Internet Draft
<draft-herbert-gue-02.txt>
Category: Experimental
Expires April 2015

T. Herbert
Google
L. Yong
Huawei USA

October 24, 2014

Generic UDP Encapsulation
<draft-herbert-gue-02.txt>

Status of this Memo

This Internet-Draft will expire on June 24, 2014.

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 for network virtualization, can be constructed. GUE is extensible by allowing optional meta data as part of the encapsulation, and is generic in that it can encapsulate packets of various IP protocols.

Table of Contents

1. Introduction 3
2. Packet formats 4
   2.1. GUE header preamble 4
   2.2. GUE header 5
   2.3. Flags and optional fields 6
3. Message types 7
   3.1. Control messages 7
   3.2. Data messages 7
4. Operation 7
   4.1. Network tunnel encapsulation 8
   4.2. Transport layer encapsulation 8
   4.3. Encapsulator operation 8
   4.4. Decapsulator operation 9
   4.5. Router and switch operation 9
   4.6. Middlebox interactions 9
   4.7. NAT 10
   4.8. UDP checksum 10
   4.9. MTU and fragmentation issues 10
   4.10 Congestion control 11
5. Inner flow identifier properties 11
   5.1. Flow classification 11
   5.2. Inner flow identifier properties 12
6. Motivation for GUE 12
7. Security Considerations 14
   7.1. GUE security fields 14
   7.2. GUE and IPsec 14
8. IANA Considerations 15
9. References 16
   9.1. Normative References 16
   9.2. Informative References 16
Appendix A: NIC processing for GUE 17
   A.1. Receive multi-queue 17
   A.2. Checksum offload 18

      A.2.1. Transmit checksum offload 18
      A.2.2. Receive checksum offload 19
    A.3. Transmit Segmentation Offload 19
    A.4. Large Receive Offload 20
  Appendix B: Privileged ports 21
  Appendix C: Inner flow identifier as a route selector 21
  Appendix D: Hardware protocol implementation considerations . . . 21
  Authors' Addresses 22

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.

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

2. Packet formats

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

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

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

2.1. GUE header preamble

The first byte of the GUE header provides the header type, indicator of a control or data message, and header length:

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

Contents are:

- Ver: GUE protocol version. The rest of the fields after the preamble are defined based on the version. This field is three bits allowing four possible values.

- Control flag: When set indicates a control message, not set indicates a data message.

- 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. Maximum header length is 132 bytes.

2.2. GUE header

The header format for version 0x0 of GUE in UDP 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|C|   Hlen  |  Proto/ctype  |             Flags           |P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Fields (optional)                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Private flags(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 (length of UDP header and payload).

- o Checksum: Standard UDP checksum.

The GUE header consists of:

- o Preamble byte: Version number (0x0), C bit, and header length.

- o Proto/ctype: When the C bit is set this field contains a control message type for the payload. When C bit is not set, the field holds the IP protocol number for the encapsulated packet in the payload. The control message or encapsulated packet begins at

the offset provided by Hlen.

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

- 'P' Private flag. Indicates presence of private flags option in the optional fields.

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

- Private flags: An optional field indicated by the P bit. This field is set of private flags which may in turn indicate presence of private fields.

- Private fields: Optional fields that are present when a corresponding bit in the private flags 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.

2.3. Flags and optional fields

Flags and associated optional fields are the primary mechanism of extensibility in GUE. There are sixteens flag bits in the GUE header, one of which is reserved to indicate the presence of a private flags optional field. New flags will be defined in other specifications.

A flag may indicate presence of optional fields. The size of an optional field indicated by a flag must be fixed.

The private flags optional field is comprised of thirty-two flag bits. Private flags retain the same properties of the regular header flags for parsing. The semantics of the private flags are specific to an implementation or site that defines them.

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

Optional fields are placed in order of the flags. New flags should be allocated from high to low order bit contiguously without holes. Flags allow random access, for instance to inspect the field corresponding to the Nth flag bit, an implementation only considers the previous N-1 flags to determine the offset. Flags after the Nth flag are not pertinent in calculating the offset of the Nth flag.

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

3. Message types

3.1. Control messages

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

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

Control messages can be defined for OAM type messages. For instance, an echo request and corresponding echo reply message may be defined to test for liveness.

3.2. Data messages

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

Data messages are a primary means of encapsulation and can be used to create tunnels for overlay networks.

4. 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.

4.1. Network tunnel encapsulation

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

4.2. Transport layer encapsulation

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.

4.3. Encapsulator operation

Encapsulators create GUE data messages, 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].

4.4. Decapsulator operation

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

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

4.5. Router and switch operation

Routers and switches should forward GUE packets as standard UDP/IP packets. The outer five-tuple should contain sufficient information to perform flow classification corresponding to the flow of the inner packet. A switch should not 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.

4.6. Middlebox 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.

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. To operate in this environment, a GUE tunnel must assume connected semantics defined by the UDP five tuple and the use of GUE encapsulation must

be symmetric between both endpoints. The source port set in the UDP header must be the destination port the peer would set for replies.

4.7. NAT

IP address and port translation can be performed on the UDP/IP headers adhering to the requirements for NAT with UDP [RFC4787]. In the case of stateful NAT, connection semantics must be applied to a GUE tunnel as described above.

When using transport mode encapsulation and traversing a NAT, the IP addresses may be changed such that the pseudo header checksum used for checksum calculation is modified and the checksum will be found invalid at the receiver. To compensate for this, A GUE option can be added which contains the checksum over the source and destination addresses when the packet is transmitted. Upon receiving this option, the delta of the pseudo header checksum is computed by subtracting the checksum over the source and and destination addresses from the checksum value in the option. The resultant value is then added into checksum calculation when validating the inner transport checksum.

4.8. UDP checksum

When the outer IP protocol is IPv6, the UDP checksum must be set on transmission. A zero checksum may be used only if all the provisions of RFC6936 ("Applicability of Zero UDP Checksum with IPv6") are met.

When the outer IP protocol is IPv4, the UDP checksum should be set on transmission. If the GUE header contains data that when corrupted can lead to misdirecting the packet to an incorrect receiver (for instance virtual network ID), then the UDP checksum must be set on transmission unless applicable conditions (those which would pertain to IPv4) of RFC6936 are met.

A decapsulator must always validate non-zero UDP checksums for GUE packets following normal UDP checksum verification procedures. By default a decapsulator must drop IPv6 UDP packets with a zero checksum unless configured otherwise.

4.9. 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 en5404]capsulation 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.

4.10 Congestion control

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

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

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

DCCP may be used to provide congestion control for encapsulated flows. In this case, the protocol stack for an IP tunnel may be IP-GUE-DCCP-IP. Alternatively, GUE can be extended to include congestion control (related date carried in GUE optional fields). Congestion control mechanisms will be elaborated in other specifications.

5. Inner flow identifier properties

5.1. 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:

- 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.

- 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.

- 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.

5.2. 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:

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

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

- 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.

- 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.

- 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.

6. 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:

o 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.

o 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.

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

o GUE is extensible. New flags and optional fields can be defined.

o 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.

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

o GUE includes both data messages (encapsulation of packets) and

control messages (such as OAM).

7. Security Considerations

Encapsulation of IP protocols within GUE should not increase security risk, nor provide additional security in itself. As suggested in section 5 the source port for of UDP packets in GUE should be randomly seeded to mitigate some possible denial service attacks.

GUE is most useful when it is in the outermost header of a packet which allows for flow hash calculation as well as making GUE header data (such as virtual network identifier) visible to switches and middleboxes. GUE must be amenable to encapsulating (and being encapsulated within) IPsec. Also, we allow provisions to secure the GUE header itself without external protocol.

7.1. GUE security fields

Security fields should be used to provide integrity and authentication of the GUE header. Security negotiation (interpretation of security field, key management, etc.) is expected to be negotiated out of band between two communicating hosts. Security fields will be specified in future documents.

7.2. GUE and IPsec

GUE may be used to encapsulate IPsec packets. This allows the benefits of deriving a flow hash for the inner, potentially encrypted, packet. In this case the protocol stack may be:

```
+-------------------------------+
|                               |
|        UDP/IP header          |
|                               |
|-------------------------------|
|                               |
|         GUE Header            |
|                               |
|-------------------------------|
|                               |
|     ESP/AH/private security   |
|                               |
|-------------------------------|
|                               |
|       Encapsulated packet     |
|                               |
+-------------------------------+
```

Note that the security does not cover the GUE header (does not authenticate it for instance). GUE security optional fields may be used to provide authentication or integrity of the GUE header.

8. IANA Considerations

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

9. References

9.1. 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.

[RFC4787] Audet, F., Ed., and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

9.2. 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.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[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

[MNGCB] Network Transport Circuit Breakers draft-ietf-tsvwg-circuit-breaker-00

[REMCSUM] Remote Checksum Offload draft-herbert-remotecsumoffload-00

Appendix A: NIC processing for GUE

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

A.1. Receive multi-queue

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

program the queue that is used for a given flow which is identified either by an explicit five-tuple or by the flow's hash.

GUE encapsulation should be compatible with multi-queue NICs that support five-tuple hash calculation for UDP/IP packets as input to RSS. The 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 RSS support is often disabled in NICs to avoid out of order reception that can occur when UDP packets are fragmented. As discussed above, fragmentation of GUE packets should be mitigated by fragmenting packets before entering a tunnel, path MTU discovery in higher layer protocols, or operator adjusting MTUs. Other UDP traffic may not implement such procedures to avoid fragmentation, so enabling UDP RSS support in the NIC should be a considered tradeoff during configuration.

A.2. Checksum offload

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

A.2.1. Transmit checksum offload

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

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

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

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

A.2.2. Receive checksum offload

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

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

A.3. Transmit Segmentation Offload

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

The process of TSO can be generalized as:

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

- For each created segment:

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

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

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

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

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

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

A.4. Large Receive Offload

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

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

The conservative approach to supporting LRO for GUE would be to assign packets to the same flow only if they have identical five-

tuple and were encapsulated the same way. That is the outer IP addresses, the outer UDP ports, GUE protocol, GUE flags and fields, and inner five tuple are all identical.

Appendix B: 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

An 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 the path by altering the inner flow identifier. For instance, a host may store a flow hash in its PCB for an inner flow, and may alter the value upon detecting that packets are traversing a lossy path. Changing the 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.

Appendix D: Hardware protocol implementation considerations

A low level protocol, such is GUE, is likely interesting to being supported by high speed network devices. Variable length header (VLH) protocols like GUE are often considered difficult to efficiently implement in hardware. In order to retain the important characteristics of an extensible and robust protocol, hardware vendors may practice "constrained flexibility". In this model, only certain combinations or protocol header parameterizations are implemented in hardware fast path. Each such parameterization is fixed length so that the particular instance can be optimized as a fixed length protocol. In the case of GUE this constitutes specific combinations of GUE flags, fields, and next protocol. The selected combinations would naturally be the most common cases which form the "fast path", and other combinations are assumed to take the "slow path".

In time, needs and requirements of the protocol may change which may

manifest themselves as new parameterizations to be supported in the fast path. To allow allow this extensibility, a device practicing constrained flexibility should allow the fast path parameterizations to be programmable.

Authors' Addresses

Tom Herbert
Google
1600 Amphitheatre Parkway
Mountain View, CA
US

EMail: therbert@google.com

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

EMail: lucy.yong@huawei.com