distribution. When a packet is encapsulated with GUE, the source port in the outer UDP packet is set to reflect the flow of the inner packet. When a device computes a five-tuple hash on the outer UDP/IP header of a GUE packet, the resultant value classifies the packet per its inner flow.

2. Packet formats

The payload of a UDP packet destined to a GUE port starts with a GUE header. If a packet is being encapsulated it immediately follows the GUE header.

2.1. GUE header preamble

The first byte of the GUE packet header contains a packet type and header length.

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|  Type |  Hlen |
+-+-+-+-+-+-+-+-+
```

Contents are:

o Type: type of header. The rest of the fields in the header are defined based the type.

o Hlen: Length in 32-bit words of the GUE header, including

        optional fields but not the first four bytes of the header.
        Computed as (header_len - 4) / 4. All GUE headers are a multiple
        of four bytes in length.

2.2. GUE encapsulation header

The GUE encapsulation header is used to encapsulate packets for various IP protocols. Encapsulation with a GUE header has the general format:

```
   +-------------------------------+
   |                               |
   |        UDP/IP header          |
   |                               |
   |-------------------------------|
   |                               |
   |         GUE Header            |
   |                               |
   |-------------------------------|
   |                               |
   |     Encapsulated packet       |
   |                               |
   +-------------------------------+
```

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

The UDP and GUE encapsulation header format 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Source port            |      Destination port         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Length              |           Checksum            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  0x0  |  Hlen |   Protocol    |V|R|R|R|R|R|R|R|R|R|R|R|R|R|P|P|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |               Virtual network ID (optional)                   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                   Private fields (optional)                   ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The contents of the UDP header are:

- o Source port (inner flow identifier): This should be set to a value that represents the encapsulated flow. The properties of the inner flow identifier are described below.

- o Destination port: The GUE assigned port number, XXXX.

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

- o Checksum: Either the standard UDP checksum or zero indicating no checksum calculated. Zero checksum is recommended.

The GUE header consists of:

- o Type: Set to 0x0 to indicate GUE encapsulation header.

- o Hlen: Length in 32-bit words of the GUE header, including optional fields but not the first four bytes of the header. Computed as (header_len - 4) / 4. The length of the encapsulated packet is determined from the UDP length and the Hlen: encapsulated_packet_length = UDP_Length - 8 - GUE_Hlen.

- o Protocol: IP protocol number for the next header. The next header begins at the offset provided by Hlen.

- o 'R' Reserved flag. Must be set to zero for sending.

- o 'V' Virtualization flag. Indicates presence of the Virtual Network Identifier (VNID) field. The VNID is used to tunnel layer 2 or layer 3 packets for network virtualization. Use and semantics of this field should be defined in separate documents.

- o 'P' Private flag. Indicates flags reserved for private use (as per private use policy specified in [RFC2434]). These flags may indicate the presence of private fields. These flags can only be used between a sender and a receiver that have agreement as to their meaning.

- o Virtual network ID (4 octets): Used in network virtualization to identify the virtual network that packet was sent on. Only present if virtualization bit is set.

- o Private fields: An implementation may define private fields that are present when a corresponding private bit is set. A private field must have a length which is a multiple of four bytes, and must be correctly accounted for in the GUE header length.

3. Operation

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

```
   +---------------+                       +---------------+
   |               |                       |               |
   |   Server 1    |                       |    Server 2   |
   |               |                       |               |
   +---------------+                       +---------------+
          |                                        ^
          V                                        |
   +---------------+   +---------------+   +---------------+
   |               |   |               |   |               |
   | Encapsulator  |-->|    Layer 3    |-->| Decapsulator  |
   |               |   |    Network    |   |               |
   +---------------+   +---------------+   +---------------+
```

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

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

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

3.1. Encapsulator operation

Encapsulators create encapsulation headers, set the source port to the inner flow identifier, set flags and optional fields in the GUE header, and forward packets to a decapsulator.

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

If an encapsulator is tunneling packets, that is encapsulating packets of layer 2 or layer 3 protocols (e.g. EtherIP, IPIP, ESP tunnel mode), it should follow standard conventions for tunneling of one IP protocol over another. Diffserv interaction with tunnels is described in [RFC2983], ECN propagation for tunnels is described in [RFC6040].

3.2. Decapsulator operation

A decapsulator performs decapsulation of GUE packets. A decapsulator is addressed by the outer destination IP address of a GUE packet. The decapsulator validates packets, including fields of the GUE header. If a packet is acceptable, the UDP and GUE headers are removed and the packet is resubmitted for IP protocol processing.

If a decapsulator receives a GUE packet with an unknown flag, bad header length (too small for included optional fields), or an otherwise malformed header, it must drop the packet and may log the event. No error message is returned back to the encapsulator.

3.3. Router and switch operation

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

A router should not modify a GUE header when forwarding a packet. It may encapsulate a GUE packet in another GUE packet, for instance to implement a network tunnel. In this case the router takes the role of an encapsulator, and the corresponding decapsulator is the logical endpoint of the tunnel.

3.4. Middlebox and NAT interactions

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

In certain instances a middlebox may infer bidirectional connection semantics to a UDP flow. For instance a stateful firewall may create a five-tuple rule to match flows on egress, and a corresponding five-tuple rule for matching ingress packets where the roles of source and destination are reversed for the IP addresses and UDP port numbers.

NAT for UDP assumes bidirectional connection semantics.

GUE primarily assumes unidirectional flow properties, there is no necessary correspondence between the UDP ports of GUE packet for encapsulated flows in different directions. GUE could be extended to provide bidirectional semantics, however that is outside the scope of this document.

3.5. UDP checksum

GUE packets should be sent with a zero checksum if the encapsulated packet contains its own checksum or can be checked with some alternate means. Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums [RFC6936] provides analysis and motivation of sending zero checksums when using UDP as an encapsulation protocol.

If a receiver receives a GUE packet with a non-zero checksum, it must perform normal UDP checksum verification.

3.6. MTU and fragmentation issues

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

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

4. Inner flow identifier properties

4.1. Flow classification

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

To support flow classification, the source port of the UDP header in GUE is set to a value that maps to the inner flow. This is referred to as the inner flow identifier. The inner flow identifier is set by the encapsulator; it can be computed on the fly based on packet contents or retrieved from a state maintained for the inner flow.

Examples of deriving an inner flow identifier are:

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

- o If the encapsulated packet is an AH transport mode packet with TCP as next header, the inner flow identifier could be a hash over a three-tuple: TCP protocol and TCP ports of the encapsulated packet.

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

The five-tuple hash commonly used to identify a flow in UDP will cover the outer source address, destination address, source port (inner flow identifier), and destination port. These values should be mostly persistent for the lifetime of an encapsulated flow, only changing infrequently (at most once every thirty seconds).

4.2. Inner flow identifier properties

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

- o The value set in the source port should be within the ephemeral port range. IANA suggest this range to be 49152 to 65535, where the high order two bits of the port are set to one. This provides fourteen bits for the inner flow identifier value.

- o The inner flow identifier should have a uniform distribution across encapsulated flows.

- o An encapsulator may occasionally change the inner flow identifier used for an inner flow per its discretion (for security, route selection, etc). Changing the value should happen no more than once every thirty seconds.

- o Decapsulators, or any networking devices, should not attempt any interpretation of the inner flow identifier, nor should they attempt to reproduce any hash calculation. They may use the value to match further receive packets for steering decisions, but cannot assume that the hash uniquely or permanently identifies a flow.

- o Input to the inner flow identifier is not restricted to ports and addresses; input could include flow label from an IPv6

packet, SPI from an ESP packet, or other flow related state in the encapsulator that is not necessarily conveyed in the packet.

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

5. Motivation for GUE

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

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

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

GUE has the following discriminating features:

- UDP encapsulation leverages specialized network device processing for efficient transport. The semantics for using the UDP source port as an identifier for an inner flow are defined.

- GUE permits encapsulation of arbitrary IP protocols, which includes layer 2 3, and 4 protocols. This potentially allows nearly all traffic within a data center to be normalized to be either TCP or UDP on the wire.

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

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

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

- Private flags and fields allow local customization and experimentation while being compatible with processing in network nodes (routers and middleboxes).

- GUE can provide encapsulation for a virtual network that provides layer 3 connectivity. In contrast, VXLAN and NVGRE are defined to only provide layer 2 services (encapsulation of Ethernet).

- GUE defines a 32 bit virtual networking identifier (in contrast to 24 bit values defined for VXLAN and NVGRE). This facilitates hierarchical assignment, local flag definitions in the identifier, and potentially obfuscation of the identifier on the wire.

6. Security Considerations

Encapsulation of IP protocols within GUE should not increase security risk, nor provide additional security in itself. As suggested in section 3 the source port for of UDP packets in GUE should be randomly seeded to mitigate some possible denial service attacks.

7. IANA Considerations

A well known UDP port number assignment for GUE will be requested.

8. References

8.1. Normative References

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

[RFC2434]Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, October 1998.

[RFC2983]Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000.

[RFC6040]Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, November 2010.

[RFC6936]Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013.

[RFC4459]Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, April 2006.

8.2. Informative References

[RFC2003]Perkins, C., "IP Encapsulation within IP", RFC 2003, October 1996.

[RFC3948]Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

[RFC6830]Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, January 2013.

[RFC3378]Housley, R. and S. Hollenbeck, "EtherIP: Tunneling Ethernet Frames in IP Datagrams", RFC 3378, September 2002.

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

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

[RFC2661]Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G.,

and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, August 1999.

[NVGRE] NVGRE: Network Virtualization using Generic Routing Encapsulation draft-sridharan-virtualization-nvgre-03

[VXLAN] VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks draft-mahalingam-dutt-dcops-vxlan-06

[TCPUDP] Encapsulation of TCP and other Transport Protocols over UDP draft-cheshire-tcp-over-udp-00

[GREUDP] Generic UDP Encapsulation for IP Tunneling draft-yong-tsvwg-gre-in-udp-encap-02

[GUT] Generic UDP Tunnelling (GUT) draft-manner-tsvwg-gut-02.txt

Appendix A: NIC processing for GUE

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

A.1. **Receive multi-queue**

Contemporary NICs support multiple receive descriptor queues (multi-queue). Multi-queue enables load balancing of network processing for a NIC across multiple CPUs. On packet reception, a NIC must select the appropriate queue for host processing. Receive Side Scaling is a common method which uses the flow hash for a packet to index an indirection table where each entry stores a queue number. Flow Director and Accelerated Receive Flow Steering (aRFS) allow a host to program the queue that is used for a given flow which is identified either by an explicit five-tuple or by flow hash.

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

By default, UDP support may be disabled in NICs to avoid out of order reception that can occur when UDP packets are fragmented. As discussed above, fragmentation of GUE packets should be mitigated by fragmenting packets before entering a tunnel, path MTU discovery in higher layer protocols, or operator adjusting MTUs. Other UDP traffic

may not implement such procedures to avoid fragmentation, so enabling UDP support in the NIC should be a considered tradeoff during configuration.

A.2. Checksum offload

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

A.2.1. Transmit checksum offload

NICs may provide a protocol agnostic method to offload transmit checksum that can be used with GUE. In this method the host provides checksum related parameters in a transmit descriptor for a packet. These parameters include the starting offset of data to checksum, the length of data to checksum, and the offset in the packet where the computed checksum is to be written. The host may seed the checksum with for data not covered by the NIC computation (the checksum of the pseudo header for instance).

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

NICs typically can offload only one transmit checksum per packet, so simultaneously offloading both an inner transport packet's checksum and the outer UDP checksum is likely not possible. In this case setting UDP checksum to zero (per above discussion) and offloading the inner transport packet checksum is desirable.

A.2.2. Receive checksum offload

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

A.3. Transmit Segmentation Offload

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

This is useful to reduce CPU load on host.

The process of TSO could be generalized as:

1. Split the TCP payload into segments which will allow less than MTU size packets.

2. For each segment, replicate the TCP header and all preceding headers of the original packet.

3. For each protocol header set any payload length fields to reflect the length for the segment.

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

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

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

A.4. Large Receive Offload

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

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

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

Appendix B: Privileged ports

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

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

Appendix C: Inner flow identifier as a route selector

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

Authors' Addresses

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