INTERNET-DRAFT
Intended Status: Informational
Expires: September 27, 2015

T. Herbert
Facebook
L. Yong
Huawei
O. Zia
Microsoft
March 26, 2015

Encapsulation Considerations for GUE
draft-herbert-gue-encap-considerations-01

Abstract

This document provides a description of how Generic UDP Encapsulation addresses the encapsulation considerations that are described in the "Encapsulation Considerations" Internet Draft.

Table of Contents

1 Introduction

This document provides a description of how Generic UDP Encapsulation (GUE) [I.D.herbert-gue] addresses the encapsulation considerations that are described in [I.D.rtg-dt-encap]. That draft is the result of a design team that was chartered by the routing area director to investigate and report on the common issues across different encapsulations.

The organization of this document follows that of the encapsulation considerations draft. There is a section for each of the main areas for consideration with encapsulation. These areas are:

- Entropy
- Next-protocol indication
- MTU and fragmentation
- OAM
- Security considerations
- QoS
- Congestion considerations
- Header protection
- Extensibility considerations
- Layering considerations
- Service model
- Hardware friendly
- Middleboxes

An additional section covers considerations specific to network virtualization.

2 Entropy

Similar to other UDP encapsulation proposals, the UDP source port of a GUE packet may be set to a value that reflects the inner flow. In GUE parlance, this value based on the inner flow identifier which is often a hash over the 5-tuple of the inner packet's headers. The UDP

source port provides fourteen bits of entropy assuming that the selected value is restricted to the ephemeral port range. The source port used to indicate a given flow may also change over the lifetime of a flow.

The outer IPv6 flow label may be used to provide additional entropy in the flow identifier when fourteen bits is insufficient.

If the UDP port (and possibly IPv6 flow label) still does not provide enough entropy in the flow classification, then deep parsing of the GUE payload may be performed. See section 14.

3 Next-protocol indication

The next protocol indication in GUE is in the proto/ctype field in the GUE header. For GUE data messages (as opposed to control messages, see section 5) the proto field holds an IP protocol number of the next header. An eight bit value is an efficient use of header space, and the lookup of IP protocol can be implemented as a simple 256 entry array (as in Linux).

The IP protocol number allows encapsulation of various layer 2 and layer 3 protocols. In particular, the most common protocols for tunnels are likely to be:

- IPv4: number 4
- IPv6: number 41
- Ethernet: via EtherIP with number 97

The encapsulated protocol may also be GRE (number 47) which allows encapsulation of protocols of any Ethertype in GUE with an additional four bytes of header.

Layer 4 protocols may also be encapsulated within GUE (e.g. ESP, UDP, ICMP, etc.). In this case the UDP and GUE encapsulating headers are considered to be inserted between IP and the transport header. In this way, the outer UDP header for GUE and the encapsulated transport header logically header share the same IP header. For an encapsulated TCP or UDP header checksum calculation, the pseudo checksum is based on the outer IP header ignoring the encapsulation headers.

4 MTU and Fragmentation

Similar to other encapsulation protocols, it is recommended in GUE that fragmentation over a tunnel is avoided by configuring tunnel MTUs and using Path MTU Discovery (PMTUD) as necessary. As described

in [I.D.rtg-dt-encap], detecting mis-configuration causing packets to be dropped due to MTU issues is desirable; having the encapsulator set the don't-fragment (DF) flag in the outer IPv4 header and logging any received ICMP "packet too big" (PTB) are compatible with GUE.

As discussed in [RFC4459] it may not be possible to avoid the need for fragmentation in a all circumstances (for instance a link in a tunnels path may have the minimum MTU for IPv6 which is 1280). To accommodate this, a fragmentation option has been defined for GUE ([I.D.gue-fragmentation]).

5 OAM

Specific OAM support for GUE (and other encapsulation protocols) has not yet been defined. Due to the extensibility model of GUE and definition of control messages, there is a lot of flexibility in how OAM may be supported. GUE includes provisions to support both active OAM (OAM specific messages) and passive OAM (measurements of data messages).

5.1 Active OAM

The GUE header includes a bit that indicates that the payload contains a control message as opposed to a data message. When this bit is set, the proto/ctype field is interpreted to be a control type. Various types of control messages may be defined, including those for OAM.

5.2 Passive OAM

Options in the GUE header may be added to permit passive OAM measurements attached to data messages. This may accomplished by using be single bits of information, or by OAM measurement fields which could contain items such as sequence numbers, timestamps, etc.

6 Security Considerations

Security is a very important consideration in GUE. This is particularly motivated by the multi-tentant use case of network virtualization where isolation between tenants is a critical requirement. The GUE security model includes both considerations to protect the headers and the whole packet. For both of these, we assume that a "pluggable" secuirty model is desirable with the assumption that stronger security may be implemented over time in response to changing threats.

6.1 Integrity and authentication of the encapsulation

Addresses, port numbers, and various elements of a GUE header may need fairly strong assurances of integrity and authentication to protect against corruption or spoofing. This requirement is readily apparent in the use case of network virtualization where ensuring the integrity and authenticity of a virtual network identifier is paramount to guaranteeing isolation between virtual networks even in the presence of users with malicious intent.

To provide for this security, an optional security field is defined in GUE ([I.D.hy-gue-4-secure-transport]). This field has three possible sizes of 64, 128, or 256 bits to allow for different levels of security. The simplest mechanism is a security cookie which is a shared value that is passed in the clear and must matched on receipt. More sophisticated mechanisms may use cryptographic hashes, nonce values, reply detection, etc.

6.2 Packet level security

As GUE is contained in an IP packet, the packet itself may be encapsulated in something like ESP or DTLS to provide security. This is straightforward, however visibility of the encapsulation is lost in the network. This is problematic, for instance, if one wanted to establish firewalls to restrict packets for a certain virtual network.

Security of a GUE payload may be accomplished by applying ESP or DTLS to the payload and encapsulating ESP within GUE. In this model the protocol stack may be something like IP|UDP|GUE|ESP|IP. The GUE next protocol would indicate ESP (number 50), and the UDP and GUE headers would be sent in the clear so that encapsulation is visible to the network. As described above, measures should be taken to ensure the integrity and authentication of addresses and GUE headers. One salient property of this method is that any bits created by an application or virtualization guest are covered by the packet level security mechanism.

7 QoS

There are no specific provisions or options to provide additional QoS facilities in GUE. The provisions of "Diffserv and Tunnels" [RFC2983] are assumed.

8 Congestion Considerations

Congestions considerations that are generically specified for tunnels would be applicable to GUE. This would include mechanisms currently being defined such as circuit breaker [I.D.cirtuit-breaker] and common ECN handling for IP tunnels.

In certain cases, specific congestion control may be necessitated beyond what generic congestion control mechanisms may provide. In particular, this may be required in a data center whose native traffic and resources have been tuned to a very specific congestion control algorithm. When third party network stacks, such as those running in a VM's guest OS, are introduced into such an environment their congestion control model may substantially conflict with that of the native traffic. If traffic and resource isolation is not feasible, the only recourse may be to force third party traffic into compliance with the native congestion control.

This can potentially be accommodated in GUE in two ways:

1) Add an additional protocol layer to the encapsulation that provides congestion control. DCCP is a candidate. In this case, the protocol stack for a encapsulated packet may look like IP|UDP|GUE|DCCP|IP.

2) Add an option to GUE which provides for congestion control. This would likely include an optional header field that would contain various values needed for congestion control-- sequence numbers, timestamps, ack numbers etc. Note that some of this information may also be in common with passive OAM data.

Extending GUE with finer grained congestion is a topic for further exploration.

9 Header Protection

As with other UDP encapsulation protocols, the UDP checksum may or may not be set transmit. The requirements for setting a zero UDP checksum with IPv6 to be compliant with [RFC6935] and [RFC6936] are enumerated in [I.D.herbert-gue]. In the case that a zero checksum is used (either for IPv4 or IPv6) the GUE specification recommends that the GUE header checksum be used (unless stronger protection such as security are present).

The GUE header checksum is a UDP-lite like option in the GUE header ([I.D.herbert-guecsum]). This checksum covers the entire GUE header and a pseudo header containing the outer IP addresses and UDP port numbers. Optionally, the checksum may cover all or part of the encapsulated GUE payload.

10 Extensibility Considerations

GUE is an extensible protocol that allows a variable length header. The extensibility mechanism in GUE is flag-fields. This is similar to the use of flags and fields in GRE, where if a flag is set a field of

a specific size is present. See section 13.1 for points about the efficiency of the flag-fields solution.

The GUE header includes fifteen bit flags in the primary header and an additional thirty-two in an extension flags field. Up to 124 bytes of optional fields may be present.

All flags are considered mandatory, in the sense that a decapsulator must drop a packet it receives with set flags that are unknown to it. This requirement ensures that new flags with non-trivial semantics can be added without breaking compatibility. A middlebox may ignore flags (see section 14).

11 Layering Considerations

GUE does not include any specific provisions for layering of encapsulations other than the fact that it can encapsulate an encapsulated packet represented by an IP protocol. As described in section 3, encapsulation of GRE may be used to encapsulate packets of an arbitrary Ethertype.

Conceptually, the GUE header could be disassociated from UDP and defined as its own IP protocol (similar to GRE being an IP protocol). In this manner GUE could effectively function as an IP extension header and layered encapsulations would essentially be equivalent to multiple extension headers.

12 Service Model

The base service model of GUE is equivalent to that of IP. Packets can be lost, reordered, duplicated, etc. To enact a more elaborate service model over GUE, such as pseudo-wire semantics or reliable tunnels, could be done as a layered encapsulation of a protocol that provides the service.

The exception to the above occurs when encapsulated traffic has the ability to negatively impact unrelated networking traffic. In this case, a service model that provides congestion control or DDOS protection is a candidate to implement within the encapsulation layer (e.g. see section 8).

13 Hardware Friendly

13.1 Switch friendliness

GUE is mostly intended to be an end to end tunneling protocol. Switches may inspect fields as input to routing operation, however they should not modify GUE headers in flight (checksum and header

security likely make that infeasible). Encapsulation for the purposes affecting routing per hop or targeting networking services are better left to protocols dedicated to those functions such as BIER or SFC.

In the case that a switch acts as tunnel endpoint (i.e. an NVE in NVO3 parlance), considerations can be made for efficient termination and processing of UDP.

GUE has the following features to be friendly in switches:

- For a given set of flags, field offsets are fixed
- The number of possible flag combinations is 2^N for N supported flags. This is a much smaller number than for TLVs which are combinatorial.
- Minimal overhead. Other than flags, there is no additional overhead associated with fields
- Flag fields are amenable to hardware parallel parsing mechanisms such as TCAM (based on the above points)
- The GUE header checksum obviates need for full packet checksum support
- Hardware support of variable flag-fields has already be demonstrated in GRE

13.2 Host friendliness

The first requirement of encapsulation in the host is that it works with existing NIC offloads. The five common offloads in question are RSS (Receive Side Scaling), TX-csum (transmit checksum offload), RX-csum (receive checksum offload), LSO (Large Segment Offload), and LRO (Large Receive Offload).

RSS is already solved by enabling RSS for UDP which is available on most NICs. This works for any UDP encapsulation that uses source port for flow entropy.

TX-csum and RX-csum offload for encapsulated checksums (no outer UDP checksum) are already generically supported by NICs that provide NETIF_HW_CSUM and CHECKSUM_UNNECESSARY (in Linux parlance). The outer UDP checksum may also be enabled, and this can be leveraged in GUE to provide checksum offload of inner transport checksums for legacy devices (via checksum-unnecessary conversion and remote checksum offload). The ability for NICs to support offload of multiple checksums in a packet may also become pertinent in time.

For LSO (TSO), the Linux stack already demonstrates a generic solution for segmentation with UDP encapsulation in GSO. This can be implemented in HW as method to provide LSO for any UDP encapsulation.

For LRO, a device needs to do deep parsing of the GUE payload. This can be accomplished by skipping over any options using the Hlen field. Packets should only be considered to match if the GUE encapsulation, including any optional fields and private data, are identical.

The implementation of GUE in a software stack is fairly straight forward and can be efficient. The UDP layer in the software stack should already handle the processing of the UDP/IP headers including that of the checksum. For the GUE headers, processing the flag-fields is likely the most difficult operation. Given the practical constraints on flag-fields, they can be processed without a loop and without the need to check lengths or duplicate fields. The check for unsupported set flags can be implemented with a simple masked comparison on the flags.

14 Middlebox Considerations

The following GUE features facilitate middlebox handling:

- The Hlen field allows middlebox to skip over optional fields to perform deep parsing
- The meaning of proto/ctype field is invariant regardless of flags
- Flag-fields permit random access for inspection
- Middleboxes are not required to understand all possible fields. A principle in GUE is that new fields cannot cause reinterpretation of old fields.

15 Network virtualization

The primary requirement for network virtualization is that a virtual network is indicated as part of the encapsulation (i.e. a virtual network identifier or VNI-ID).

GUE defines a 32 bit VNI-ID in an optional field ([I.D.hy-nvo3-gue-4-nvo]). There is no predefined structure to this value. An implemention may apply a hierarchical structure (for instance a tenant might have virtual sub-networks), as well as allocating bits to indicate class or other attributes (such as a bit indicating a trusted or untrusted virtual network).

A first order requirement of network virtualization is that isolation between virtual networks be ensured. As described in section 6, the GUE security option should be used to provide integrity and authentication.

16 References

16.1 Normative References

[I.D.rtg-dt-encap] Nordmark, E., Tian, A., Gross, J.," Hudson, J., Kreeger, L., Garg, P., Thaler, P., Herbert, T., "Encapsulation Considerations", draft-rtg-dt-encap-01.

[I.D.herbert-gue] Herbert, T., and Yong, L., "Generic UNP Encapsulation", draft-herbert-gue-03, work in progress.

16.2 Informative References

[RFC4459] Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, April 2006, <http://www.rfc-editor.org/info/rfc4459>.

[RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000, <http://www.rfc-editor.org/info/rfc2983>.

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, April 2013, <http://www.rfc-editor.org/info/rfc6935>.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013, <http://www.rfc-editor.org/info/rfc6936>.

[I.D.gue-fragmentation] Herbert, T. and Templin, F., "Fragmentation option for Generic UDP Encapsulation", draft-herbert-gue-fragmentation-00, work in progress

[I.D.hy-gue-4-secure-transport] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Secure Transport", draft-hy-gue-4-secure-transport-00, work in progress.

[I.D.tsvwg-circuit-breaker] Fairhurst, G., "Network Transport Circuit Breakers", draft-ietf-tsvwg-circuit-breaker-00

[I.D.herbert-remotecsumoffload] Herbert, T., "Remote checksum offload for encapsulation", draft-herbert-remotecsumoffload-00, work in progress

[I.D.hy-nvo3-gue-4-nvo] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Network Virtualization

Overlay", work in progress

Authors' Addresses

Tom Herbert
Facebook
Menlo Park, CA
USA

EMail: tom@herbertland.com

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

Osama Zia
Microsoft

EMail: osamaz@microsoft.com